CSI 3140

WWW Structures, Techniques and Standards

Separating Programming and Presentation:
JSP Technology

Why JSP?

◆ Servlet/CGI approach: server-side code is a program with HTML embedded

◆ JavaServer Pages (and PHP/ASP/ColdFusion) approach: server-side "code" is a document with program embedded

- Supports cleaner separation of program logic from presentation
- Facilitates division of labor between developers and designers

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html" />
  <jsp:output
    omit-xml-declaration="yes"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

  <head>
    <title>
      HelloCounter.jspx
    </title>
  </head>
```

Default namespace is XHTML

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html" />
  <jsp:output
    omit-xml-declaration="yes"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

  <head>
    <title>
      HelloCounter.jspx
    </title>
  </head>
</head>
```

Also uses two JSP-defined namespaces

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
```

JSP-defined
markup (initialization)

```
  <jsp:directive.page contentType="text/html" />
  <jsp:output
    omit-xml-declaration="yes"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

  <head>
    <title>
      HelloCounter.jspx
    </title>
  </head>
```

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html" />
  <jsp:output
    omit-xml-declaration="yes"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />
```

```
<head>
  <title>
    HelloCounter.jspx
  </title>
</head>
```

Standard XHTML

JSP Example

```
<body>
  <jsp:scriptlet>
    /* Initialize and update the "visits" variable. */
  </jsp:scriptlet>
  <c:if test="${empty visits}">
    <c:set var="visits" scope="application" value="0" />
  </c:if>
  <c:set var="visits" scope="application" value="${visits+1}" />

  <p>
    Hello World!
  </p>
  <p>
    This page has been viewed
      ${visits}
    times since the most recent
    application restart.
  </p>
</body>
</html>
```

JSP scriptlet

JSP Example

```
<body>

  <jsp:scriptlet>
    /* Initialize and update the "visits" variable. */
  </jsp:scriptlet>
  <c:if test="${empty visits}">
    <c:set var="visits" scope="application" value="0" />
  </c:if>
  <c:set var="visits" scope="application" value="${visits+1}" />

  <p>
    Hello World!
  </p>
  <p>
    This page has been viewed
      ${visits}
    times since the most recent
    application restart.
  </p>
</body>
</html>
```

JSP-based program logic:
initialize and increment variable

JSP Example

```
<body>

    <jsp:scriptlet>
       /* Initialize and update the "visits" variable. */
    </jsp:scriptlet>
    <c:if test="${empty visits}">
       <c:set var="visits" scope="application" value="0" />
    </c:if>
    <c:set var="visits" scope="application" value="${visits+1}" />

    <p>
       Hello World!
    </p>
    <p>
       This page has been viewed
       ${visits}
       times since the most recent
       application restart.
    </p>
</body>
</html>
```

Replaced with value of variable

JSP Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head><title>
       HelloCounter.jspx</title></head><body><p>
        Hello World!</p><p>
        This page has been viewed
          3
        times since the most recent
        application restart.</p></body></html>
```

Output
XHTML
document
after 3 visits

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
<jsp:directive.page contentType="text/html" />
<jsp:output
  omit-xml-declaration="yes"
  doctype-root-element="html"
  doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
  doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

<head>
  <title>
    HelloCounter.jspx
  </title>
</head>
```

JSP Example

- Used `html` as root element
  - Can use HTML-generating tools, such as Mozilla Composer, to create the HTML portions of the document
  - JSP can generate other XML document types as well

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html" />
  <jsp:output
    omit-xml-declaration="yes"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

  <head>
    <title>
      HelloCounter.jspx
    </title>
  </head>
```

JSP Example

◆ Namespaces

- JSP (basic elements, normal prefix jsp)

- Core JSP Standard Tag Library (JSTL) (prefix c)

  - Tag library: means of adding functionality beyond basic JSP

  - JSTL included in with JWSDP 1.3 version of Tomcat

  - JSTL provides tag libraries in addition to core (more later)

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html" />
  <jsp:output
    omit-xml-declaration="yes"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

  <head>
    <title>
      HelloCounter.jspx
    </title>
  </head>
```

JSP Example

♦ JSP elements

- `directive.page`: typical use to set HTTP response header field, as shown (default is text/xml)

- `output`: similar to XSLT output element (controls XML and document type declarations)

