Trust. Collaboration. Innovation.

ISO 9001
BUREAU VERITAS
Certification

BUREAU VERITAS
1828

J2EE Training

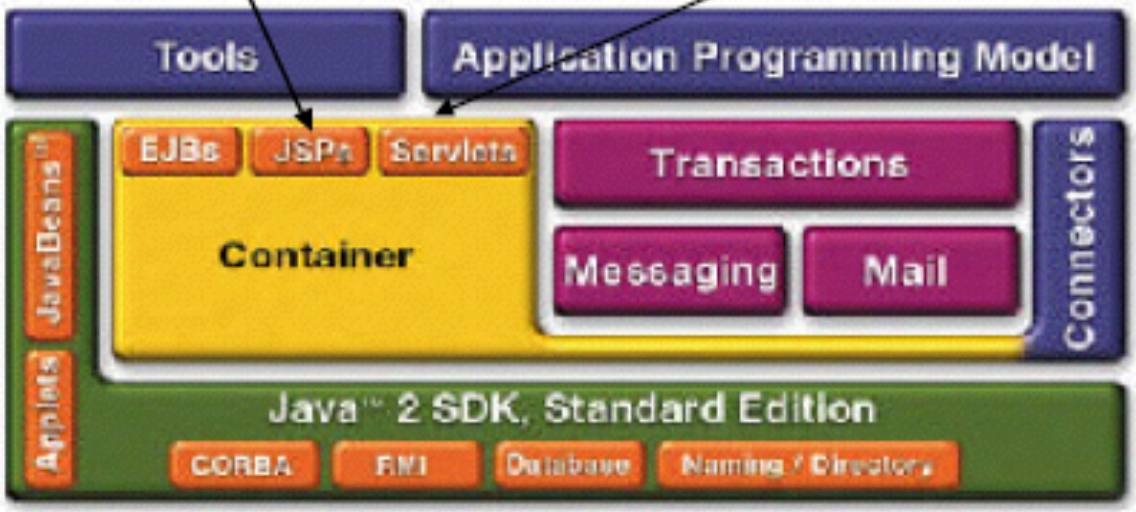
Agenda

- JSP in big picture of Java EE
- JSP Architecture
- Life-cycle of JSP page
- Steps for developing JSP-based Web application
- Dynamic contents generation techniques in JSP
- Invoking Java code using JSP scripting elements
- JavaBeans for JSP
- Error handling

JSP in a big picture Of J2EE

An extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to **return dynamic content to a client**
Typically the template data is HTML or XML elements.
The client is often a **Web browser**

Java Servlet A Java program that extends the functionality of a Web server, generating dynamic content and interacting with Web clients using a **request-response paradigm**



Static and Dynamic Contents?

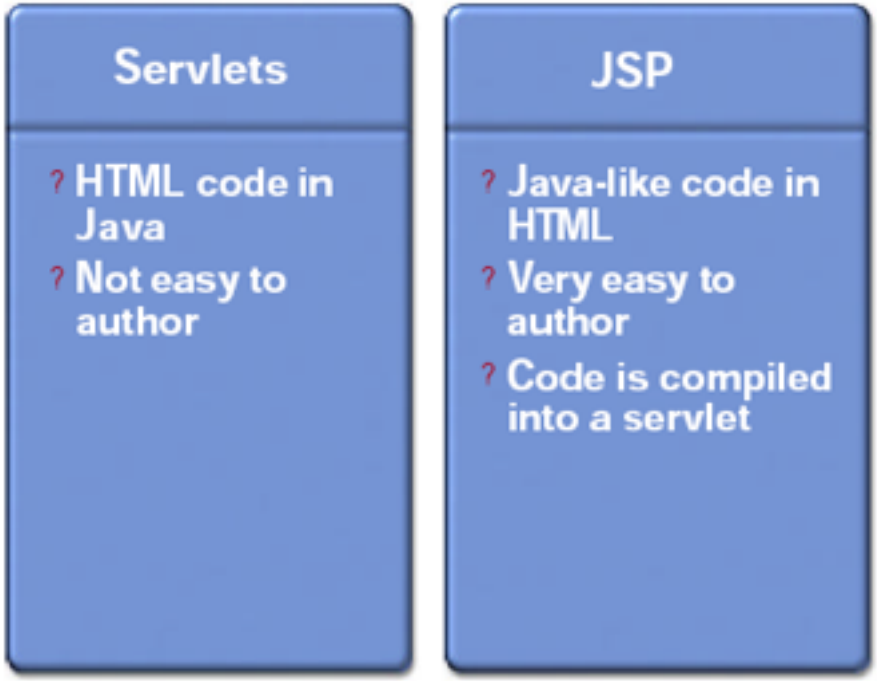
- Static contents
  - Typically static HTML page
  - Same display for everyone
- Dynamic contents
  - Contents is dynamically generated based on conditions
  - Conditions could be
    - User identity
    - Time of the day
    - User entered values through forms and selections

