Java Server Page (JSP)

CS 4640
Programming Languages for Web Applications

[Based in part on SWE432 and SWE632 materials by Jeff Offutt]
[Robert W. Sebesta, “Programming the World Wide Web”]
Web Applications

• A web application uses enabling technologies to
  • Make web site contents dynamic
  • Allow users of the system to implement business logic on the server

• Web applications allow users to affect state on the server

An enabling technology makes web pages interactive and responsive to user inputs
Server Side Processing

UI implemented in a browser

Client

HTTP Request

data

Web server

Container engine

Program components

Server

HTTP Response

HTML

HTTP Response

Apache Tomcat

review
Enabling Technologies
Plug-in Scripted Pages

- **Scripted pages** look like HTML pages
- Execution is on the **server**, not on the client
- They have HTML with **program statements** that get and process data
- **PHP** scripts are **interpreted** within the server
- Scripted pages are generally easy to develop and deploy
- They mix logic with HTML -- can be difficult to read and maintain
Java Scripts and Java Servlets

• **Java Scripts** provide **client-side** execution ability
  - Interpreted
  - Cumbersome and error prone
  - Non-portable

• **Java Servlets** provide **server-side** execution
  - Compiled
  - Portable
  - Robust
  - Mixes static (HTML) with dynamic (business logic)
  - “Java that creates HTML”

HTML in Java
Java Server Page (JSP)

- JSP is a server-side technology that supports dynamic content.
- JSPs turn servlets "inside-out".
- JSPs are translated to servlets, compiled, then executed.
- This encourages separation of tasks.

Java in HTML

Diagram:
- Page Layout: Graphics designer
- Writing HTML: ?
- Integration with JSP: Web Java programmer
- App Development: Java, JavaBeans
  - Java programmer
What we can do with JSPs

- Collect inputs from users through web forms
- Retrieve information from some data sources
- Present information retrieved from some data sources
- Create web pages dynamically
- Provide web services
- Social networking system
- Jeopardy game
- And so much more
Why use JSPs

• Built on top of Java servlets API
  • Has access to Java APIs

• Can be used in combination with servlets and other Java classes
  • Servlets handle the business logic, Java classes process additional back-end logic
  • JSPs deal with the front page
Why use JSPs

- **HTML**
  - Cannot dynamically create information

- **(standard) JavaScript**
  - Generate HTML dynamically on the client
  - Cannot perform complex tasks (such as accessing and retrieving information from a database)
    - This is possible with node.js

- **Servlet**
  - Rely on `println()` to generate HTML document
  - Hard to develop / read / maintain

- **JSP**
  - Generate HTML dynamically on the server
  - Can perform complex tasks
  - Simple to write / read / maintain
JSP Processing

http://localhost:8080/cs4640/hello.jsp

1. Request

Web client

5. Response

Server with JSP container

2. Read

hello.jsp

3. Run JSP

4. Response

JSP execution – mental model of JSP developer
JSP Processing

1. request

2. if no hello.class exist or hello.jsp is newer than existing hello.class

3. Translate

4. Compile

5. Execute

6. response

Web client

http://localhost:8080/cs4640/hello.jsp

JSP execution – actual implementation
JSP Life Cycle

Initialization

jspInit()

Main logic

_jspService()

Destruction

jspDestroy()

Request

Response
JSP Elements

JSP syntax:

<% x ... %>

where x is one of the following

1. @ Directive: Global information for page
   Language, import statements, etc.

2. Scripting Elements: Java code
   • ! Declarations: Class level variables and methods
   • Scriptlets: A block of Java code, can make external calls
   • = Expressions: Values to be printed

3. Actions: To modify runtime behavior
(1) JSP Directives

Information sent to the JSP container

```%@ page attribute=value … %>
• Page attributes are listed in book
• You will usually use the defaults
```

```%@ include file="filename" %>
• File inserted into the JSP inline before JSP is compiled
```

```%@ taglib uri="tagLibURI" prefix="tagPrefix" %>
• Declare a tag library used in the JSP
• Usually for custom tags
```
Java code to define class-level variables and methods

```jsp
<%! int Sum = 0;
    private void AddToCount (int X)
    {
        // To be called from a scriptlet
        Sum = Sum + X;
    }
%>
```

`jspInit()` and `jspDestroy()` can also be defined here to initialize and clean up state
(2) JSP Scripts – Scriptlets

• Blocks of general **Java** code
• Placed in `_jspService()` by translator
• Can **access variables** from JSP Declarations
• Scriptlets can access servlet **objects**
  • `request` : HttpServletRequest object
  • `response` : HttpServletResponse object
  • `out` : for printing

```jsp
<% 
    String nameVal = request.getParameter("LASTNAME");
    out.println(nameVal);
%>
```

Must use the name “request”
(2) JSP Scripts – Expressions

Abbreviated scriptlet print statement

```html
<p>
The user’s last name is <%= nameVal %>
</p>
```