JSP Example

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html" />
  <jsp:output
    omit-xml-declaration="yes"
    doctype-root-element="html"
    doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
    doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

  <head>
    <title>
      HelloCounter.jspx
    </title>
  </head>
```

JSP Example

```
<body>
  <jsp:scriptlet>
    /* Initialize and update the "visits" variable. */
  </jsp:scriptlet>
  <c:if test="${empty visits}">
    <c:set var="visits" scope="application" value="0" />
  </c:if>
  <c:set var="visits" scope="application" value="${visits+1}" />

  <p>
    Hello World!
  </p>
  <p>
    This page has been viewed
      ${visits}
    times since the most recent
    application restart.
  </p>
</body>
</html>
```

JSP Example

- Template data: Like XSLT, this is the HTML and character data portion of the document

- Scriptlet: Java code embedded in document
  - While often used in older (non-XML) JSP pages, we will avoid scriptlet use
  - One use (shown here) is to add comments that will not be output to the generated page

JSP Example

```
<body>

  <jsp:scriptlet>
    /* Initialize and update the "visits" variable. */
  </jsp:scriptlet>
  <c:if test="${empty visits}">
    <c:set var="visits" scope="application" value="0" />
  </c:if>
  <c:set var="visits" scope="application" value="${visits+1}" />

  <p>
    Hello World!
  </p>
  <p>
    This page has been viewed
      ${visits}
    times since the most recent
    application restart.
  </p>
</body>
</html>
```

JSP Example

◆ Core tag library supports simple programming

- **if**: conditional
  - **empty**: true if variable is non-existent or undefined
- **set**: assignment
  - application scope means that the variable is accessible by other JSP documents, other users (sessions)

JSP and Servlets

- JSP documents are not executed directly
  - When a JSP document is first visited, Tomcat
    1. Translates the JSP document to a servlet
    2. Compiles the servlet
  - The servlet is executed
- Exceptions provide traceback information for the servlet, *not* the JSP
  - The servlets are stored under Tomcat `work` directory

JSP and Servlets

◆ A JSP-generated servlet has a `_jspService()` method rather than `doGet()` or `doPost()`

- This method begins by automatically creating a number of implicit object variables that can be accessed by scriptlets

Object name	Instance of
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
out	javax.servlet.jsp.JspWriter

JSP and Servlets

◆ Translating template data:

```
out.write("<head>");
out.write("<title>");
out.write("\n        HelloCounter.jspx");
out.write("</title>");
out.write("</head>");
out.write("<body>");
```

◆ Scriptlets are copied as-is to servlet:

```
/* Initialize and update the "visits" variable. */
```

JSP and Servlets

◆Scriptlets can be written to use the implicit Java objects:

```
<jsp:scriptlet>
  out.write("<p>Hello " +
            request.getParameter("username") +
            "!</p>");
</jsp:scriptlet>
```

◆We will avoid this because:

- It defeats the separation purpose of JSP
- We can incorporate Java more cleanly using JavaBeans technology and tag libraries

JSP and Servlets

◆JSP elements translate to:

JSP default

```
response.setContentType("text/html;charset=UTF-8");

out.write("<!DOCTYPE html PUBLIC
\"-//W3C//DTD XHTML 1.0 Strict//EN\"
\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">\n");
```

◆${visits} in template code translates to out.write() of value of variable

◆Core tags (*e.g.*, if) normally translate to a method call

Web Applications

◆ A web application is a collection of resources that are used together to implement some web-based functionality

◆ Resources include

- Components: servlets (including JSP-generated)
- Other resources: HTML documents, style sheets, JavaScript, images, non-servlet Java classes, *etc*.

Web Applications

◆ Sharing data between components of a web application

- Tomcat creates one `ServletContext` object per web application
- Call to `getServletContext()` method of a servlet returns the associated `ServletContext`
- `ServletContext` supports `setAttribute()`/`getAttribute()` methods

Web Applications

◆ Within Tomcat, all of the files of a simple web app are placed in a directory under `webapps`

- JSP documents can go in the directory itself
- "Hidden" files--such as servlet class files--go under a `WEB-INF` subdirectory (more later)

◆ Once the web app files are all installed, use Tomcat Manager to deploy the app

Web Applications

- Deploying a web app consisting of a single JSP document `HelloCounter.jspx`:
  - Create directory `webapps/HelloCounter`
  - Copy JSP doc to this directory
  - Visit `localhost:8080/manager/html`
  - Enter `HelloCounter` in "WAR or Directory URL" box and click Deploy button
- Web app is now at URL `localhost:8080/HelloCounter/HelloCounter.jspx`

Web Applications

◆ Manager app:

- Stop: web app becomes unavailable (404 returned)

- Start: web app becomes available again

- Reload: stop web app, restart with latest versions of files (no need to restart server)

- Undeploy: stop app and *remove all files!*
  - <u>Always</u> keep a copy of app outside `webapps`

Web Applications

◆ Set parameters of a web application by

- Creating a deployment descriptor (XML file)
- Saving the descriptor as `WEB-INF/web.xml`

◆ Simple example `web.xml`:

```
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <display-name>HelloCounter</display-name>

</web-app>
```

Web Applications

TABLE 8.2: Some elements of web application deployment descriptors.

Element	Use (as child of web-app)
display-name	Provides name to be displayed for application (for example, in Manager's Display Name field)
description	Provides text describing the web application for documentation purposes
context-param	Provides parameter value that can be used by components for initialization
servlet	Associates a name with either a servlet class or a JSP document and optionally sets other options and parameters for the servlet/JSP document
servlet-mapping	Associates a URL (or a set of URL's) with one of the servlet names defined by a servlet element
session-config	Specifies the default for the length of time that a session can be idle before being terminated
mime-mapping	Associates file extensions with MIME types

Web Applications

welcome-file-list	Specifies a list of files. If an HTTP request is mapped to a directory within this application, the server will search for within the directory for one of these files and respond with the first file found. If no file is found, the directory contents are displayed by default.
error-page	Specifies a resource (static web page or application component) that will provide the HTTP response when either a specified HTTP error status code is generated or a specified Java exception is thrown to the container.
jsp-config	Associates certain information with the JSP documents of an application, such as the location of tag library files and settings for certain JSP options
security-role	Defines a "role" (e.g., manager, customer) to be used for purposes of allowing or denying access to certain resources of a web application
security-constraint	Specifies application resources that should be access-protected and indicates which user roles will be granted access to these resources
login-config	Specifies how the container should request user name and password information (which will subsequently be mapped to one or more roles) when a user attempts to access a protected resource

Web Applications

◆ Some examples:

- Setting an initial value accessible by `application.getInitParameter()`:

```
<context-param>
  <param-name>initialVisitsValue</param-name>
  <param-value>527</param-value>
</context-param>
```

- Setting the length of time (in minutes) before a session times out:

```
<session-config>
  <session-timeout>1</session-timeout>
</session-config>
```

Web Applications

- Mapping URLs to app components:

```xml
<servlet>
   <servlet-name>visit_count</servlet-name>
   <jsp-file>/HelloCounter.jspx</jsp-file>
</servlet>
<servlet-mapping>
   <servlet-name>visit_count</servlet-name>
   <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>visit_count</servlet-name>
   <url-pattern>/visitor/*</url-pattern>
</servlet-mapping>
```

Web Applications

◆ There are four URL patterns (from high to low precedence)

TABLE 8.3: Forms of URL Patterns

Name	Example	Post-context path matched
Exact	/HelloCounter.jspx	The path /HelloCounter.jspx
Path-prefix	/visitor/*	The path /visitor or any path beginning with /visitor/
Extension	*.jsp	Any path ending in .jsp
Default	/	Any path

◆ If no URL pattern matches, Tomcat treats path as a relative file name

Web Applications

- Methods on `request` object for obtaining path information:
  - Example: `/HelloCounter/visitor/test.jsp`
  - `getContextPath()`: returns `/HelloCounter`
  - `getServletPath()`: returns `/visitor`
  - `getPathInfo()`: returns `/test.jsp`

JSP Expression Language (EL)

- ${visits+1} is an example of an EL expression embedded in a JSP document
  - ${…} is the syntax used in JSP documents to mark the contained string as an EL expression
  - An EL expression can occur
    - In template data: evaluates to Java String
    - As (part of) the value of certain JSP attributes: evaluates to data type that depends on context

JSP Expression Language (EL)

◆ EL literals:

- `true`, `false`
- decimal integer, floating point, scientific-notation numeric literals
- strings (single- or double-quoted)
- `null`

JSP Expression Language (EL)

◆ EL variable names: like Java

- Can contain letters, digits, _ , and $
- Must not begin with a digit
- Must not be reserved:

```
and    div   empty  eq    false  ge    gt    instanceof
le     lt    mod    ne    not    null  or    true
```

JSP Expression Language (EL)

- EL operators:
  - Relational: `<, >, <=, >=, ==, !=`
    - Or equivalents: `lt, gt, le, ge, eq, ne`
  - Logical: `&&, ||, !`
    - Or equivalents: `and, or, not`
  - Arithmetic:
    - `+, -` (binary and unary), `*`
    - `/, %` (or `div, mod`)
  - `empty`: `true` if arg is `null` or empty string/array/`Map`/`Collection`
  - Conditional: `? :`
  - Array access: `[ ]` (or object notation)
  - Parentheses for grouping

JSP Expression Language (EL)

◆ EL automatic type conversion

- Conversion for + is like other binary arithmetic operators (+ does *not* string represent concatenation)

- Otherwise similar to JavaScript

JSP Expression Language (EL)

◆ EL provides a number of implicit objects

◆ Most of these objects are related to but not the same as the JSP implicit objects

- JSP implicit objects cannot be accessed directly by name in an EL expression, but can be accessed indirectly as properties of one of the EL implicit objects

JSP Expression Language (EL)

TABLE 8.4: EL implicit objects.

EL Implicit Object Name	Represents
pageContext	Container for JSP implicit objects
pageScope	Values accessible via calls to page.getAttribute()
requestScope	Values accessible via calls to request.getAttribute()
sessionScope	Values accessible via calls to session.getAttribute()
applicationScope	Values accessible via calls to application.getAttribute()
param	Values accessible via request.getParameter()
paramValues	Values accessible via request.getParameterValues()
header	Values accessible via request.getHeader()
headerValues	Values accessible via request.getHeaders()
cookie	Map from cookie names to their associated Cookie values (data obtained via request.getCookies())
initParam	Values accessible via application.getInitParameter()

JSP Expression Language (EL)

- **`pageContext`**: provides access to JSP implicit objects
  - Ex: EL expression `pageContext.request` is reference to the JSP `request` object
- **`page`**: JSP implicit object representing the servlet itself
- JSP objects `page`, `request`, `session`, and `application` all have `getAttribute()` and `setAttribute()` methods
  - These objects store EL scoped variables (*e.g.*, `visits`)

JSP Expression Language (EL)

- Reference to non-implicit variable is resolved by looking for an EL scoped variable in the order:
  - page
  - request
  - session
  - application
- If not found, value is `null`
- If found, value is `Object`
  - JSP automatically casts this value as needed

JSP Expression Language (EL)

◆ All EL implicit objects except `pageContext` implement Java Map interface

◆ In EL, can access Map using array or object notation:
- Servlet: `request.getParameter("p1")`
- EL:
  `param['p1']`
  or
  `param.p1`

JSP Expression Language (EL)

◆ Array/List access:

If EL scoped variable `aVar` represents

- Java array; or
- `java.util.List`

and if EL scoped variable `index` can be cast to integer

then can access elements of aVar by

- aVar[index]
- aVar.index

JSP Markup

- Three types of markup elements:
  - Scripting
    - Ex: `scriptlet`
    - Inserts Java code into servlet
  - Directive
    - Ex: `directive.page`
    - Instructs JSP translator
  - Action
    - Standard: provided by JSP itself
    - Custom: provided by a tag library such as JSTL

JSP Markup

- Two JSPX directives
  - `directive.page`; some attributes:
    - `contentType`
    - `session`: `false` to turn off use of `session` object
    - `errorPage`: component that will generate response if an exception is thrown
    - `isErrorPage`: true to access EL implicit `exception` object
  - `directive.include`: import well-formed XML

```
<jsp:directive.include file="../common/disclaimer.jspf" />
```

JSP Markup

TABLE 8.7: Some JSTL core actions.

Action	Purpose
set	Assign a value to a scoped variable, creating the variable if necessary
remove	Destroy a scoped variable
out	Write data to out implicit object, escaping XML special characters
url	Create a URL with query string
if	Conditional (if-then) processing
choose	Conditional (if-then-elseif) processing
forEach	Iterate over a collection of items

JSP Markup

- ◆ Common variables:
  - **var**
    - Name of a scoped variable that is assigned to by the action
    - Must be a string literal, not an EL expression
  - **scope**
    - Specifies scope of scoped variable as one of the literals `page`, `request`, `session`, or `application`
    - `page` default scope, unless otherwise specified

JSP Markup

- ◆ **set** action
  - Setting (and creating) a scoped variable