What is JSP Page?

- A text-based document capable of returning both static and dynamic content to a client browser
- Static content and dynamic content can be intermixed
- Static content
  - HTML, XML, Text
- Dynamic content
  - Java code
  - Displaying properties of JavaBeans
  - Invoking business logic defined in Custom tags

Sample JSP Page (blue – static Red – dynamic contents)

```
<html>
<body>
    Hello World!
    <br>
    Current time is <%= new java.util.Date() %>
</body>
</html>
```


Servlets
? HTML code in Java
? Not easy to author
JSP
? Java-like code in HTML
? Very easy to author
? Code is compiled into a servlet

JSP Benefits

> Content and display logic are separated
> Simplify web application development with JSP, JavaBeans and custom tags
> Supports software reuse through the use of components (JavaBeans, Custom tags)
> Automatic deployment
  – Recompile automatically when changes are made to JSP pages
> Easier to author web pages
> Platform-independent

Why JSP over Servlet?

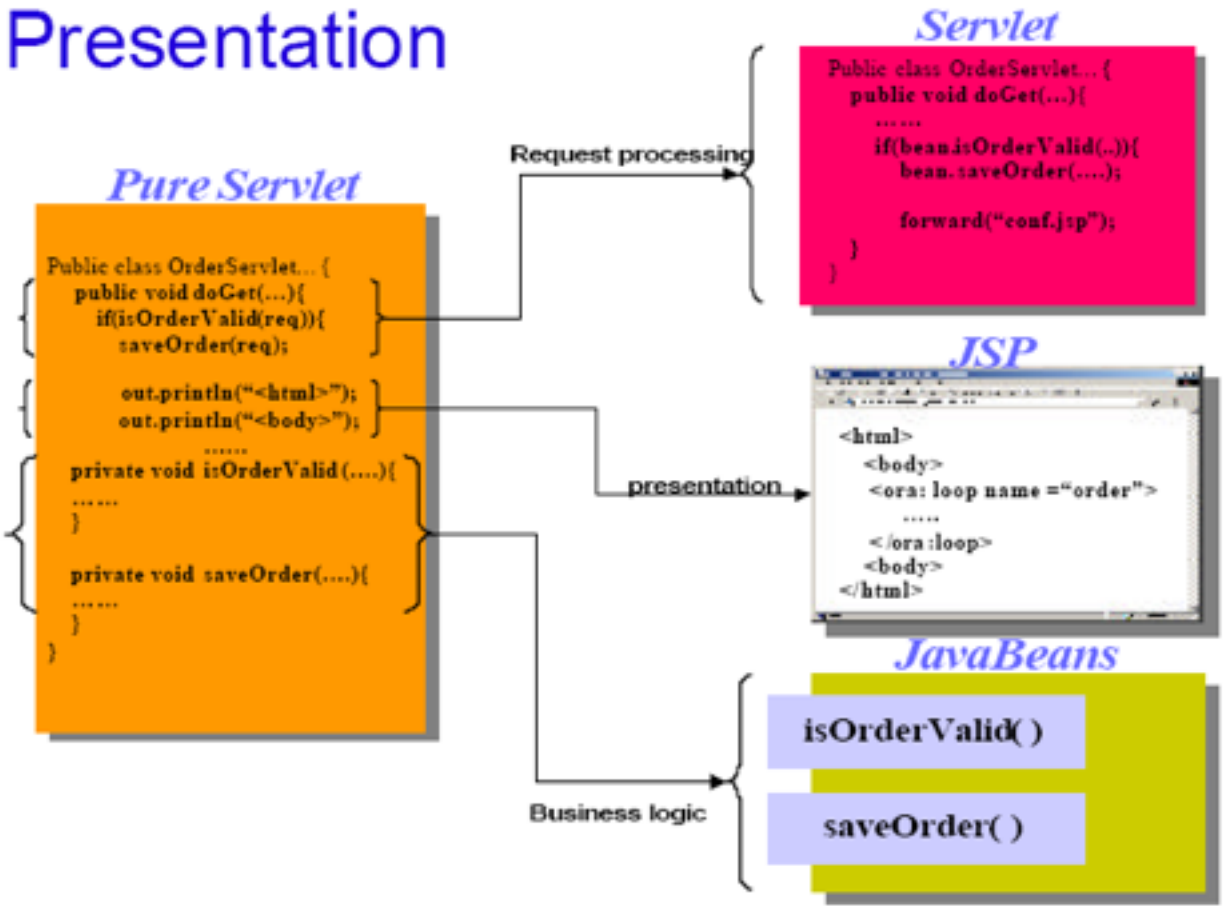
> Servlets can do a lot of things, but it is pain to:
  - Use those println() statements to generate HTML page
  - Maintain that HTML page
> No need for compiling, Packaging, CLASSPATH setting

