More JSP

Advanced Topics in Java

Khalid Azim Mughal
khalid@ii.uib.no
http://www.ii.uib.no/~khalid/atij/

Version date: 2006-09-04

Overview

- Including Resources in JSP Pages using the `jsp:include` Action and the `include` Directive
- Using JavaBeans in JSP Documents using `jsp:useBean`, `jsp:getProperty`, `jsp:setProperty` Actions
- The MVC2 Architecture for Web Applications

Including Resources in JSP Pages

- Allows factoring out common resources, and thereby encourages reuse.
- Including resources at *Request Time*.

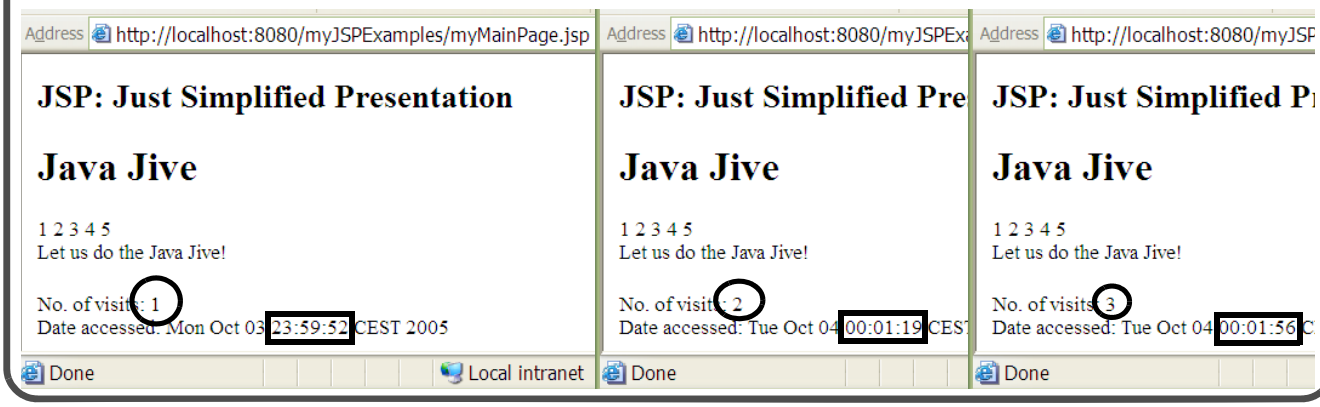
```
<jsp:include page="relative url" />
```

  - The *output* from the included resource is placed in the main page.
  - Ensures that the main page is always current with the resource.
  - The main page need not change when the included pages change.
  - The included page cannot affect the overall structure of the main page.
  - The included page uses the same request object as the main page.
  - Example: See `myJSPExamples` application: `myFooter.jsp`, `myHeader.jsp` and `myMainPage.jsp`.

```
<%-- myHeader.jsp --%>
<h2>JSP: Just Simplified Presentation</h2>

<%-- myFooter.jsp --%>
<%@ page import = "java.util.Date" %>
<%! int visitCount = 0; %>
<p>No. of visits: <%= ++visitCount %><br/>
Date accessed: <%= new Date() %>
</p>
```



```
<%-- myMainPage.jsp --%>
<html>
<head><title>Java Jive (with include action)</title></head>
<body>
<jsp:include page = "myHeader.jsp" />
<h1> Java Jive </h1>
1 2 3 4 5<br/>
Let us do the Java Jive!
<jsp:include page = "myFooter.jsp" />
</body>
</html>
```



- Including resources at *Translation Time*.