```
<c:set var="visits" scope="application" value="${visits+1}" />
```

  - Setting/creating an element of **Map**

```
<c:set target="${applicationScope}"    Map
       property="visits"                Key
       value="${visits+1}" />
```

    - Actually, this fails at run time in JWSDP 1.3 (which treats EL implicit object **Map**s as read-only)

JSP Markup

◆ `remove` action

- Only attributes are `var` and `scope`
- Removes reference to the specified scoped variable from the scope object

```
<c:remove var="visits"
          scope="application" />
```

JSP Markup

- **out** action
  - Normally used to write a string to the **out** JSP implicit object
  - Automatically escapes all five XML special characters

    ```
    <c:out value="${messy}" />
    ```

  - If value is **null** output is empty string
    - Override by declaring **default** attribute

JSP Markup

- **`url`** action
  - **`value`** attribute value is a URL to be written to the **`out`** JSP implicit object
  - URL's beginning with **`/`** are assumed relative to context path
  - **`param`** elements can be used to define parameters that will be URL encoded

```
<c:url value="/somewhere">
  <c:param name="username" value="Kim Sam" />
</c:url>
```

/myApp/somewhere?username=Kim+Sam

JSP Markup

- Alternative to the `value` attribute (`set` and `param` elements)
  - If element has content, this is processed to produce a `String` used for `value`
  - Even `out` element will produce string, *not* write to the `out` object

```
<c:set var="clean">
   <c:out value="${messy}" />
</c:set>
```

Assigns value of variable `messy` (XML escaped) to scoped variable `clean`

JSP Markup

- **if** action
  - General form includes scoped variable to receive **test** value

    Assigned Boolean value
    of **test** attribute

```
<c:if test="${visits gt 3}" var="testResult">
  You're becoming a regular!
</c:if>
```

  - The element can be empty if **var** is present

JSP Markup

◆ **choose** action

```
<c:choose>
  <c:when test="${visits eq 1}">
    Hi!</c:when>
  <c:when test="${visits eq 2}">
    Welcome back!</c:when>
  <c:otherwise>
    You're a regular!</c:otherwise>
</c:choose>
```

JSP Markup

♦ **forEach** action

■ Used to increment a variable:

```
<c:forEach var="i" begin="2" end="8" step="2">
  ${i}
</c:forEach>
```

■ Used to iterate over a data structure:

```
<ul>
  <c:forEach var="aHeader" items="${header}">
    <li><c:out value="${aHeader}" /></li>
  </c:forEach>
</ul>
```

JSP Markup

- ◆ forEach action
  - ■ Can iterate over array, Map, Collection, Iterator, Enumeration
  - ■ Elements of Map are Map.Entry, which support key and value EL properties:

```
<ul>
  <c:forEach var="aHeader" items="${header}">
    <li>
      <strong><c:out value="${aHeader.key}:" /></strong>
      <c:out value="${aHeader.value}" />
    </li>
  </c:forEach>
</ul>
```

JavaBeans Classes

◆ JSTL Core actions are designed to be used for simple, presentation-oriented programming tasks

◆ More sophisticated programming tasks should still be performed with a language such as Java

◆ JavaBeans technology allows a JSP document to call Java methods

JavaBeans Classes

◆Example

```
package my;
public class TestBean {
    private String greeting = "Hello World!";
    public String getWelcome() {
        return greeting;
    }
}
```

◆Requirements: JavaBeans class must
- Be `public` and not `abstract`
- Contain at least one *simple property design pattern* method (defined later)

JavaBeans Classes

♦ Using a JavaBeans class in JSP