Do I have to Use JSP over Servlet or vice-versa?

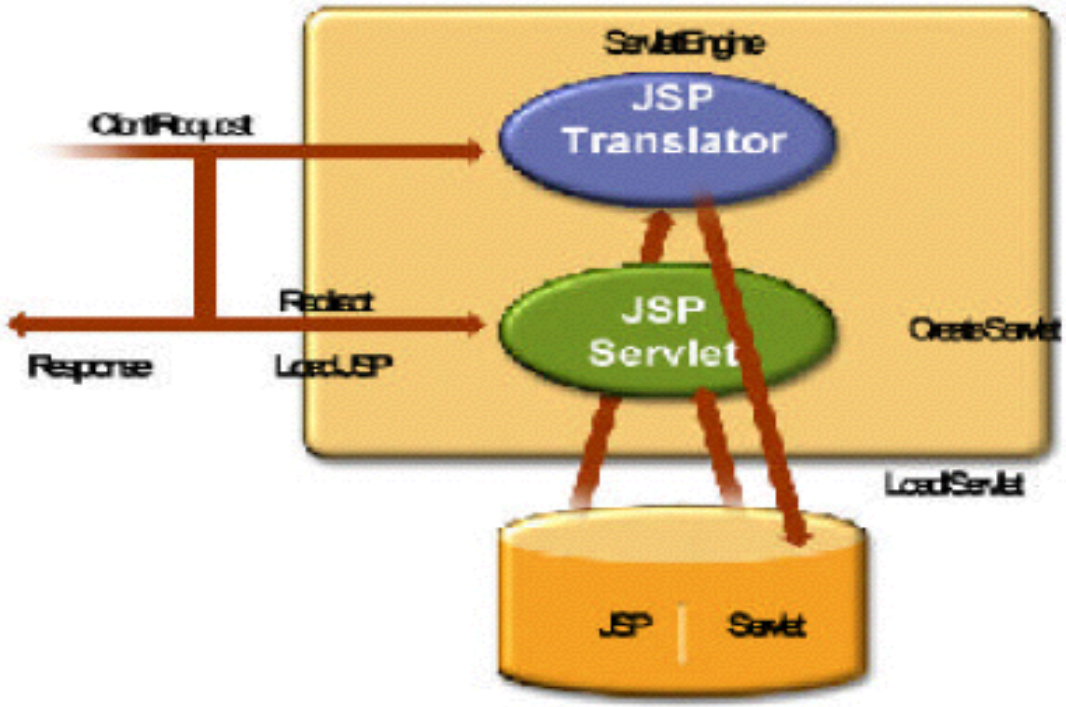
> No, you want to use both leveraging the strengths of each technology
  - Servlet's strength is “controlling and dispatching”
  - JSP's strength is “displaying”
> In a typically production environment, both servlet and JSP are used in a so-called MVC (Model-View-Controller) pattern
  - Servlet handles controller part
  - JSP handles view part

