Web Applications

- A web application uses enabling technologies to
  1. make web site contents dynamic
  2. allow users of the system to affect business logic on the server
- Web applications allow users to affect state on the server
  - Some rule out search engines because they do not affect the server's state

A web application is a program deployed on the web
Enabling Technologies

- Web server responses can be static or dynamic
  - Static: HTML document is retrieved from the file system and returned to the client
  - Dynamic: HTML document is generated by a program in response to an HTTP request
- Dynamic web pages are created by enabling technologies

An enabling technology is a mechanism that is used to make web pages interactive and responsive to user inputs

Server Side Processing

HTTP Request → data → browser → Client → HTML → Server → servlet container → HTTP Response → web server
Servlet Overview

Enabling Technologies—CGI

- CGI: Common Gateway Interface allows clients to execute applications on the server
- The first enabling technology
- CGI applications usually reside in a special “safe” directory
- Can be written in any language; PERL is most common
- CGI apps typically:
  1. process data
  2. modify server state
  3. return information (usually an HTML page)
Enabling Technologies

Problems with CGI

• CGI does not automatically provide session management services
• Each and every execution of a CGI module requires a new process on the web server
• Perl is a difficult language in which to write large applications

Solution: Plug-ins on the Web server

Enabling Technologies

Web Server Plug-ins

• A plug-in is an extension to a web server that allows a different program to handle certain types of requests
  – images, sound, video
  – compiled module applications
  – scripted page applications
• Plug-ins typically keep an active process as long as the web server is active
Enabling Technologies - Plug-ins

Compiled Modules

- Compiled modules are executable programs that the server uses
- Common compiled module application plug-ins:
  - Microsoft’s .NET ASP
  - Netscape Server API (NSAPI)
  - J2EE Java servlets
- Compiled modules are efficient and very effective
- They allow programmers to clearly separate the front-end from the back-end
  - Aids design
  - Complicates implementation

5 November 2013 © Offutt

Enabling Technologies - Plug-ins

Scripted Pages

- Scripted pages look like HTML pages that happen to process business logic
- Execution is on the server, not on the client
  - unlike JavaScripts
- They have HTML with program statements that get and process data
- JSPs are compiled and run as servlets
  - very clean and efficient

5 November 2013 © Offutt
Enabling Technologies - Plug-ins

Scripted Pages (2)

- Common scripted pages:
  - MacroMedia’s Cold Fusion
  - Microsoft’s Active Server Pages (ASP)
  - Java Server Pages (JSP)
- Scripted pages are generally easy to develop, deploy, and modify
- They mix logic with HTML, so can be difficult to read and maintain

Java Server Pages (JSP)

- Java Scripts provide client-side execution ability
  - Interpreted
  - Cumbersome and error prone
  - Non-portable
- Java Servlets provide server-side execution
  - Compiled
  - Portable
  - Robust
  - Not integrated with HTML – Java creates HTML
  - Mixes static (HTML) with dynamic (business logic)
  - “Java that creates HTML”
Java Server Pages (2)

• A JSP is a *scripted page* that mixes programming statements into HTML

• JSP scriptlets:
  – Have a Java-like syntax
  – Can use external objects and call methods
  – Can process form data

• JSPs are translated to Java servlets, then compiled

• This helps separate presentation from data

---

First Look at JSP Code

```html
<%@page import = "java.util.Date"%>
<HTML>
<BODY>
<CENTER>
<H1>Java Server Page Example</H1>
The current time is <%= new Date() %>
</CENTER>
</BODY>
</HTML>
```

[http://cs.gmu.edu:8080/offutt/jsp/642/date.jsp](http://cs.gmu.edu:8080/offutt/jsp/642/date.jsp)
JSP Processing – Simple View

1. request

2. Read

2. if no hello.jsp
2. timestamp

3. Run JSP
4. Response
3. if no or newer than
hello.jsp

hello.class

5. response

It's actually a little more complicated …

JSP execution – mental model of JSP developer

JSP Processing

1. request

2. Read

2. if no or
hello.jsp

3. Translate

2. timestamp

3. if no or
hello.jsp

4. Compile

5. Execute

hello.java

hello.class

JSP execution – actual implementation
JSP Servlet Methods

- jspInit()
- jspDestroy()
- _jspService()

Can be defined in JSP declarations
Called when servlet starts and ends

Servlet service()
Includes all Java from JSP

JSP Example – Counter

<BODY>
<!-- Set global information for the page -->
<%@ page language="java" %>
<!-- Declare the variable -->
<%! int count = 0; %>
<!-- Scriptlet - Java code -->
<% for (int i = 0; i < 10; i++) {
    count = count+1;
}%>
<br/>
The counter value is: <%= count %>
</BODY>
JSP Example

http://cs.gmu.edu/~offutt/classes/642/examples/jsp/

Just do one, save the others

JSP Elements

JSP syntax: <%%X ... %>> // 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
    •  (blank) Scriptlets: A block of Java code
        Can call methods
    •  = Expressions: Values to be printed