```
<jsp:useBean id="testBean" class="my.TestBean" />

<head>
  <title>
    BeanTester.jspx
  </title>
</head>
<body>
  <h1>
    <c:out value="${testBean.welcome}" />
  </h1>
</body>
```

JavaBeans Classes

◆ Using a JavaBeans class as shown:

- Class must have a default (no-argument) constructor to be instantiated by `useBean`
    - Automatically supplied by Java in this example
- Class should belong to a package (avoids need for an `import`)
    - This class would go in `WEB-INF/classes/my/` directory

◆ Instance of a JavaBeans class is a bean

JavaBeans Classes

♦ Simple property design patterns
- Two types: getter and setter
  - Both require that the method be `public`
  - getter:
    - no arguments
    - returns a value
    - name begins with `get` (or `is`, if return type is `boolean`) followed by upper case letter
  - setter:
    - one argument (same type as setter return value)
    - void
    - name begins with `set` followed by upper case letter

JavaBeans Classes

♦EL calls simple property design method in response to access of bean property:

  ■ Attempt to read property generates call to associated `get`/`is` method (or error if none exists)

  ■ Attempt to assign value to property generates call to associated `set` method (or error)

JavaBeans Classes

♦ Example `setter` method

```
public void setWelcome(String welcome) {
    greeting = welcome;
}
```

♦ Calling setter from JSP

```
<c:set target="${testBean}" property="welcome" value="Howdy!" />
```

JavaBeans Classes

◆Simple property design pattern methods associate bean properties with beans

- Name of bean property obtained by removing get/is/set method prefix and following the rule:
  - If remaining name begins with two or more upper case letters, bean property name is remaining name: `setAValue() → AValue`
  - If remaining name begins with a single upper case letter, bean property name is remaining name with this letter converted to lower case: `getWelcome() → welcome`

Instantiating Beans

◆ Beans can be instantiated by a servlet and made available to JSP via scope objects

- Servlet

```
import my.TestBean;
...
HttpSession session = request.getSession();
TestBean testBean = new TestBean();
session.setAttribute("testBean", testBean);
```

- JSP: no need for useBean action

```
${sessionScope.testBean.welcome}
```

Instantiating Beans

◆ `useBean` only instantiates a bean if one does not already exist, can optionally perform initialization

```
<jsp:useBean id="testBean" class="my.TestBean" scope="session">
  <c:set target="${testBean}" property="welcome" value="Greetings!" />
</jsp:useBean>
```

Evaluated only if useBean instantiates TestBean

Using Beans

◆Example: mortgage calculation

```java
package mortgage;
public class Mortgage
{
    private double amount = -1.0;
    private int nMonths = -1;
    private double intRate = -1.0;

    public void setAmount(double amount) {
        this.amount = amount;
    }
    public void setMonths(int nMonths) {
        this.nMonths = nMonths;
    }
    public void setRate(double intRate) {
        this.intRate = intRate;
    }
    public double getPayment() {
        return ... ;
    }
}
```

Using Beans

```
<jsp:useBean id="calc" class="mortgage.Mortgage" />
<p>The monthly payment for the values you entered would be
    <c:set target="${calc}" property="amount"
           value="${param.mortgageAmount}" />
    <c:set target="${calc}" property="months"
           value="${param.period}" />
    <c:set target="${calc}" property="rate"
           value="${param.rate}" />

    ${calc.payment}
</p>
```

Call to getPayment() method

Java API Bean Properties

◆ Many Java API methods conform to simple property design patterns

```
<jsp:scriptlet>
   out.write(request.getPathInfo());
</jsp:scriptlet>
```

◆ Can usually treat as bean properties

```
${pageContext.request.pathInfo}
```

Tag Libraries

◆ Wouldn't it be nicer to write the mortgage app as

```
<p>The monthly payment for the values you entered would be
   <myTag:mortgage amount="${param.mortgageAmount}"
                   period="${param.period}"
                   rate="${param.rate}" />
</p>
```

Tag Libraries

```
<jsp:root version="2.0"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:c="http://java.sun.com/jsp/jstl/core">

<jsp:directive.attribute name="amount" required="true" />
<jsp:directive.attribute name="period" required="true" />
<jsp:directive.attribute name="rate" required="true" />

  <jsp:useBean id="calc" class="mortgage.Mortgage" scope="application" />

      <c:set target="${calc}" property="amount"
             value="${amount}" />
      <c:set target="${calc}" property="months"
             value="${period}" />
      <c:set target="${calc}" property="rate"
             value="${rate}" />