JSP Architecture

Separate Request processing From Presentation

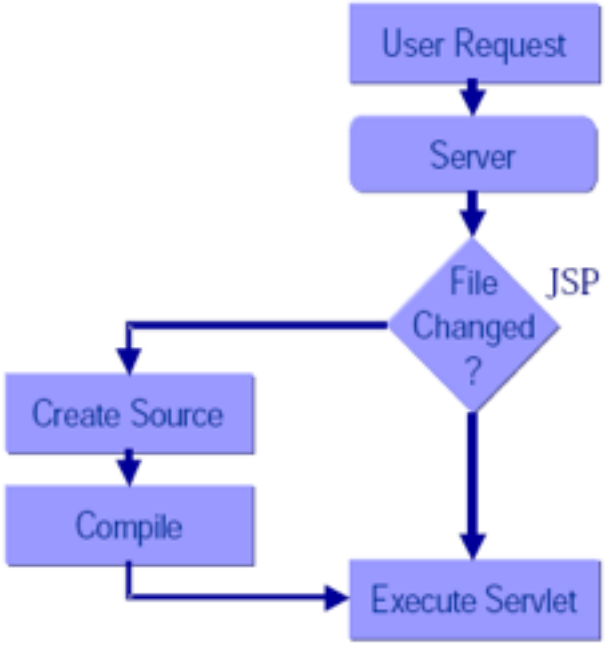


JSP Architecture



JSP Life Cycle

How Does JSP Work?



JSP Page Lifecycle Phases

> Transalation Phase
> Compile Phase
> Execution Phase

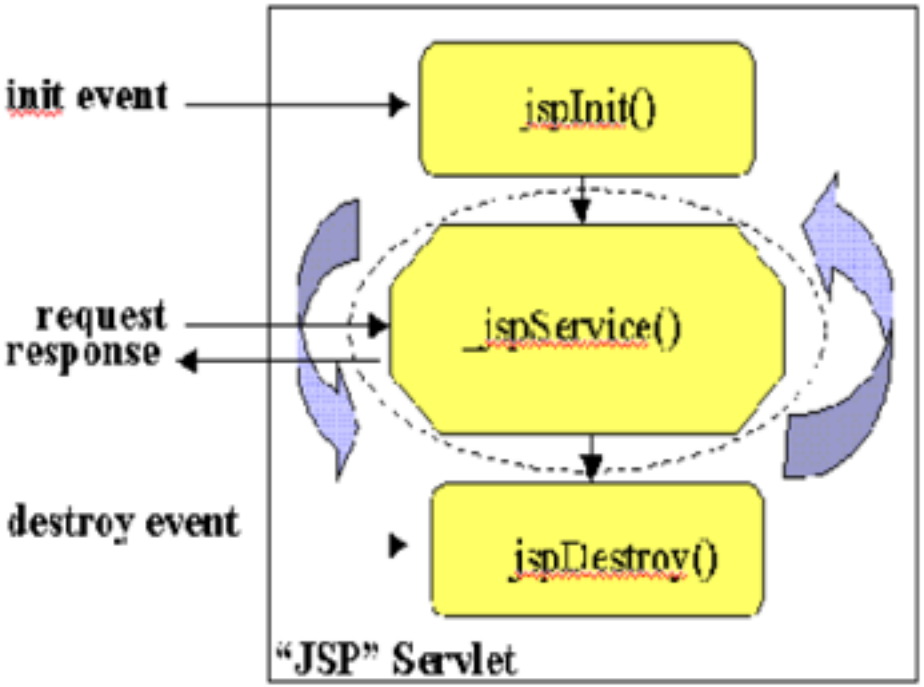
Translation/Compilation Phase

> JSP files get translated into servlet source code, which is then compiled
> Done by the container automatically
> The first time JSP page is accessed after it is deployed (or modified and redeployed)
> For JSP page "pageName", the source code resides
  - <AppServer_HOME>/work/Standard Engine/localhost/context_root/pageName$jsp.java
  - <AppServer_HOME>/work/Standard Engine/localhost/date/index$jsp.java

Translation/Compilation Phase

> Static Template data is transformed into code that will emit data into the stream
> JSP elements are treated differently
– Directives are used to control how Web container translates and executes JSP page
– Scripting elements are inserted into JSP page's servlet
class
– Elements of the form <jsp:xxx .../> are converted into
method calls to JavaBeans components

