Session 19

Introduction to Server-Side Scripting

Lecture Objectives

- Recognize that a server script provides a way to extend an HTML page to include logic and insertion of data
- Understand the operation of a template engine
- Understand how a template engine can translate a JSP into a servlet
Server Side Scripting

- Server side scripting - running scripts server-side to produce dynamic web page content before the page is sent to the user’s web browser.
- A template engine processes a scripted template to produce content that can be sent to the browser.

Examples:
- JSP
- ThymeLeaf
- PHP
- NodeJS
- JSF

One goal of server side scripting is to separate the view from data and business logic.

In CSE336, we don’t consider server side scripting languages not covered in the CS base programming sequence.

Translation Approaches

Examples:

- JSP
  - Translation
  - Servlet

- ThymeLeaf
  - Translation
  - HTML5

Translation to HTML5 allows the template to be designed using html design tools.

JSP capabilities are a subset of those of ThymeLeaf.
JavaServer Page (JSP)

- Large base of existing JSP pages, but little reason to do new development with JSPs
- Used to rapidly create dynamically-generated Web pages
- A JSP is:
  - A text-based document (filename extension of .jsp) that processes a request and constructs a response
  - Translated into a servlet

JSP Translation

- The Web container translates the JSP into the equivalent servlet (and compiles it into a servlet class)

```
MyJSP.jsp
  HTML
  scriptlets
  EL
  JSP elements

MyJSP.jsp.java
  Java code
```

Can include any static text (e.g., XML)

Translation

Generated code is placed in a sub-directory
HelloWorld.jsp

```java
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

Doesn't this look like an html file?

Hello World!

HelloWorld.jsp Generates a Web Page

```java
<%@page contentType="text/html" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

You can place the JSPs in the root or in any sub-directory (e.g., JSPs)

Hello World!
**Generated HelloWorld Servlet ...**

```java
private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();
private static java.util.List<String> _jspx_dependants;
private org.glassfish.jsp.api.ResourceInjector _jspx_resourceInjector;
public java.util.List<String> getDependants() {
    return _jspx_dependants;
}
```

... Generated HelloWorld Servlet ...

```java
public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;

    Note the predefined JSP variables

    When you use the identifier "session", it refers to this variable in the generated servlet
```
try {
    response.setContentType("text/html;charset=UTF-8");
    pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;
    _jspx_resourceInjector = (org.glassfish.jsp.api.ResourceInjector) application.getAttribute("com.sun.appserv.jsp.resource.injector");
    out.write("\n");
    out.write("<!DOCTYPE html>\n");
    out.write("<html>\n");
    out.write("    <head>\n");
    out.write("        <title>Hello World</title>\n");
    out.write("    </head>\n");
    out.write("    <body>\n");
    out.write("        <h1>Hello World!</h1>\n");
    out.write("    </body>\n");
    out.write("</html>\n");
}
catch (Throwable t) {
    if (! (t instanceof SkipPageException)) {
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
            out.clearBuffer();
        if (_jspx_page_context != null)
            _jspx_page_context.handlePageException(t);
        else throw new ServletException(t);
    }
} finally {
    _jspxFactory.releasePageContext(_jspx_page_context);
}
Example - HelloWorldToday

The JSP will display today's date

Hello World Today!

Today's date is: Wed Mar 22 16:45:50 EDT 2017

HelloWorldToday.jsp

... <html>  
<head>  
<title>Hello World Today</title>  
</head>  
<body>  
<h1>Hello World Today!</h1>  
<p>Today's date is: <%= new java.util.Date()%> </p>  
</body>  
</html>

Note the syntax of the JSP expression

This JSP expression translates to:

```
out.print( new java.util.Date());
```

This style of inserting code into your template is not effective.

© Robert Kelly, 2017-2018
Are We on Track?

Part 1 (Create a new jsp from an html file)
- Copy your project html
- Create a new jsp in your IDE
- Drop the html into your JSP source
- Run

Overview - JSP Page Contents

- HTML (or XML or ...)
- JSP constructs
  - Directives – control the overall structure of the servlet (page, include, and taglib directives)
  - Scripting elements
    - Expressions – inserted into servlet output
      `<%= expression %>`
    - Scriptlets – inserted into servlet code
      `<% code %>`
    - Declarations – inserted into body of servlet class
      `<%! code %>`
  - Actions – control behavior of the JSP engine

You can think of a JSP as an HTML page with escapes to insert dynamic data
Scripting Elements

- JSP tags that allow code to be embedded in a JSP page
- The code contained in these JSP scripting elements is inserted into the corresponding location of the JSP servlet
- JSP scripting elements include the following:
  - Expressions – single line of code
  - Scriptlets – blocks of code
  - Declarations – class level declarations (e.g., new instance variables/methods)
- Not considered to be a good programming practice – use servlets, beans, and custom tags for data and control
- Use of Expression Language (EL) is much better JSP practice – we cover this in the next session

JSP Scriptlets

- You can insert arbitrary code into the JSP
- Form: `<% Java Code %>`
- Scriptlets are sometimes used as a crude way to:
  - Set Http response headers and status codes
  - Update a database
  - Provide conditional code and loop

Scriptlets might be included in a CSE336 exam to demonstrate an understanding of JSP operation
Predefined Variables

- Also referred to as implicit objects, and includes
  - request – HttpRequest object
  - response – HttpResponse object
  - out – PrintWriter object
  - session – note that sessions are created automatically
  - application – the ServletContext object that can be used to store persistent data (using setAttribute and getAttribute methods)
  - config – servletConfig object
  - pageContext
  - page – not typically used by JSP authors

Be careful with the name inconsistency

What Does the Container Do With Your JSP?

- Looks at directives to determine if anything should be done at translation
- Creates an HttpServlet subclass
- Writes import statements into the servlet (if there is a page directive with an import attribute)
- Writes JSP declaration code
- Builds the service method
- Merges HTML, scriptlets, and expressions into the service method
JSP Summary

- JSPs are text-based documents that contain
  - Static template data (e.g., html, xml)
  - JSP elements (for constructing dynamic content) denoted by `<% ... %>`
- JSPs access dynamic data through objects that
  - Are provided with the environment (e.g., Session object)
  - You create (e.g., Java bean)
- JSPs can employ an alternate XML-based JSP syntax
- JSPs encapsulate the design view of a page (separate from code for dynamic actions, often contained in java beans and custom tags)

Have You Satisfied the Lecture Objectives

- Recognize that a server script provides a way to extend an html page to include logic and insertion of data
- Understand the operation of a template engine
- Understand how a template engine can translate a JSP into a servlet