JSP architecture

- Advantages:
  - all the advantages of servlets
  - efficient, robust, secure, access to Java libraries
  - uses "page-centric" coding
    - HTML that contains embedded code

- Disadvantages:
  - minor performance issue first time a JSP is loaded
    - has to be compiled on-the-fly
    - only affects very first request to a JSP
    - after first request, compiled code is cached for faster execution

Topics

- JSP architecture
- JSP syntax
  - objects
  - directives
  - scripting elements
  - standard actions
  - expression language
- JSP and JavaBeans
- Tag libraries
- JSP deployment
- Model 1 and Model 2 JSP architectures
Not JSP syntax

- JSP is an embedded language like ASP, PHP

```jsp
<% String firstName = request.getParameter("FIRSTNAME"); out.println("hi " + firstName);
%>
```

```php
<?php
    $firstname = $HTTP_POST_VARS['FIRSTNAME'];
echo "hi ". $firstname . "\n";
?>
```

JSP syntax

- Java code is enclosed by `<% ... %>`
- Outside those tags, it is normal HTML
- Browser output:
  - URL = http://127.0.0.1:7001/MyApp/hw.jsp?FIRSTNAME=Wayne

```jsp
<% String firstName = request.getParameter("FIRSTNAME"); out.println("hi " + firstName);
%>
```

JSP Objects

- You can access Java objects from within JSP's
- Main types of objects
  - Implicit objects (provided by the container)
  - Application-specific objects
    - Local, instance & class variables
    - JavaBeans
    - EJBs
- Web container provides a context to provide scope

JSP implicit objects

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Type</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>javax.servlet.http.HttpServletRequest</td>
<td>Represents user's request</td>
</tr>
<tr>
<td>response</td>
<td>javax.servlet.http.HttpServletResponse</td>
<td>Represents user's response</td>
</tr>
<tr>
<td>pageContext</td>
<td>javax.servlet.jsp.PageContext</td>
<td>Contains attributes of this page</td>
</tr>
<tr>
<td>session</td>
<td>javax.servlet.http.HttpSession</td>
<td>Represents user's session</td>
</tr>
<tr>
<td>application</td>
<td>javax.servlet.ServletContext</td>
<td>Contains attributes for the entire application</td>
</tr>
</tbody>
</table>

JSP implicit objects

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Type</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>javax.servlet.jsp.jspWriter</td>
<td>Output stream for the response</td>
</tr>
<tr>
<td>config</td>
<td>javax.servlet.ServletConfig</td>
<td>Contains initialization parameters and the ServletContext</td>
</tr>
<tr>
<td>page</td>
<td>java.lang.Object</td>
<td>Current servlet object</td>
</tr>
<tr>
<td>exception</td>
<td>java.lang.Throwable</td>
<td>Only available for pages designated as error pages</td>
</tr>
</tbody>
</table>

Kinds of embedded JSP code

1. Directives
2. Scripting Elements
   - declarations
   - scriptlets
   - expressions
3. Actions
   - include
   - forward
   - useBean
   - get/setProperty
4. expression language (EL)
JSP Directives

• Note: `<%@   %>` tags

• Directives are instructions to the JSP compiler
  - e.g. "include", includes at compile-time, NOT run-time

• Main directives
  - page
  - include
  - taglibs (later)

Scripting Elements: Declarations

• Declarations executed when page is initialised
  - used to define class-wide variables and methods
  - declarations must produce no output

```plaintext
<%! int i = 0; %>
<%! int icubed = 0; %>
<%! public int cubed (int j) {
  return (j * j * j);
}
%
```

Scripting Elements: Expressions

• Expressions are primarily for inserting values of Java variables into HTML code without having to type in full as a scriptlet:
  - eg: `<% out.print(i); %>` vs `<%= i %>`

```html
<BODY>
  <p>You entered the value <%= i %></p>
  <p>That number cubed is <%= icubed %></p>
  <p>I can say the same thing with <%= cubed(i) %></p>
</BODY>
```

Standard Actions

• Well-known tags that affect the runtime behaviour
  • Generates the Java code that corresponds to the required task - during conversion