      ${calc.payment}

</jsp:root>
```
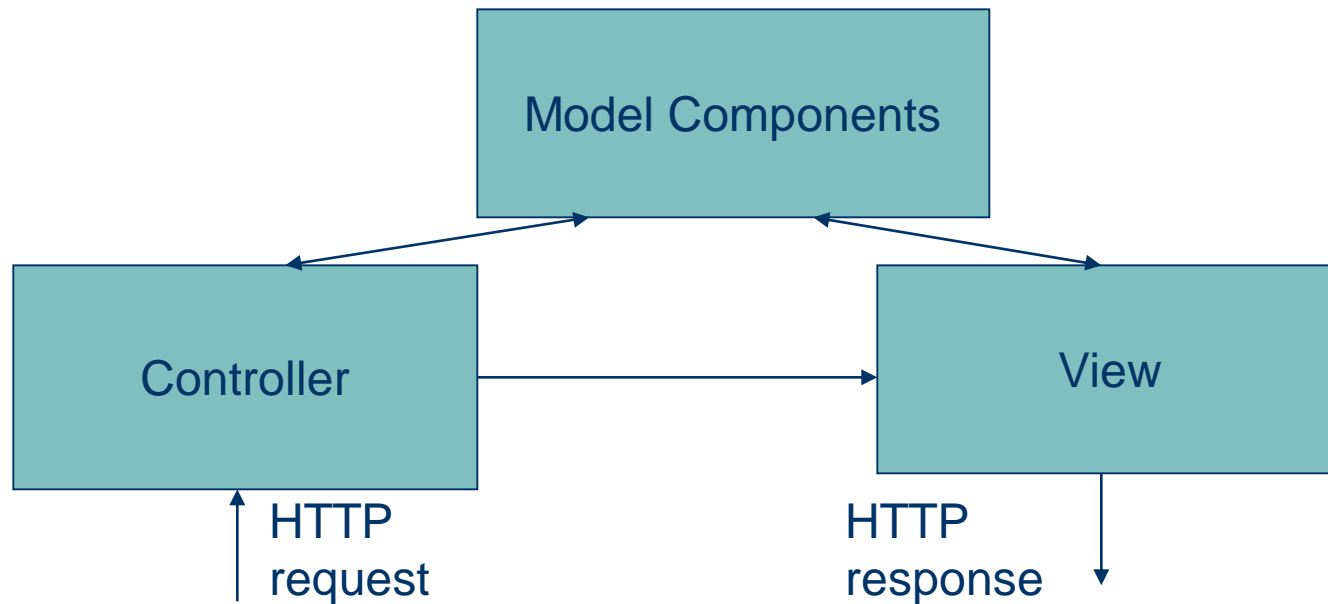
Tag Libraries

- Place custom tag definition in a tag file having the name of the custom action
  - `mortgage.tagx`
- Place tag file in a tag library (*e.g.*, directory containing tag files)
  - `/WEB-INF/tags`
- Add namespace declaration for tag library
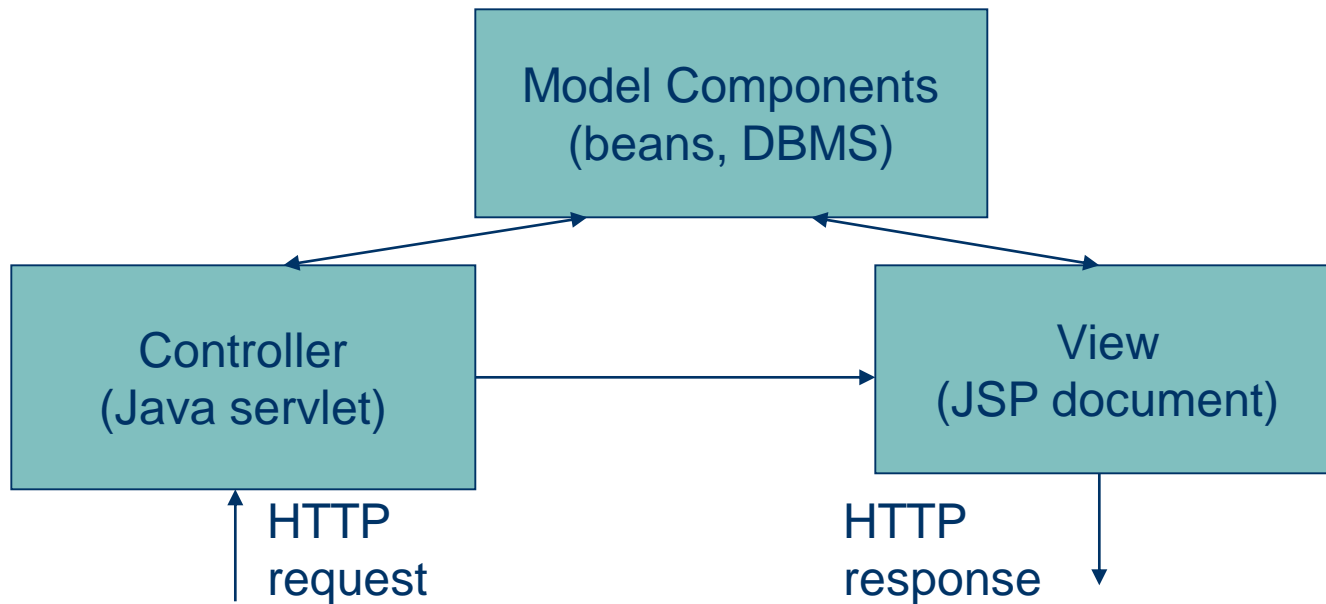
```
xmlns:myTag="urn:jsptagdir:/WEB-INF/tags"
```

MVC

◆ Many web apps are based on the Model-View-Controller (MVC) architecture pattern

MVC

◆ Typical JSP implementation of MVC