JSP Lifecycle Methods during Execution Phase



Steps for Development

Web Application Development and Deployment Steps

1. Write (and compile) the Web component code (Servlet or JSP) and helper classes referenced by the web component code

2. Create any static resources (for example, images or HTML pages)

Web Application Development and Deployment Steps

3.Create deployment descriptor (web.xml)

4.Build the Web application (*.war file or deployment-ready directory)

5.Install or deploy the web application into a Web container

> Clients (Browsers) are now ready to access them via URL

Comparing Servlet And JSP Code

GreetingServlet.java (1)

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * This is a simple example of an HTTP Servlet.  It responds to the GET
 * method of the HTTP protocol.
 */
public class GreetingServlet extends HttpServlet {

    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException
 {

      response.setContentType("text/html");
      response.setBufferSize(8192);
      PrintWriter out = response.getWriter();

      // then write the data of the response
      out.println("<html>" +
                  "<head><title>Hello</title></head>");
```

GreetingServlet.java (2)

```
    // then write the data of the response
    out.println("<body  bgcolor=\"#ffffff\">" +
       "<img src=\"duke.waving.gif\">" +
       "<h2>Hello, my name is Duke. What's yours?</h2>" +
       "<form method=\"get\">" +
       "<input type=\"text\" name=\"username\" size=\"25\">" +
       "<p></p>" +
       "<input type=\"submit\" value=\"Submit\">" +
       "<input type=\"reset\" value=\"Reset\">" +
       "</form>");

    String username = request.getParameter("username");

    // dispatch to another web resource
    if ( username != null && username.length() > 0 ) {
          RequestDispatcher dispatcher =
             getServletContext().getRequestDispatcher("/response");

          if (dispatcher != null)
          dispatcher.include(request, response);

    }
    out.println("</body></html>");
    out.close();
}
```

greeting.jsp

```
<html>
<head><title>Hello</title></head>
<body bgcolor="white">
<img src="duke.waving.gif">
<h2>My name is Duke. What is yours?</h2>

<form method="get">
<input type="text" name="username" size="25">
<p></p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>

<%
    String username = request.getParameter("username");
    if ( username != null && username.length() > 0 ) {
%>
    <%@include file="response.jsp" %>
<%
    }
%>
</body>
</html>
```

ResponseServlet.java

```
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

// This is a simple example of an HTTP Servlet.  It responds to the GET
// method of the HTTP protocol.
public class ResponseServlet extends HttpServlet {

    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException{
        PrintWriter out = response.getWriter();

        // then write the data of the response
        String username = request.getParameter("username");
        if ( username != null && username.length() > 0 )
          out.println("<h2>Hello, " + username + "!</h2>");
    }

    public String getServletInfo() {
        return "The Response servlet says hello.";
    }
 }
```

response.jsp