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

• Other standard actions
  - `<jsp:useBean>`
  - `<jsp:setProperty>`
  - `<jsp:getProperty>`
  - etc.
### Standard Actions

```jsp```
<jsp:forward page="MyOtherServlet" />
<jsp:include page="MyIncludedServlet" />
<jsp:useBean id="myperson"
    class="au.edu.uts.it.PersonBean"
    scope="session" />
```

- Uses XML syntax
- Actions are run-time behaviours
  - "jsp:include" includes a page at run-time

### Expression Language

- Expression Language (EL) introduced with JSP 2.4 & J2EE 1.4
- It complements JSP Expressions with simpler syntax. Each `$(object)` would be same as `<%= object %>`
- Supports:
  - JSP implicit objects
  - JavaBeans
  - Simple calculations

### Expression Language

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>param</td>
<td>Request params &amp; values</td>
<td><code>$(param.newcol)</code></td>
</tr>
<tr>
<td>header</td>
<td>HTTP headers</td>
<td><code>$(header.Host)</code></td>
</tr>
<tr>
<td>cookie</td>
<td>Cookie names &amp; value</td>
<td><code>$(cookie[&quot;id&quot;])</code></td>
</tr>
<tr>
<td>initParam</td>
<td>Init params in web.xml</td>
<td><code>$(initParam.name)</code></td>
</tr>
<tr>
<td>pageContext</td>
<td>Use this to get request, response, session</td>
<td><code>$(pageContext.request.remoteHost)</code></td>
</tr>
</tbody>
</table>

- Use `pageContext` to get servlet parameter, eg: the example is same as `<%= request.getRemoteHost() %>`

### EL operators

- EL also has some common programming operators:
  - Math: `+ - * / div% mod`
  - Relational: `== eq != ne < lt > gt <= le >= ge`
  - Negative: `-`
  - Logical: `&& and || or ! not`
  - Empty: `empty (used to test for null or "")`

- You can also access Java collections using array syntax eg:
  - `$(myarray[1])`

### JSP Execution

- In practice, JSPs are converted into servlets for execution
- Conversion involves:
  1. Container generates Java source code
  2. Container invokes Java compiler
  3. Class file is executed as with a normal servlet
  4. A copy of the class file is kept (cached) to avoid unnecessary recompilation in future

- See for yourself!
  - Weblogic specific:
    `/home/chw/weblogic/AdminServer/tmp/_WL_user/_appsdir_labs_war/fr8116/jsp_servlet/__jspfilename.java`) !!!

### Topics

- JSP architecture
- JSP syntax
  - Objects
  - Directives
  - Scripting elements
  - Standard actions
  - Execution
- JSP and JavaBeans
- Tag libraries
- JSP deployment
- Model 1 and Model 2 JSP architectures
Page View with Bean

- Refinement

Using JavaBeans

- JavaBeans are a client-side component model
  - not to be confused with Enterprise JavaBeans
- Allows you to encapsulate reusable functionality in a "bean"
  - a Bean is a normal Java class that follows some rules
    - called "design patterns"
- One basic design pattern is for "properties"
  - using a getter and/or a setter method
  - more complex patterns too - see a JavaBeans tutorial

Using JavaBeans

public class Person {
    private String yourname;
    public String getName() {
        return yourname;
    }
    public void setName(String yourname) {
        this.yourname = yourname;
    }
    public void otherMethod() {
        // ...
    }
}

• Note: method naming convention is important

Using JavaBeans

- Using beans
  - <jsp:useBean id="fred" class="com.myapp.Person" scope="page" />
- Setting properties
  - <jsp:setProperty name="fred" property="name" value="Fred Jones" />
    - this invokes Person.setName("Fred Jones")
- Getting properties
  - <jsp:getProperty name="fred" property="name" />
    - this invokes Person.getName()

JSP Scopes

- JSP allows you to save data into a scope
- Scopes can be one of:

<table>
<thead>
<tr>
<th>Scope</th>
<th>Where saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>Only for the current page, destroyed when you leave the page.</td>
</tr>
<tr>
<td>request</td>
<td>kept while request is active.</td>
</tr>
<tr>
<td>session</td>
<td>saved in the current session.</td>
</tr>
<tr>
<td>application</td>
<td>saved for the whole application</td>
</tr>
</tbody>
</table>