  `<%@ include file="relative url" />`

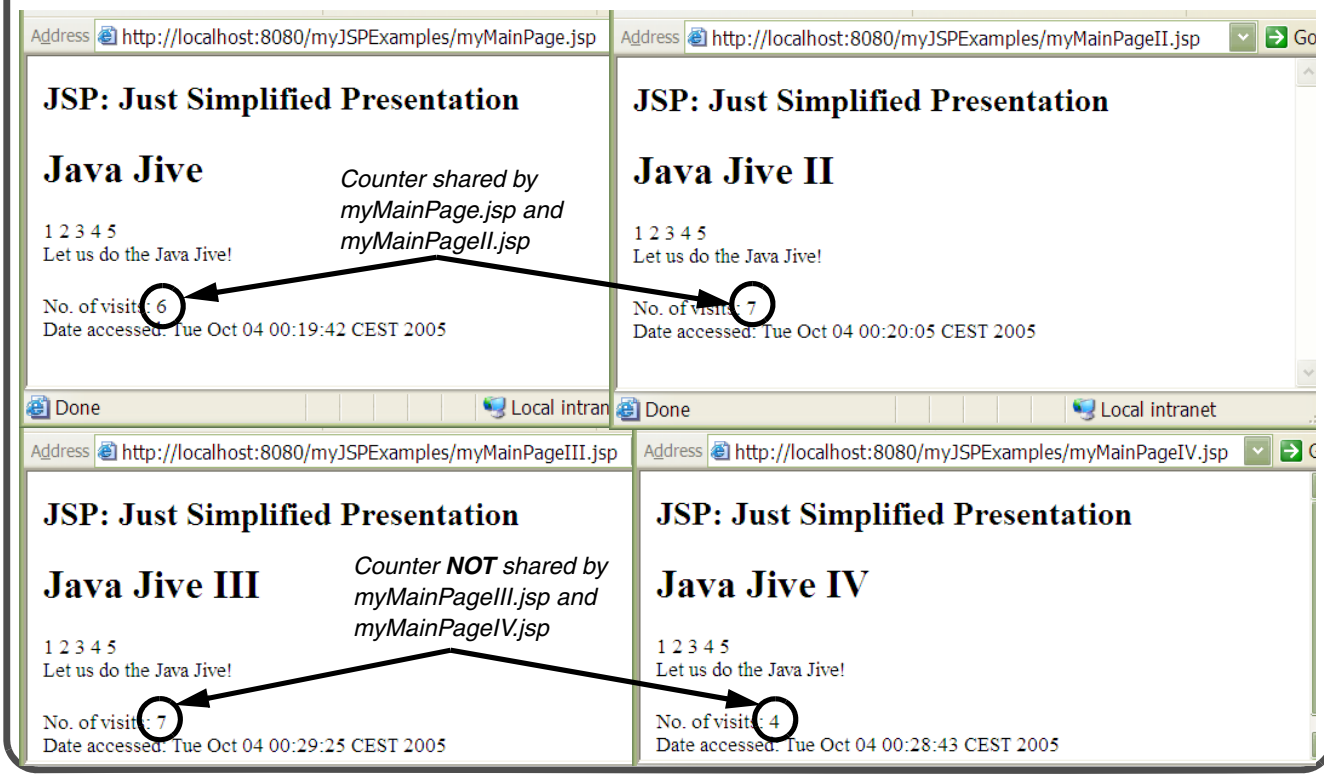
  – The *contents* of the included resource are literally inserted in the main page at translation time, before the composite JSP page is translated and compiled.
  – The page is only current with the resource at the time the page is translated.
  – The included resources can affect the main page.
  – Example: See `myJSPExamples` application: `myFooter.jsp`, `myHeader.jsp` and `myMainPageIII.jsp`.

```
<%-- myMainPageIII.jsp --%>
<html>
<head><title>Java Jive III (with include directive)</title></head>
<body>
<%@ include file = "myHeader.jsp" %>
<h1> Java Jive III</h1>
1 2 3 4 5<br/>
Let us do the Java Jive!
<%@ include file = "myFooter.jsp" %>
</body>
</html>
```


- `myMainPage.jsp` and `myMainPageIII.jsp` have the same behavior.
- What is the big deal about using one or the other?

Best Practices for including Resources

- Ensure that the composite result of the main page and the included pages form *valid content* (for example, valid HTML source) that can be interpreted *correctly* by the client.
- With an `include` action the *same* resource is *shared* by *all* main pages.
  - The `jsp:include` action is the recommended choice in most cases.
  - The `include` directive is mostly used when declarations are shared between the main page and the included pages, or when included files need to affect the main page, for example to set response headers.
    - Example: See `myJSPExamples` application: `myFooter.jsp`, `myHeader.jsp`, `myMainPage.jsp`, and `myMainPageII.jsp`.
      - Note `myMainPage.jsp`, and `myMainPageII.jsp` have identical code.
- With an `include` directive *copies* of the resource are included in *all* main pages.
  - Example: See `myJSPExamples` application: `myFooter.jsp`, `myHeader.jsp`, `myMainPageIII.jsp`, and `myMainPageIV.jsp`.
    - Note `myMainPageIII.jsp`, and `myMainPageIV.jsp` have identical code.



Interpreting URLs

- The server interprets a relative URL relative to the *current directory of the main page*.
- The server interprets an absolute URL relative to the *document root of the web application* containing the main page.

```
<%-- File: /minApp/jsp/mainpage.jsp --%>
<jsp:include page="page1.jsp" />     <%-- Includes: /minApp/jsp/page1.jsp --%>
<jsp:include page="/page1.jsp" />    <%-- Includes: /minApp/page1.jsp --%>
```


Using JavaBeans in JSP Documents

- Strategy for inserting content in JSP
- Invoke code in separate utility classes called JavaBeans by using the `jsp:useBean`, `jsp:getProperty`, and `jsp:setProperty` actions.

JavaBeans

- Java classes written in a standard format.
- Tools and other programs can query such classes for information without writing explicit code.
- Advantage when used with JSP:
  - Avoiding Java syntax
  - Simplified object sharing
  - Facilitate handling of request parameters
- Implementing JavaBeans:
  - Either implicit or explicit default constructor is required.
  - No `public` instance variables can be declared.
  - A bean has *properties* with persistent values that are accessed through `get-` and `set-` methods.
    - A `boolean` property `p` can in addition have a `boolean isP()` method.