```
<h2><font color="black">Hello, <%=username%>!</font></h2>
```

Dynamic Content Generation Techniques in JSP

Techniques

a) Call Java code directly within JSP
b) Call Java code indirectly within JSP
c) Use JavaBeans within JSP
d) Develop and use your own custom tags
e) Leverage 3rd-party custom tags or JSTL (JSP Standard Tag Library)
f) Follow MVC design pattern
g) Leverage proven Model2 frameworks

(a) Call Java code directly

- Place all Java code in JSP page
- Suitable only for a very simple Web application
  - hard to maintain
  - hard to reuse code
  - hard to understand for web page authors
- Not recommended for relatively sophisticated Web applications
  - weak separation between contents and presentation

(b) Call Java code indirectly

> Develop separate utility classes
> Insert into JSP page only the Java code needed to invoke the utility classes
> Better separation of contents generation from presentation logic than the previous method
> Better re usability and maintainability than the previous method
> Still weak separation between contents and presentation, however

(c) Use JavaBeans

> Develop utility classes in the form of JavaBeans
> Leverage built-in JSP facility of creating JavaBeans instances, getting and setting JavaBeans properties
  – Use JSP element syntax
> Easier to use for web page authors
> Better re usability and maintainability than the previous method

(d) Develop and Use Custom Tags

> Develop sophisticated components called custom tags
  - Custom tags are specifically designed for JSP
> More powerful than JavaBeans component
  - More than just getter and setter methods
> re usability, maintainability, robustness
> Development of custom tags are more difficult than creating JavaBeans, however

(e) Use 3rd-party Custom tags or JSTL

> There are many open source and commercial custom tags available
  - Apache Struts
> JSTL (JSP Standard Tag Library) standardize the set of custom tags that should be available over Java EE platform at a minimum
  - As a developer or deployer, you can be assured that a standard set of custom tags are already present in Java EE compliant platform (J2EE 1.3 and after)

(f) Design/Use MVC Design Pattern

> Follow MVC design pattern
  - Model using some model technologies
  - View using JSP
  - Controller using Servlet
> Creating and maintaining your own MVC framework is highly discourage, however

(g) Use Proven MVC Model2 Frameworks

> There are many to choose from
  - Apache Struts
  - JavaServer Faces (JSF)
  - Other frameworks: Echo, Tapestry, WebWorks, Wicket

Invoking Java Code with JSP Scripting Elements

JSP Scripting Elements

- Lets you insert Java code into the servlet that will be generated from JSP page
- Minimize the usage of JSP scripting elements in your JSP pages if possible
- There are three forms
  - Expressions: <%= Expressions %>
  - Scriptlets: <% Code %>
  - Declarations: <%! Declarations %>

Expressions

> During execution phase
  - Expression is evaluated and converted into a String
  - The String is then Inserted into the servlet's output stream directly
  - Results in something like out.println(expression)
  - Can use predefined variables (implicit objects) within expression

Expressions

- Format
  - <%= Expression %> or
  - <jsp:expression>Expression</jsp:expression>
  - Semi-colons are not allowed for expressions

Scriptlets

- Used to insert arbitrary Java code into servlet's jspService() method
- Can do things expressions alone cannot do
  - setting response headers and status codes
  - writing to a server log
  - updating database
  - executing code that contains loops, conditionals
- Can use predefined variables (implicit objects)
- Format:
  - <% Java code %> or
  - <jsp:scriptlet> Java code</jsp:scriptlet>

Declarations

- Used to define variables or methods that get inserted into the main body of servlet class
  - Outside of _jspSevice() method
  - Implicit objects are not accessible to declarations
- Usually used with Expressions or Scriptlets
- For initialization and cleanup in JSP pages, use declarations to override jspInit() and jspDestroy() methods

Declarations

> Format:
  - <%! method or variable declaration code %>
  - <jsp:declaration> method or variable declaration code </jsp:declaration>

Including and Forwarding to Other Web Resource

Including Contents in a JSP Page

- Two mechanisms for including another Web resource in a JSP page
  - include directive
  - jsp:include element

Include Directive

- Is processed when the JSP page is translated into a servlet class
- Effect of the directive is to insert the text contained in another file-- either static content or another JSP page--in the including JSP page
- Used to include banner content, copyright information, or any chunk of content that you might want to reuse in another page
- Syntax and Example
  - <%@ include file="filename" %>
  - <%@ include file="banner.jsp" %>

jsp:include Element

- Is processed when a JSP page is executed
- Allows you to include either a static or dynamic resource in a JSP file
  - static: its content is inserted into the calling JSP file
  - dynamic: the request is sent to the included resource, the included page is executed, and then the result is included in the response from the calling JSP page
- Syntax and example
  - <jsp:include page="includedPage" />
  - <jsp:include page="date.jsp"/>

Which One to Use it?

- Use include directive if the file changes rarely
  - It is faster than jsp:include
- Use jsp:include for content that changes often
- Use jsp:include if which page to include cannot be decided until the main page is requested

Forwarding to another Web component

- Same mechanism as in Servlet
- Syntax
  - <jsp:forward page="/main.jsp" />
- Original request object is provided to the target page via jsp:parameter element

