

Java Server Pages (JSP)

TLS

Rewards and Recognition



Objectives

At the end of this session, you will be able to:

- Understand need for JSP
- Describe JSP Architecture
- Demonstrate JSP Tags
- Demonstrate Implicit Objects
- Demonstrate Action Tags
- Demonstrate use of Java Bean in JSP Page
- Describe Beans Scope.
- Compare JSP and Servlets
- Understand and use JSTL



Agenda

Following topics is to be covered in the session,

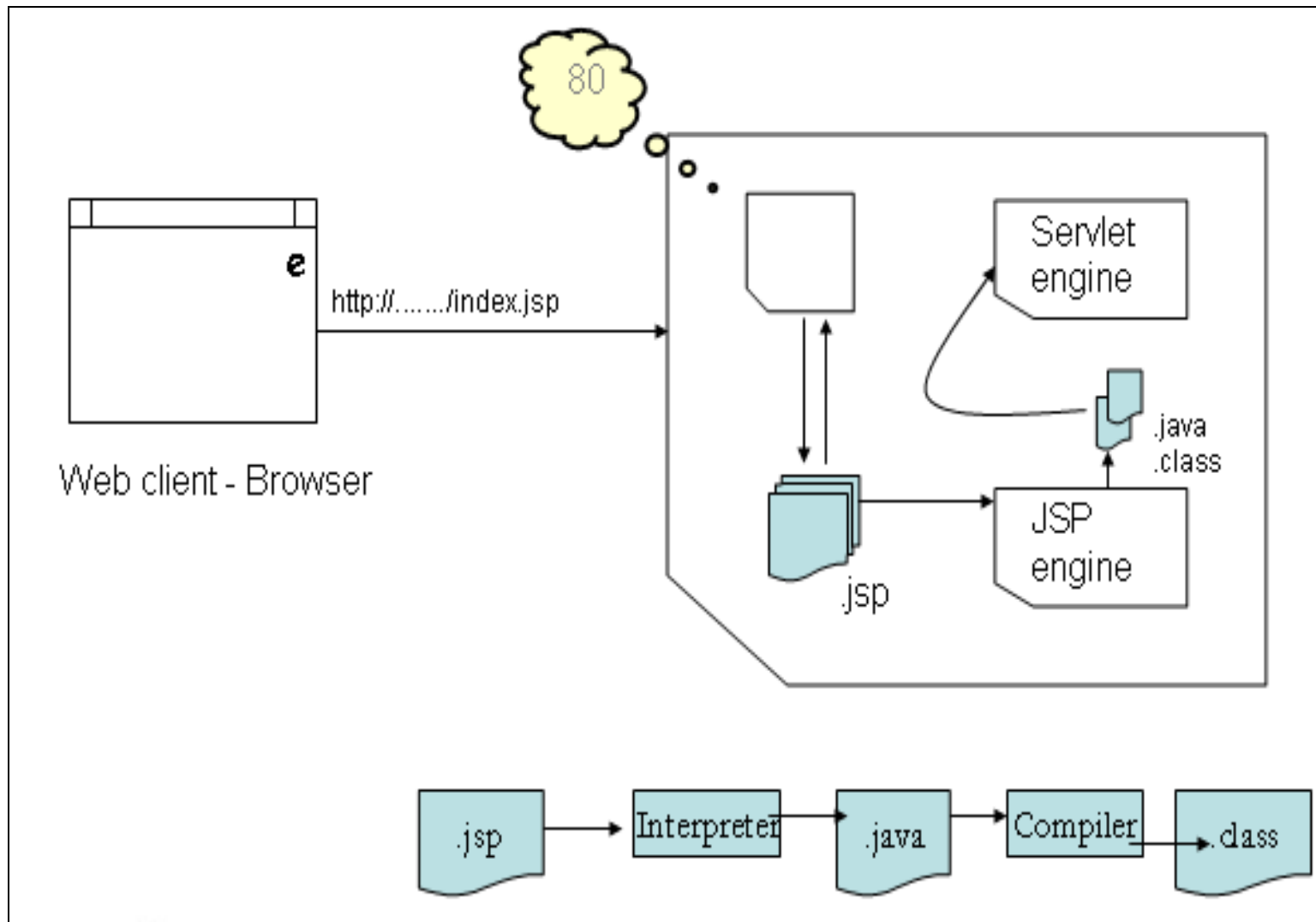
- Need for JSP
- JSP Architecture
- JSP Tags
- Implicit Objects
- Action Tags
- useBean Action Tag
- Bean's Scope
- JSP vs Servlets



Need .. JSP

- Server side technology for generating HTML dynamically
- Allows separation of presentation logic from business logic
- Compiled and Type-safe

JSP Architecture



Translating and Executing JSP Pages

- A JSP page is executed in a JSP container, generally installed in a Web server
- The underlying semantic model is that of a servlet
- A typical JSP container will translate the JSP page to a Java servlet
- By default, translation and compilation of a JSP page is likely to occur the first time the page is accessed