Expression is evaluated and turned into a string
(3) JSP Actions

• Tags to change the **behavior** of the JSP

• Action types:
  • `<jsp:include>`
  • `<jsp:useBean>`
  • `<jsp:setProperty>`
  • `<jsp:getProperty>`
  • `<jsp:forward>`
  • `<jsp:param>`
  • `<jsp:plugin>`
(3) JSP Actions – Include

• `<jsp:include>` can be used to include either a static or dynamic resource

Static

• A static file is loaded inline into the JSP before translation and compiling
• The same content is included every time
• Faster than dynamic include (no separated execution, refer to JSP processing slide 11)

Dynamic

• A web software component is run and the results (or responses) are included as part of the final response at runtime
• A dynamic include can result in different content each time
(3) JSP Actions – Include

Static

<jsp:include page="copyright.html" />

Dynamic

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

myjsp.jsp is compiled
myjsp.jsp is executed
Output from myjsp.jsp is included in the current JSP

Current output (of the current JSP) is flushed before myjsp.jsp is included
(3) JSP Actions – Java Beans

A Java Bean is a Java class with 3 characteristics:

1. public class
2. public constructor with no arguments
3. public get and set methods (called getters and setters)

Property : A special, simple data object (that is, variable)

- getName() … <jsp:getProperty>
- setName(String name) … <jsp:setProperty>
- Note that a bean is not a Java language feature, but a design convention (pattern)

(note: Java Beans will be discussed when we work on state handling in JSP)
(3) JSP Actions – Java Beans

useBean

• Causes a JavaBean object to be instantiated
• Gives a name to the new object (id=)
• Defines the scope (scope=)
• Declares the location (bean details)

(note: Java Beans will be discussed when we work on state handling in JSP)
Syntax for using a bean

```
<%@ page import="jspexamples.*" %>
<jsp:useBean id="letterColor" class="AlphabetCode" scope="page" />
```

- **Converting to Java import statement**: Java bean should always be in a package as required by most Web servers.
- **ID name to use for object**: AlphabetCode letterColor = new ...
- **Name of class**: AlphabetCode
- **Defines accessibility**: JSPs offer several useful scopes for variables (page, request, session, application)

**Note**: `scope="application"` allows Beans to be shared among different servlets → lead to interactions among each other ...

*(note: Java Beans will be discussed when we work on state handling in JSP)*
(3) JSP Actions – Properties

**setProperty** gives a value to a property in a bean

```xml
<jsp:setProperty name="langBean" property="language" value="Java"/>
```
Equivalent to the call: `langBean.setLanguage("Java");`

```
<jsp:setProperty name="langBean" property="*"/>
```
Sets all of the properties with values from HTML form

**getProperty** retrieves the value of a property

```xml
<jsp:getProperty name="langBean" property="language"/>
```
Equivalent to the call: `langBean.getLanguage();`

*(note: Java Beans will be discussed when we work on state handling in JSP)*
Case of property name is very important

- Property must begin with a lower case letter (e.g., “language”)
- Getters and setters must have the property name start with a capital letter (e.g., setLanguage(), getLanguage())

(note: Java Beans will be discussed when we work on state handling in JSP)
(3) JSP Actions – Java Bean Summary

• Using Java Beans increases separation between the HTML and Java

• The Beans / Property pattern provides a convenient standard for implementing standard Java classes

• JSP’s useBean uses Java reflection to translate property names (for example, “language”) to method calls that are assumed to exist ( “setLanguage()” and “getLanguage()” )

• The bean does not have to have an object with the name of the property, as long as it has a getter or setter

(note: Java Beans will be discussed when we work on state handling in JSP)
### (3) JSP Actions – Forwarding

**jsp:Forward** sends a request to another JSP on the **same** server

- Similar to a method call, but no return

```
<jsp:forward page="anotherPage.jsp" />
```

- When this statement is reached, execution will **jump** to the JSP anotherPage.jsp
- Use as a **front-end** when we need to decide which JSP to execute based on some input data
- Use to authenticate users
Deploying JSPs

- Import all beans into your JSP
  ```jsp
  <%@ page import="your-package.*" %>
  ```

- Put bean .class files into
  - local
    `/tomcat/webapps/your-context-or-your-project-folder/WEB-INF/classes/your-package-folder/`
  - `labunix03`
    `/opt/tomcat/webapps/your-computingID/WEB-INF/classes/your-package-folder/`

- Put .jsp files into
  - local
    `/tomcat/webapps/your-context-or-your-project-folder/`
  - `labunix03`
    `/opt/tomcat/webapps/your-computingID/`
Running JSPs

• Run your JSP from a web browser by entering the URL
  • Local
    http://localhost:8080/your-context-or-your-project-folder/your-jsp.jsp
  • labunix
    http://labunix03.cs.virginia.edu:8080/your-computingID/your-jsp.jsp

• A JSP is translated to a Java servlet, which is then compiled by the servlet engine. You don’t need to compile JSP.