Example: A JavaBean

- See file `horoscopeJSPII\WEB-INF\src\kam\jsp\HoroBean.java`.

```
package kam.jsp;

public class HoroBean
       implements Comparable<HoroBean> {

  private String client;
  private int age;

  private String sign;
  private String horoscope;

  public int getAge() {
    return age;
  }
  public void setAge(int age) {
    this.age = age;
  }
  public String getClient() {
    return client;
  }
  public void setClient(String client) {
    this.client = client;
  }
  public String getHoroscope() {
    return horoscope;
  }
  public void setHoroscope(String horoscope) {
    this.horoscope = horoscope;
  }
  public String getSign() {
    return sign;
  }
  public void setSign(String sign) {
    this.sign = sign;
  }
  public int compareTo(HoroBean b2) {
    int result =
              this.client.compareTo(b2.client);
    if (result == 0)
      result = this.age - b2.age;
    return result;
  }
}
```

Instantiating Beans

```
<jsp:useBean id="beanVar" class="fully qualified class name" />
```

- A local variable *beanVar* is declared in the `_jspService()` method and initialized with an instance of the specified class which implements a bean.
- In order to be accessible, the bean classes must be installed in the `/WEB-INF/classes` directory or contained in `jar` files that are installed in `/WEB-INF/lib` directory.
  - Bean classes should be organized in packages.
  - Fully qualified name of the class is essential, imports or no imports.

Optional Scope Instantiation

```
<jsp:useBean id="horoBean" class="kam.jsp.HoroBean" scope="page" />
```

- Instantiate only if an object does not exit with the same id in the specified scope.

Optional Bean Source

```
<jsp:useBean id="horoBean" beanName="HoroBean-file.obj" scope="page" />
```

- A bean can be deserialized from a file.

Optional Type Specification I

```
<jsp:useBean id="horoBean" class="kam.jsp.HoroBean"type="java.lang.Comparable"/>
```

- A type can be specified for the bean variable, that is compatible with the class type.

Optional Type Specification II

```
<jsp:useBean id="horoBean" type="kam.jsp.HoroBean" />
```

- The bean is *not* created. It is assumed that it already exists in the (default) scope.

Setting Bean Properties

```
<jsp:setProperty name="beanVar"
                 property="propertyName"
                 value="newPropertyValue" />
```

is equivalent to

```
<% beanVar.setPropertyName("newPropertyValue");  %>
```

- The property is set to the `String` value.

```
<jsp:setProperty name="beanVar" property="propertyName"
                                value="<%= expression %>" />
```

- The data type of the value of the expression must match the data type of the property.
  - The property is set to the value of the expression.
- A bean can be initialized conditionally at the time of its creation:

```
<jsp:useBean id="horoBean" class="kam.jsp.HoroBean" >
  <jsp:setProperty name="horoBean" property="client"    value="Quasi Modo"/>
  <jsp:setProperty name="horoBean" property="sign"      value="pisces"/>
  <jsp:setProperty name="horoBean" property="horoscope" value="Watch your back"/>
</jsp:useBean>
```

  - The body of the `jsp:useBean` action is only executed once when the bean is created.

Accessing Bean Properties

```
<jsp:getProperty name="beanVar" property="propertyName" />
```

is equivalent to

```
<%= beanVar.getPropertyName()  %>
```

- The *output* of the `jsp:getProperty` action is the `String` representation of the property value.
- The value of the `name` attribute in the `jsp:getProperty` action is the same as the value of the `id` attribute in the `jsp:useBean` action.

```
...
<jsp:getProperty name="horoBean" property="client" />
<jsp:getProperty name="horoBean" property="sign" />
<jsp:getProperty name="horoBean" property="horoscope" />
...
```

JavaBeans and Form Processing

Passing Form Parameters to Individual Bean Properties using the `request` *Object:*

```
<jsp:setProperty name="horoBean" property="client"
                 value="<%= request.getParameter(\"client\") %>"/>
<jsp:setProperty name="horoBean" property="sign"
                 value="<%= request.getParameter(\"sign\") %>"/>
```

Passing Form Parameters to Individual Bean Properties using the `param` *Attribute:*

```
<jsp:setProperty name="beanVar"
                 property="beanPropertyName"
                 param="formParameterName" />
```

is equivalent to

```
<jsp:setProperty name="beanVar"
                 property="beanPropertyName" />
```

- The second form assumes that the name of the bean property is identical to the name of the form parameter.
- The value of the form parameter is automatically converted to the data type of the bean property and assigned to the named property.
  - Conversion is from the `String` type to a primitive type or to a primitive type's wrapper class.