EL and JavaBeans

- Expression Language also can access JavaBeans
- Instead of:

  `<jsp:getProperty name="fred" property="name" />`
  
  `\{note: this invokes Person.getName() \}`

  use:

  `$\{fred.name\}$` instead
Topics

- JSP architecture
- JSP syntax
  - objects
  - directives
  - scripting elements
  - standard actions
  - execution
- JSP and JavaBeans
- → Tag libraries
- JSP deployment
- Model 1 and Model 2 JSP architectures

Tag libraries

- Create your own custom tags that can be used in documents
  - encapsulate reusable functionality
  - simplify page coding
  - Apache Jakarta project has large range of taglibs
    http://jakarta.apache.org/taglibs
  - J2EE 1.4 adopted some of these as the Java Standard Tag Library (JSTL)
    http://java.sun.com/products/jsp/jstl

```jsp
<%@ taglib uri="/hello" prefix="examples" />
<html>
<p>Hello message is:<examples:hello />
</html>
```

Tag libraries

- Four things needed:
  1. in JSP file:
```jsp
<%@ taglib uri="/hello" prefix="examples" />
<html>
<p>Hello message is:<examples:hello />
</html>
```

  2. Update web.xml with the taglib declaration
```xml
<web-app>
  <taglib>
    <taglib-uri>/hello</taglib-uri>
    <taglib-location>/WEB-INF/hellotags.tld</taglib-location>
  </taglib>
</web-app>
```

  3. Create a Tag Library Descriptor (TLD) file in WEB-INF directory: (hellotags.tld)
```xml
<taglib>
  <tag>
    <name>hello</name>
    <tagclass>HelloTag</tagclass>
  </tag>
</taglib>
```

  4. Create Java class that implements the tag handler
```java
public class HelloTag extends TagSupport {
  public int doStartTag() throws JspTagException { ... } 
}
```

Topics

- JSP architecture
- JSP syntax
  - objects
  - directives
  - scripting elements
  - standard actions
  - execution
- JSP and JavaBeans
- Tag libraries
- → JSP deployment
- Model 1 and Model 2 JSP architectures

JSP deployment

- For a simple JSP, just include it in a WAR file, no need to put anything in web.xml file

- If you want to call your JSP by a different name, you can map it in web.xml, e.g.
```xml
<servlet>
  <servlet-name>MyFirstJSP</servlet-name>
  <jsp-file>foobar.jsp</jsp-file>
</servlet>
```

```xml
<servlet-mapping>
  <servlet-name>MyFirstJSP</servlet-name>
  <url-pattern>/foobar/*</url-pattern>
</servlet-mapping>
```
JSP deployment – tag libraries

- For tag libraries, need to specify in web.xml:

```xml
<web-app>
  <taglib>
    <taglib-uri>/hello</taglib-uri>
    <taglib-location>/WEB-INF/tlds/hellotags.tld</taglib-location>
  </taglib>
</web-app>
```

WAR file directory structure

**Same as servlets:**
- Put JSP same place as html/gif etc, ie: top level directory or subdirectory
- *NOT* in WEB-INF

**Not same as servlets:**
- WEB-INF/classes
  - Java classes required to implement tag libs or supporting classes
- WEB-INF/lib
  - JAR files containing additional classes for tag libs
- WEB-INF/tlds
  - contains TLDs (not mandatory though) - conventional
  - not tag handler classes

JSTL

- J2EE 1.4 provides Java Standard Tag Library (JSTL)
  - `<%@ taglib prefix="c" url="http://java.sun.com/jsp/jstl/core" %>`
- Core tags:
  - `<c:set>` `<c:remove>` set/remove attributes from JSP
  - `<c:catch>` catch Java errors!
  - `<c:forEach>` equivalent of Java `for` loop
  - `<c:forTokens>` iterate over tokens in string
  - `<c:if>` equivalent to Java `if` but no else!
  - `<c:choose>` `<c:when>` `<c:otherwise>` equivalent to Java `switch`
  - `<c:out value="\${x}" >` like `\${x}` but escapes XML chars