```
                    ┌─────────────────────┐
                    │  Model Components    │
                    │  (beans, DBMS)       │
                    └─────────────────────┘
                      ↗              ↖
        ┌──────────────────┐    ┌──────────────────┐
        │   Controller     │ →  │     View         │
        │  (Java servlet)  │    │  (JSP document)  │
        └──────────────────┘    └──────────────────┘
              ↑                        ↓
            HTTP                     HTTP
            request                  response
```

MVC

◆ Forwarding an HTTP request from a servlet to another component:

- By URL

```
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(contextRelativeURL);
```

Ex: /HelloCounter.jspx

- By name

```
<servlet>
    <servlet-name>visit_count</servlet-name>
    <jsp-file>/HelloCounter.jspx</jsp-file>
</servlet>

RequestDispatcher dispatcher =
    getServletContext().getNamedDispatcher("visit_count");
```

MVC

```
public class Controller extends HttpServlet
{
    /**
     * If session is new then increment and display the application
     * visit counter.  Otherwise (this is the continuation of an
     * active session), display a message.
     */
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException
    {
        HttpSession session = request.getSession();
        if (session.isNew()) {
            RequestDispatcher visitDispatch =
                getServletContext().getNamedDispatcher("visit_count");
            visitDispatch.forward(request, response);
        }
        else {
            RequestDispatcher laterDispatch =
                getServletContext().getNamedDispatcher("visit_later");
            laterDispatch.forward(request, response);
        }
    }
}
```

MVC

- ◆ **How does the controller know which component to forward to?**
  - ■ `getPathInfo()` value of URL's can be used
  - ■ Example:
    - ● servlet mapping pattern in `web.xml`:

      `/controller/*`

    - ● URL ends with: `/controller/help?prod=324324`

    - ● `getPathInfo()` returns: `/help`

MVC

- JSP `include` action

```
<table style="width:100%" border="0">
  <tbody>
    <tr>
      <td style="width:20%"
        ><jsp:include page="/navbar.jspx" /></td>
      <td style="width:80%"
        ><jsp:include page="/mainContent.jspx" /></td>
    </tr>
  </tbody>
</table>
```

Execute specified component and include its output in place of the include element

MVC

◆Adding parameters to the `request` object seen by an included component:

```
<jsp:include page="/navbar.jspx">
  <jsp:param name="currentPage" value="home" />
</jsp:include>
```

request object seen by `navbar.jspx` will include
parameter named `currentPage` with value `home`

- Applications of JSP
- As mentioned before, JSP is one of the most widely used language over the web. I'm going to list few of them here:
- JSP vs. Active Server Pages (ASP)
- The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- JSP vs. Pure Servlets
- It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- JSP vs. Server-Side Includes (SSI)
- SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- JSP vs. JavaScript
- JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
- JSP vs. Static HTML
- Regular HTML, of course, cannot contain dynamic information.