- If a form parameter has the `null` value or it's value is the empty string, the corresponding bean property is not set.

```
...
<jsp:setProperty name="horoBean" property="client" param="client"/>
<jsp:setProperty name="horoBean" property="age" param="age"/>
<jsp:setProperty name="horoBean" property="sign" param="sign"/>
...
```

- The code can be simplified, assuming the property and form parameter names are identical:

```
...
<jsp:setProperty name="horoBean" property="client" />
<jsp:setProperty name="horoBean" property="age" />
<jsp:setProperty name="horoBean" property="sign" />
...
```

Passing All Form Parameters to A Bean

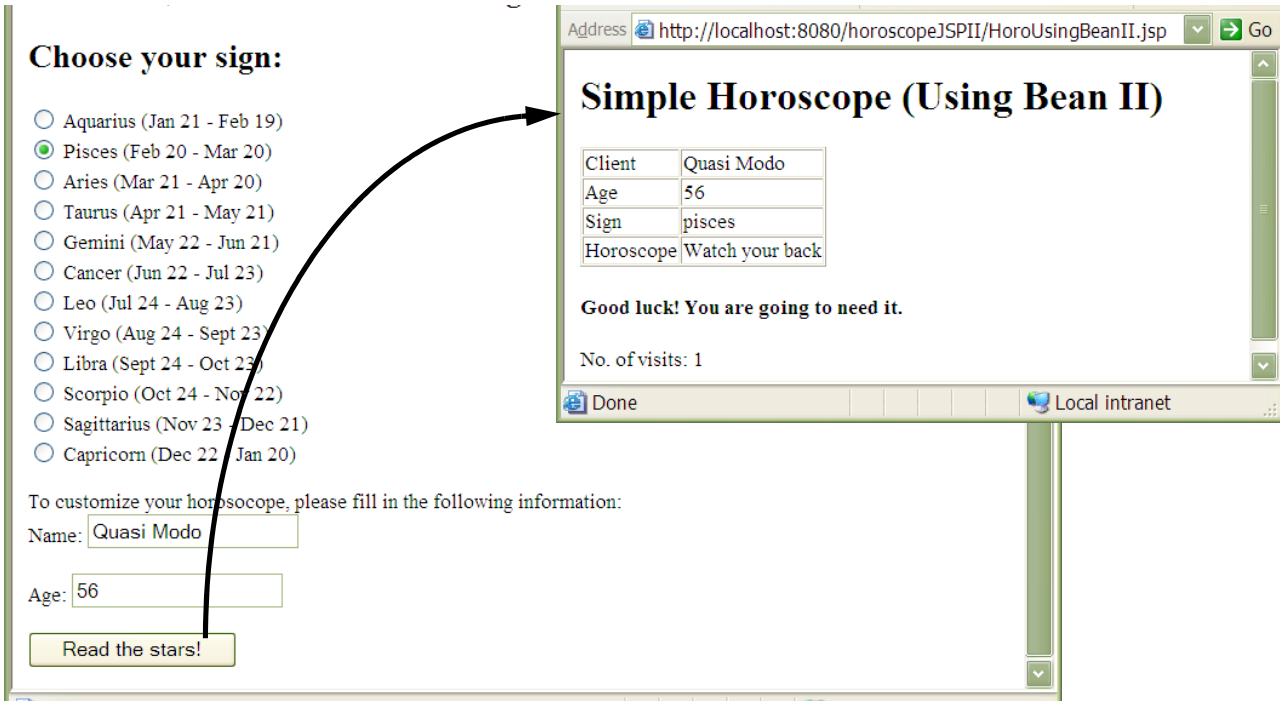
```
<jsp:setProperty name="beanVar"
                 property="*" />
```

- The names of the form parameters must match the names of the properties in the bean.
- If a parameter has the `null` value or it's value is the empty string, the corresponding bean property is not set.
- Note that properties that do not receive values via form parameters are not affected.

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

Example: see the following files in the `horoscopeJSPII` *web application.*

```
WEB-INF/src/kam/jsp/HoroBean.java
HoroUsingBean.jsp
HoroUsingBeanII.jsp
HoroUsingBeanIII.jsp
HoroscopeForm.html
HoroscopeFormII.html
HoroscopeFormIII.html
```

Sharing Beans using the `scope` Attribute in the `jsp:useBean` Action

- In addition to being bound to local variables in the `_jspService()` method, beans can also be stored in the implicit objects by specifying the appropriate value for the `scope` attribute.
- The bean can then be accessed by the `getAttribute()` method of the implicit object.
- The bean name must be unique across all scopes.

Value of `scope` attribute	Description
`"page"`	Bean is placed in the `pageContext` object associated with the current page for the duration of the request. The bean is thus not sharable. Default value.
`"request"`	Bean is placed in the `HttpServletRequest` object for the duration of the request. The bean can be shared among JSP pages and servlets if the request is forwarded to them.
`"session"`	Bean is placed in the `HttpSession` object associated with the request. The bean can be used in session tracking.
`"application"`	Bean is placed in the `ServletContext` object associated with the application. The bean can shared by all JSP pages and servlets in the web application.

- Remarks on bean sharing in a web application:
  - A bean in the page scope is not shared. It is only available during the life time of the page.
  - The bean in the request scope is shared among the main page and the included pages during the life time of the request, but it is not shared with other JSP pages. It is certainly not persistent between requests.
  - A bean in the session scope is shared by all the JSP pages between requests from the same client during the life time of the session.
  - A bean in the application scope is available to any JSP page in the web application, regardless of request or session.