```
<jsp:forward page="..." >
<jsp:param name="param1" value="value1"/>
</jsp:forward>
```

Directives

Directives

- Directives are messages to the JSP container in order to affect overall structure of the servlet
- Do not produce output into the current output stream
- Syntax
  - <%@ directive {attr=value}* %>

Three Types

> page: Communicate page dependent attributes and communicate these to the JSP container
  - <%@ page import="java.util.* %>
> include: Used to include text and/or code at JSP page translation-time
  - <%@ include file="header.html" %>
> Taglib: Indicates a tag library that the JSP container should interpret
  - <%@ taglib uri="mytags" prefix="code" %>

Implicit Objects

> A JSP page has access to certain implicit objects that are always available, without being declared first
> Created by container
> Corresponds to classes defined in Servlet

Implicit Objects

> request (HttpServletRequest)
> response (HttpServletRepsonse)
> session (HttpSession)
> application(ServletContext)
> out (of type JspWriter)
> config (ServletConfig)
> pageContext

Java Beans

What Are Java Beans?

> Java classes that can be easily reused and composed together into an application
> Any Java class that follows certain design conventions can be a JavaBeans component
  - properties of the class
  - public methods to get and set properties
> Within a JSP page, you can create and initialize beans and get and set the values of their properties
> JavaBeans can contain business logic or data base access logic

JavaBeans Design Conventions

- JavaBeans maintain internal properties
- A property can be
  - Read/write, read-only, or write-only
  - Simple or indexed
- Properties should be accessed and set via getXxx and setXxx methods
  - PropertyClass getProperty() { ... }
  - PropertyClass setProperty() { ... }
- JavaBeans must have a zero-argument (empty) constructor

Why Use JavaBeans in JSP Page?

> A JSP page can create and use any type of Java programming language object within a declaration or scriptlet like following:

```
<%
    ShoppingCart cart =
(ShoppingCart)session.getAttribute("cart");
    // If the user has no cart, create a new one
    if (cart == null) {
        cart = new ShoppingCart();
        session.setAttribute("cart", cart);
    }
%>
```

Why Use JavaBeans in JSP Page?

> JSP pages can use JSP elements to create and access the object that conforms to JavaBeans conventions

```
<jsp:useBean id="cart" class="cart.ShoppingCart"
scope="session"/>
```

Create an instance of “ShoppingCart” if none exists, stores it as an attribute of the session scope object, and makes the bean available throughout the session by the identifier “cart”

Why Use JavaBeans in JSP Page?

> No need to learn Java programming language for page designers
> Stronger separation between content and presentation
> Higher re usability of code
> Simpler object sharing through built-in sharing mechanism
> Convenient matching between request parameters and object properties

Creating a JavaBeans

> Declare that the page will use a bean that is stored within and accessible from the specified scope by jsp:useBean element

```
<jsp:useBean id="beanName"
    class="fully_qualified_classname"
scope="scope"/>
or
<jsp:useBean id="beanName"
    class="fully_qualified_classname"
scope="scope">
    <jsp:setProperty .../>
</jsp:useBean>
```

Setting JavaBeans Properties

> 2 ways to set a property of a bean
> Via scriptlet
  - <% beanName.setPropName(value); %>
> Via JSP:setProperty
  - <jsp:setProperty name="beanName" property="propName" value="string constant"/>
  - "beanName" must be the same as that specified for the id attribute in a useBean element
  - There must be a setPropName method in the bean

Setting JavaBeans Properties

> jsp:setProperty syntax depends on the source of the property
  - <jsp:setProperty name="beanName" property="propName" value="string constant"/>
  - <jsp:setProperty name="beanName" property="propName" param="paramName"/>
  - <jsp:setProperty name="beanName" property="propName"/>
  - <jsp:setProperty name="beanName" property="*"/>
  - <jsp:setProperty name="beanName" property="propName" value="<%= expression %>"/>

Getting JavaBeans Properties

> 2 different ways

- via scriptlet
  - > <%= beanName.getPropName() %>
- via JSP:setProperty
  - > <jsp:getProperty name="beanName" property="propName"/>

> Requirements

– "beanName" must be the same as that specified for the id attribute in a useBean element

– There must be a "getPropName()" method in a bean

Handling Errors

Handling Errors

- Determine the exception thrown
- In each of your JSP, include the name of the error page
  - <%@ page errorPage="errorpage.jsp" %>
- Develop an error page, it should include
  - <%@ page isErrorPage="true" %>
- In the error page, use the exception reference to display exception information
  - <%= exception.toString() %>

Thank You!!!