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

Messages sent to the JSP container

- `<%@ page attribute=value ... %>`
  - Page attributes are listed in book
  - You will usually use the defaults
- `<%@ include <filename> %>`
  - File inserted into the JSP inline before JSP is compiled
- `<%@ taglib uri="tagLibURI" prefix="tagPrefix" %>`

We won’t use these in this class

(2) JSP Scripts—Declarations

Java code to define class-level variables and methods

```java
<%! 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() when JSP is translated to Java
- Can access variables from the JSP Declaration
- Scriptlets can access servlet objects
  - request: HttpServletRequest object (our usual req)
  - response: HttpServletResponse object (our usual res)
  - out: for printing

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

Note that the name "request" must be used.

(2) JSP Scripts—Expressions

Abbreviated scriptlet print statement

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

- Dynamic: A web software component is run and the results are included as part of the response
  - A dynamic include can result in different content each time
  - `<jsp:include page="myjsp.jsp" flush="true" />`
    - `myjsp.jsp` is compiled
    - `myjsp.jsp` is executed
    - Output from `myjsp` is included in the current JSP
    - Current output is flushed before `myjsp` 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)

useBean causes a JavaBean object to be instantiated
useBean gives a name to the new object (id=)
useBean defines the scope
useBean declares the location (bean details)
(3) JSP Actions—Java Bean Example

- Syntax for using a bean:
  
  ```
  <%@ page import="jspexamples.*" %>
  <jsp:usebean id="letterColor" class="AlphabetCode" scope="page" />
  ```

  - ID name to use for object
    (AlphabetCode LetterColor = new … )
  - Name of class
  - JSPs offer several useful scopes for variables …

- Note that scope="application" allows Beans to be shared among different servlets — DON'T USE IT!
  - That can lead to interactions among each other … more later …

(3) JSP Actions—Properties

- `setProperty` gives a value to a property in a bean
  
  ```
  - <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
  
  ```
  - <jsp:getProperty name="langBean" property="language"/>
  ```

  - Equivalent to the call: `langBean.getLanguage();`

- Case of property name is very important
  
  - Property must begin with a lower case letter ("language")
  - Getters and setters must have the property name start with a capital letter (`setLanguage()`, `getLanguage()`)
(3) JSP Actions—Java Bean Summary

- Using Java Beans allows for more separation between the HTML and Java
- The Beans / Property pattern provides a very 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()”)
- It is not necessary for the bean to have an object with the name of the property

(3) JSP Actions—Forwarding

- jsp:Forward allows a request to be forwarded to another JSP

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

  - When this statement is reached, execution will jump to the JSP anotherPage.jsp
    - Control does not return to the original JSP
    - The request and response objects are forwarded
  - Use as a front-end when we need to decide which JSP to execute based on some input data
  - Use to authenticate users (see student info system example)
Deploying JSPs on Apps Cluster

It is unfortunately tricky to get JSPs to interface with Java Beans

- A JSP is translated to a Java servlet, which is then compiled by the servlet engine
- Therefore the bean has to be in a directory that is in the Java CLASSPATH of the servlet engine
- On our webapps server, the Java servlet engine CLASSPATH includes the directory where we put servlets:
  /data/swe642fall2013/swe642/WEB-INF/classes/
- Put bean .class files into your “package” directory

Deploying JSPs on Hermes

1. Import the bean into your JSP:
   `<%@ page import="username.*" %>`
2. Copy the bean’s .class file into the directory
   `/data/swe642fall2013/swe642/WEB-INF/classes/`
   `cp useBean.class /data/swe642fall2013/swe642/WEB-INF/classes/username/`
3. Copy a JSP file into your package directory in the JSP directory:
   `mkdir /data/swe642fall2013/swe642/jsp/YOURNAME`
   `cp useBean.jsp /data/swe642fall2013/swe642/jsp/YOURNAME/`
4. Now you can run your JSP from your browser by entering the URL:
   `http://apps-swe642.vse.gmu.edu:8080/swe642/jsp/YOURNAME.useBean.jsp`
5. Look for the Java versions of your JSPs in:
   `/usr/share/tomcat6/work/Catalina/apps-swe642.vsnet.gmu.edu/swe642/org/apache/jsp/YOURNAME`
JSP & Java Bean Examples

http://cs.gmu.edu/~offutt/classes/642/examples/jsp/