Example: Sharing Beanies

- See the following files in the `myJSPExamples` web application:

  `index.html`, `WEB-INF/src/kam/jsp/SharedBeanie.java`, `Page1A.jsp`, `Page1B.jsp`, `Page2.jsp`
- A bean is instantiated in each scope and its properties set, only if it does not exist from before in that scope.
- Properties of the bean in each scope are printed.
- Note that `Page1A.jsp` includes `Page1B.jsp`.

```
<%-- Page1A.jsp --%>
<html>
  <head><title>Sharing Beanies</title></head>
  <body><h1>Sharing Beanies in Implicit Objects </h1>

<jsp:useBean id="beanieInPage" class="kam.jsp.SharedBeanie" scope="page" >
<jsp:setProperty name="beanieInPage" property="beanieName"
                 value="beanieInPage"/>
<jsp:setProperty name="beanieInPage" property="createrPage"
                 value="Page1A.jsp"/>
<jsp:setProperty name="beanieInPage" property="dateCreated"
                 value="<%= new java.util.Date() %>" />
</jsp:useBean>

<jsp:useBean id="beanieInRequest" class="kam.jsp.SharedBeanie" scope="request" >
<jsp:setProperty name="beanieInRequest" property="beanieName"
                 value="beanieInRequest"/>
<jsp:setProperty name="beanieInRequest" property="createrPage"
                 value="Page1A.jsp"/>
<jsp:setProperty name="beanieInRequest" property="dateCreated"
                 value="<%= new java.util.Date() %>" />
</jsp:useBean>
```



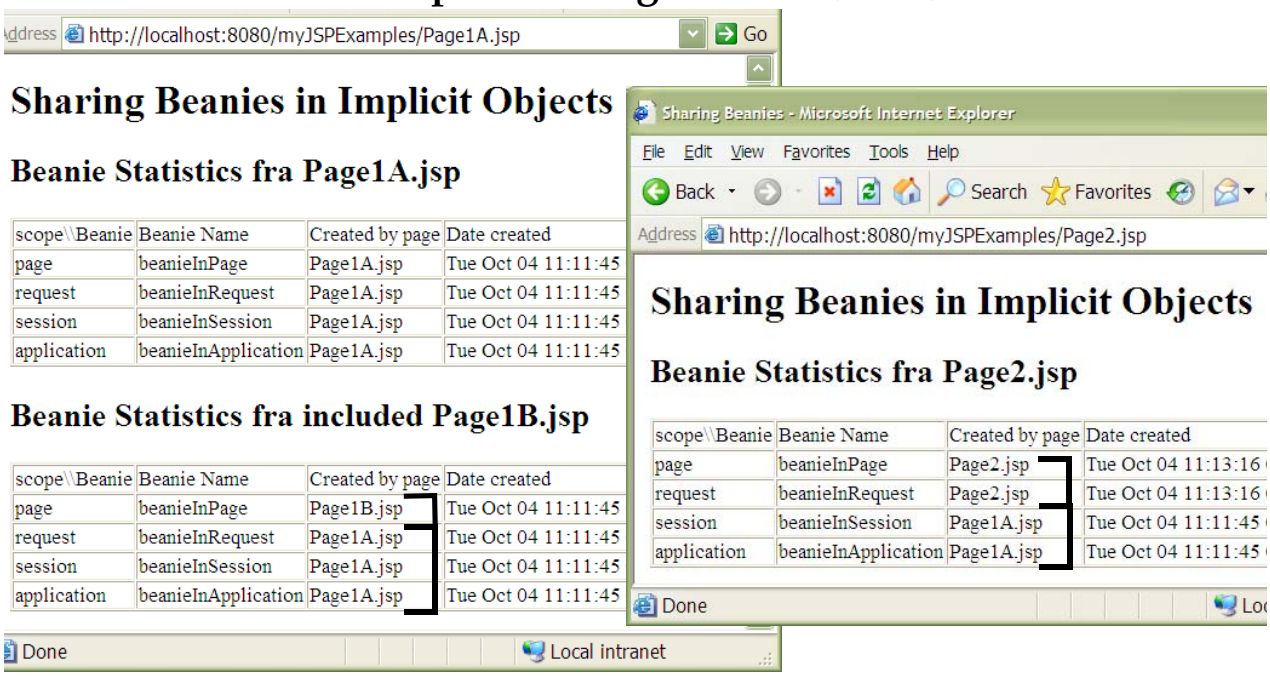
```
<jsp:useBean id="beanieInSession" class="kam.jsp.SharedBeanie" scope="session" >
<jsp:setProperty name="beanieInSession" property="beanieName"
                 value="beanieInSession"/>
<jsp:setProperty name="beanieInSession" property="createrPage"
                 value="Page1A.jsp"/>
<jsp:setProperty name="beanieInSession" property="dateCreated"
                 value="<%= new java.util.Date() %>" />
</jsp:useBean>

<jsp:useBean id="beanieInApplication" class="kam.jsp.SharedBeanie" scope="application"
>
<jsp:setProperty name="beanieInApplication" property="beanieName"
                 value="beanieInApplication"/>
<jsp:setProperty name="beanieInApplication" property="createrPage"
                 value="Page1A.jsp"/>
<jsp:setProperty name="beanieInApplication" property="dateCreated"
                 value="<%= new java.util.Date() %>" />
</jsp:useBean>
```

```
<h2>Beanie Statistics fra Page1A.jsp </h2>
    <table border="1" cellspacing="1" cellpadding="1">
      <tr>
        <td>scope\\Beanie</td>
        <td>Beanie Name</td>
        <td>Created by page</td>
        <td>Date created</td>
      </tr>

      <tr>
        <td>page</td>
        <td><jsp:getProperty name="beanieInPage" property="beanieName" /> </td>
        <td><jsp:getProperty name="beanieInPage" property="createrPage" /> </td>
        <td><jsp:getProperty name="beanieInPage" property="dateCreated" /> </td>
      </tr>

      <tr>
        <td>request</td>
        <td><jsp:getProperty name="beanieInRequest" property="beanieName" /> </td>
        <td><jsp:getProperty name="beanieInRequest" property="createrPage" /> </td>
        <td><jsp:getProperty name="beanieInRequest" property="dateCreated" /> </td>
      </tr>
```



```
      <tr>
        <td>session</td>
        <td><jsp:getProperty name="beanieInSession" property="beanieName" /> </td>
        <td><jsp:getProperty name="beanieInSession" property="createrPage" /> </td>
        <td><jsp:getProperty name="beanieInSession" property="dateCreated" /> </td>
      </tr>
      <tr>
        <td>application</td>
        <td><jsp:getProperty name="beanieInApplication" property="beanieName" /> </td>
        <td><jsp:getProperty name="beanieInApplication" property="createrPage" /> </td>
        <td><jsp:getProperty name="beanieInApplication" property="dateCreated" /> </td>
      </tr>
    </table>
<h2>Beanie Statistics fra included Page1B.jsp </h2>
<jsp:include page="Page1B.jsp" />
</body></html>
```