JSTL – more libraries

- More JSTL tag libraries:
  - just use @taglib with the following parameters:

<table>
<thead>
<tr>
<th>Function</th>
<th>prefix</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML processing</td>
<td>x</td>
<td>xml</td>
</tr>
<tr>
<td>Internationalisation, formatting</td>
<td>fmt</td>
<td>fmt</td>
</tr>
<tr>
<td>SQL access</td>
<td>sql</td>
<td>sql</td>
</tr>
<tr>
<td>String, Collection functions</td>
<td>fn</td>
<td>functions</td>
</tr>
</tbody>
</table>

JSTL example

- Scriptlet based
  - `<ul>`
  - `<% for (int i=0; i<arr.length; i++) { %>
    - `<li>` `<%= arr[i] %>`
    - `<% } %>`
  - `<% } %>`
  - `<;/ul>`

- JSTL based
  - `<ul>`
  - `<c:forEach var="msg" items="\${arr}">`
    - `<li>` `<c:out value="\${msg}" >` like `\${msg}` but escapes XML chars
  - `<;/c:forEach>`

- No Java code needed!!
- This also works if arr[] was a Java collection!

Topics

- JSP architecture
- JSP syntax
  - objects
  - directives
  - scripting elements
  - standard actions
  - execution
- JSP and JavaBeans
- Tag libraries
- JSP deployment
- Model 1 and Model 2 JSP architectures
**Web application architectures**

- In theory, you can use servlets and JSPs interchangeably
- However current best practice is to design the web part of your application according to a particular architecture
  - Model 1 architecture
  - Model 2 architecture

**JSP Model 1 Architecture**

1. Browser makes HTTP request to JSP
   - JSP performs control logic
2. JSP may make use of JavaBeans for accessing either state information or database data
3. JavaBean may retrieve its data from an external data source (optional)
4. JSP finishes execution and returns result to browser

**Limitations of Model 1**

- Model 1 applications have no structured way of keeping track of "what to do next"
  - JSPs can call each other in random patterns
- Model 1 applications also have no separation between "design" and "coding" functions
  - Typically these are performed by different people
- Model 1 is okay for simple applications
- For larger applications it becomes hard to keep track of JSP interdependences

**Model-View-Controller (MVC)**

- A design pattern for building applications that interact with a user (not web-specific, not Java-specific)
- **Model**
  - Code for internal representation of system state (data)
- **View**
  - Code for displaying information to user (GUI)
- **Controller**
  - Code for changing system state (logic)

**JSP Model 2 Architecture**

- Model 2 = MVC for web applications
JSP Model 2 Arch (cont’d)

1. Browser makes HTTP request to controller servlet
2. Controller servlet instantiates any JavaBeans that might be needed for data displayed to user
3. JavaBean may retrieve external data (parallel to step 4)
4. Controller forwards request to appropriate JSP
   JSP only contains presentation/view code
5. JSP accesses JavaBean data set up by controller
6. Execution finishes and result is returned to browser

Advantages of Model 2

- Introducing a "controller" provides more structure to the application
- Also provides clearer separation of "design" and "coding"
  - web designers create JSPs without much coding
  - programmers write the controller servlet without being concerned by design issues
- A large application may contain many controllers
  - e.g. one for each use case / piece of functionality

Model 2 Architecture Summary

- Building web applications using the Model 2 architecture is current best practice

- We’ll revisit Model 2 again when we consider EJBs
  - EJBs change the picture, but only slightly

- If you design a three-tier application using Model 2, migrating it to an N-tier application (with EJBs) should involve minimal code changes
  - changes only to the JavaBeans, in theory

Model 2 Implementations

- The Struts framework http://struts.apache.org/ is a common implementation of MVC architecture.
- Consists of a controller servlet + config file + JSP taglib
- You write View(JSP), Action and Model classes

Summary

- JSP has many similarities to servlets
- Again a fair amount of information to know about
  - use the documentation as a reference
- Many ways to modularise and simplify coding
  - include directive
  - forward and include actions
  - tag libraries, especially Apache Jakarta & JSTL
- Tag libraries complex, but useful for big projects
- Model 2 is best practice application architecture