Example: Sharing Beanies (cont.)

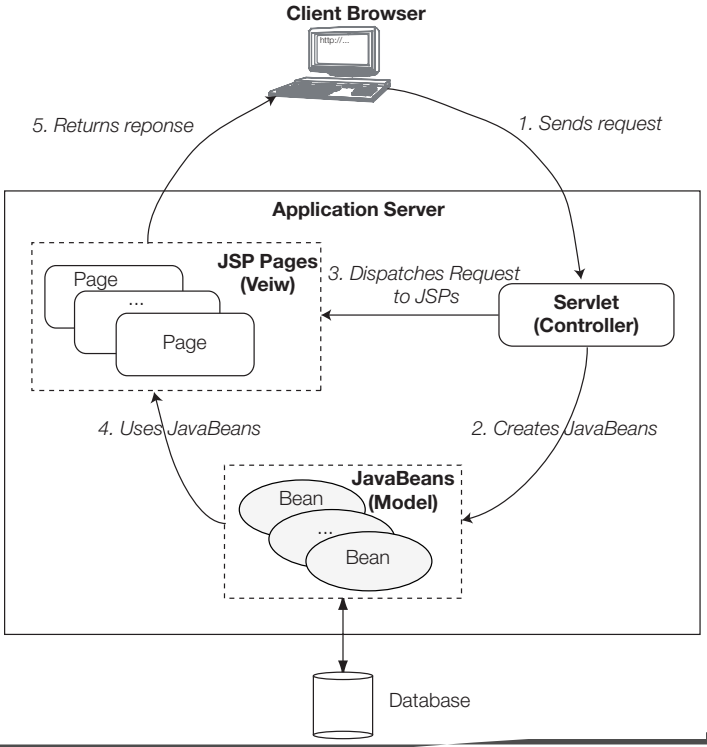


The MVC2 Architecture for Web Applications

- Servlets good at data processing and business logic, lousy at generating HTML content.
- JSPs good at generating HTML content, lousy at data processing and business logic.
- Combination of Servlets and JSPs provides generation of *flexible* web pages there HTML content can be dependent on data processing and business logic.
- Architecture Pattern: Model-View-Controller 2 Model (**MVC2**)
  1. A single servlet (*controller*) handles the incoming requests.
  2. The servlet invokes the data access code and the business logic code to create beans (*model*) that act as repository for the results.
  3. The servlet dispatches the request to appropriate JSP (*view*) to create the presentation of the response.
  4. The JSP uses the results from the beans to create the presentation.

The business-logic and data-access layers are separated from the presentation layer.

MVC2 Model



Implementing MVC2 Model

- The beans that represent the data -- *the model*
- The servlet that handles requests, invokes business-logic and data-access code, and stores the beans in the appropriate scope -- *the controller*
- The *business-logic and data-access code* that populates the beans
- The JSPs that extract the results from the beans to create the presentation -- *the view*

Creating Beans to represent Data

- Beans are usually implemented as *value objects* that can represent results and have limited additional functionality.
- A *request bean* is used to store request parameters.
- A *result bean* is used to store the results of the request.

Implementing Servlet to handle Requests

- The control servlet should read the *request information*.

```
String sign = request.getParameter("sign");
```

Populating the Beans with Results

- The control servlet should invoke the business-logic and data-access code to populate the beans.
- The `set`*`Property`*`()` method can be used to set a bean *`property`*.

```
valueBean.setProperty(propertyValue);
```


Storing the Beans in Implicit Objects

- The controller servlet can store the beans in appropriate implicit objects so that the JSPs can retrieve them.
- Obtaining the implicit object.
  - The request object is passed to the servlet, so it is readily available to store beans.
  - To create the session object, the no-argument `getSession()` method is suitable, as it will only create a session if none exists from before.

    ```
    HttpSession session = getSession();
    ```
  - To create the application/context object, the `getServletContext()` method is suitable.

    ```
    ServletContext application = getServletContext(); // Returns the same context.
    ```
- Storing in an implicit object:

  ```
  implicitObj.setAttribute("key", valueBean);
  ```
  - Thread safety may require synchronization on the implicit object, for example if several threads want to access the application/context object.

    ```
    synchronized(applicationContext) {
      applicationContext.setAttribute("horoBeanie", valueBean);
      //...
    }
    ```

Dispatching Requests to JSPs

Forwarding to an active resource using the `RequestDispatcher` *object.*

```
RequestDispatcher dispatcher = request.getRequestDispatcher("relAddress");
dispatcher.forward(request, response);
```

- The relative address identifies the active resource (servlet or JSP).
- The request is forwarded *directly* to the active resource, no round trip to the client is involved.
  - The request information is preserved automatically.
- The *output* is completely the responsibility of the active resource.
- The forwarding is done completely on the server side.
- The URL of the original request is preserved.
- Forwarding is to a resource on the same server.

Forwarding requests from JSPs

- Convenient to use the `jsp:forward` action.

```
<jsp:forward page="relative URL" />
```


Forwarding to an static resource using the `RequestDispatcher` *object.*

- The forwarded requests use the same HTTP method (GET, POST, etc.) as the original request.
- Forwarding to a static resource where the original request used the GET method works fine, but not if it was the POST method.
  - Work around: change the extension `.html` to `.jsp`, then the HTML content is created by a JSP that can handle the POST method.

Redirecting to a resource.

```
response.redirect(relAddress); // committed
```

- A response is sent to the client with a 302 status code and a URL in the `Location` header field.
  - The request information is not preserved automatically.
- Results is a new final URL.
- Redirecting can be to a resource on another server.

Including Pages

- Allows the servlet to compose the final output by *combining* the output from other JSPs.

```
RequestDispatcher dispatcher = request.getRequestDispatcher("/page1.jsp);
dispatcher.include(request, response);
dispatcher = request.getRequestDispatcher("/page2.jsp);
dispatcher.include(request, response);
...
```


Using Beans in JSPs

- The view JSPs normally do not create any new beans.
- The view JSPs normally do mutate the bean properties.
- The `jsp:useBean` action uses the `type` attribute to specify the *class* of the bean, as the bean should not be created.

```
<jsp:useBean id="horoBeanie" type="kam.jsp.HoroBean2" scope="request" />
```

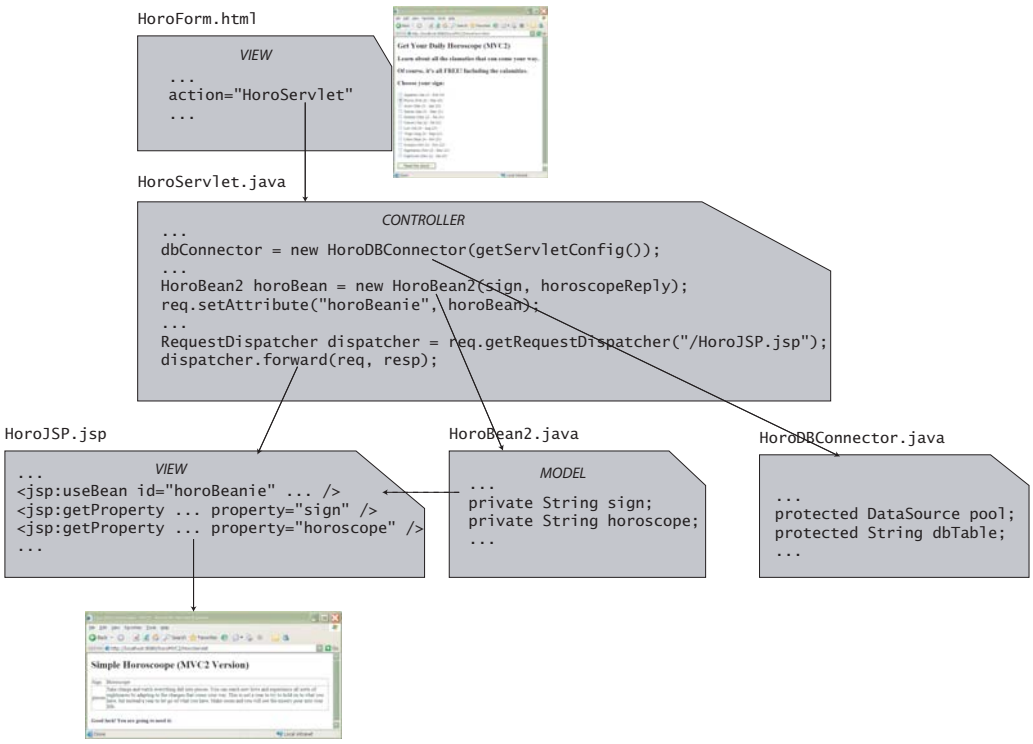
- The `jsp:useBean` action specifies where the bean is stored.
- The `jsp:getProperty` action can be used to access the bean properties.

```
<jsp:getProperty name="horoBeanie" property="sign" />
<jsp:getProperty name="horoBeanie" property="horoscope" />
```

Example: the horoMVC2 application

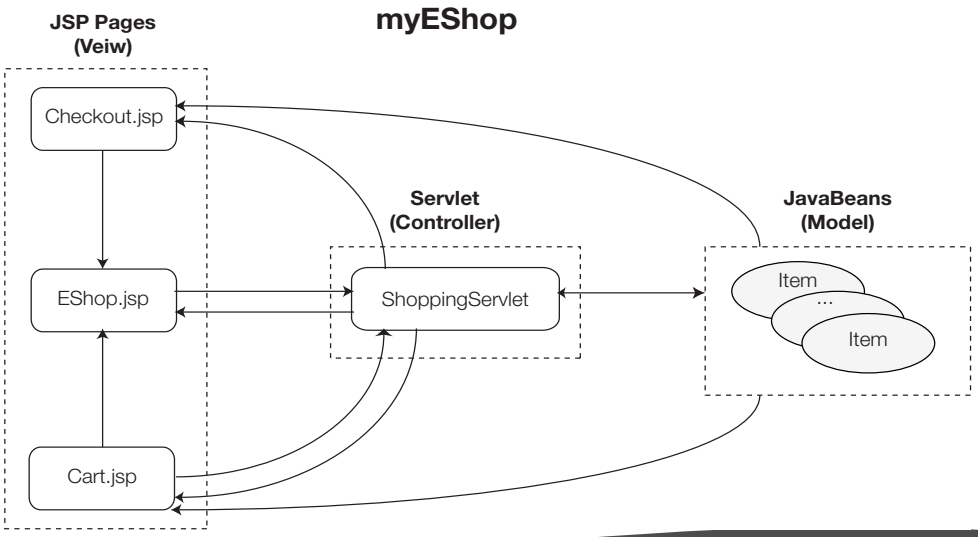
- See the following files for the horoMVC2 web application:
  - index.html, HoroForm.html, HoroJSP.jsp, WEB-INF\src\kam\jsp\HoroBean2.java, WEB-INF\src\kam\jsp\HoroDBConnector.java, WEB-INF\src\kam\jsp\HoroServlet.java, WEB-INF\web.xml
  - horoMVC2.xml

Example: the horoMVC2 application (cont.)

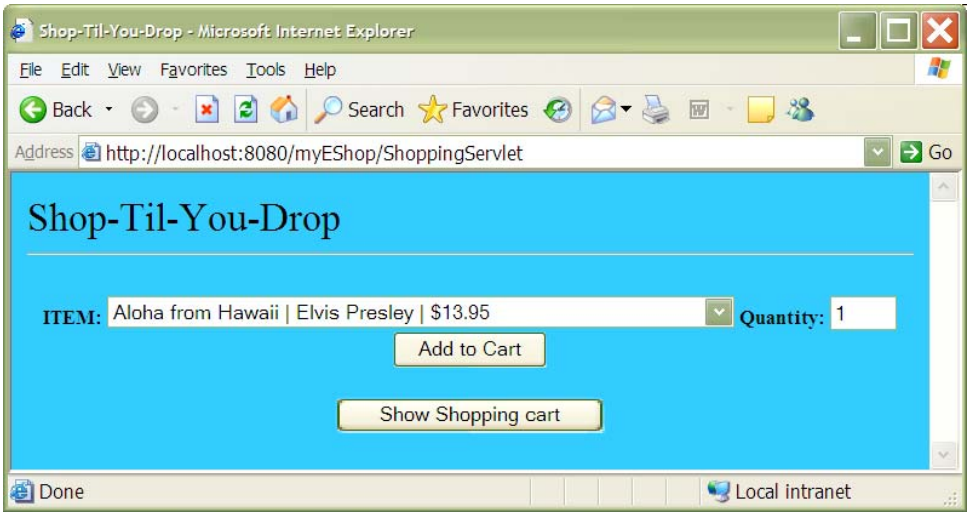


Example: the myEShop application

- See the following files for the `myEShop` web application:
  - `index.html`, `EShop.jsp`, `Cart.jsp`, `Checkout.jsp`, `error.html`, `WEB-INF\src\kam\shopping\Item.java`, `WEB-INF\src\kam\shopping\ShoppingServlet.java`, `WEB-INF\web.xml`
  - `myEShop.xml`



Example: the myEShop application (cont.)



Example: the myEShop application (cont.)

Example: the myEShop application (cont.)