JSP Features

- Standard directives guiding translation of a JSP page to a servlet
- Standard actions in the form of predefined JSP tags
- Script language - declarations, scriptlets, and expressions for including Java (or other language) fragments that embed dynamic content
- A portable tag extension mechanism, for building tag libraries — effectively extending the JSP language

JSP Tags

▪ Directives

`<%@` `%>`

- Information applicable to the whole page

▪ Declaratives

`<%!` `%>`

- Gets converted as members of the resulting servlet class
- Variables declared become global variables
- Methods become user-defined methods

▪ Scriptlets

`<%` `%>`

- Gets inserted into the service method of the resulting servlet

▪ Expressions

`<%=` `%>`

- Gets inserted into the service method of the resulting servlet
- Value is evaluated at run time and displayed on the browser in String format

JSP Tags .. Examples

- Expression Tag

```
<%= java.util.Date() %>
```

The current system date will be displayed on browser

- Directive Tags

```
<%@ page import="java.util.*" %>  
<%= new Date() %>
```

Importing package through page directive tag

- Scriptlet

```
<% if(Math.random()<0.5){ %>  
<b>Hi</b>  
<% }else{ %>  
<b>Bye</b>  
<% } %>
```

Importing package in Scriptlet

Directive Tags

- There are three main types of directive:
- page, which lets you
 - import classes
 - customize the servlet superclass
 - `<%@ page session="false" %>`
- include, which lets you
 - insert a file into the servlet class at the time the JSP file is translated into a servlet
 - `<%@ include file = " " %>`
- taglib directive
 - indicates a library of custom tags that the page can include
 - `<%@ taglib uri=" " prefix=" " %>`

page Directive... Attributes

- The page directive lets you define the following attributes:
 - import="package.class"
 - `<%@ page import="java.util.*" %>`
 - contentType="MIME-Type"
 - `<%@ page contentType="text/plain" %>`
- It is the same as
 - `<% response.setContentType("text/plain"); %>`
- isThreadSafe="true|false"
 - Normal servlet processing or implementing SingleThreadModel
- session="true|false"
 - Allowing/disallowing sessions
- buffer="sizekb|none"
 - specifies the buffer size for the JspWriter out
- autoflush="true|false"
 - Flush buffer when full or throws an exception when buffer is full

page Directive... Attributes

- extends="package.class"
- info="message"
 - A message for the getServletInfo method
- errorPage="url"
 - Define a JSP page that handles uncaught exceptions
- isErrorPage="true|false"
- language="java"

The include directive

- The include directive includes the content of the named file directly into your JSP's source, as it is compiled.

```
<%@ include file="filename" %>
```

The include directive differs from the `jsp:include` action as `action` includes the output of a page into the output stream at request processing time

- The include directive is useful for including reusable static content into your JSP, such as common footer and header elements.

Example of Jsp Include

```
<html>
  <head><title><%= title %></title></head>
    <%! String title = "Addition"; %>
  <body>
    <%@ include file="header.html" %>
    The sum of <%= request.getParameter("x") %>
    and <%= request.getParameter("y") %> is
    <%= Integer.parseInt(request.getParameter("x")) +
    Integer.parseInt(request.getParameter("y")) %>
    <%@ include file="footer.html" %>
  </body>
</html>
```

JSP declarations

- Declarations declare new data and function members for use within the JSP.
- These declarations become part of the resulting servlet class generated during page translation.
- You can write a JSP declaration in two ways:

`<%! java declarations %>`

Example of JSP Declarations

```
<%! int add(String x, String y) {  
    return Integer.parseInt(x)+Integer.parseInt(y);  
}  
%>
```

```
<html>  
  <head><title>Addition</title></head>  
  <body>  
    The sum of <%= request.getParameter("x") %>  
  
    and <%= request.getParameter("y") %> is  
  
    <%= add(request.getParameter("x"),  
    request.getParameter("y")) %>  
  </body>  
</html>
```


JSP expressions

- The text `<%= new java.util.Date() %>` is a JSP expression.
- Expressions in Java code are statements that result in a value. A JSP expression is a Java expression evaluated at request time, converted to a String, and written into the output stream.
- JSP expressions are written either as:
`<%= a-java-expression %>`

JSP Expressions

```
<html>
```

```
<head><title>Addition</title></head>
```

```
<body>
```

The sum of **`<%= request.getParameter("x") %>`**

and **`<%= request.getParameter("y") %>`** is

**`<%= Integer.parseInt(request.getParameter("x")) +
Integer.parseInt(request.getParameter("y")) %>`**

```
</body>
```

```
</html>
```

JSP Scriptlets

- Scriptlets are what their name implies: They are small sets of statements written in the scripting language, which, means writing Java code in tags
- Scriptlets are executed at request processing time. Whether they produce any output into the output stream depends on the code in the scriptlet.
- Scriptlets are written as:
`<% java-statements %>`

JSP Statements/Scriptlets

```
<html>
<head><title>Numbers</title></head>
<body>
    <% int n =
    Integer.parseInt (request.getParameter("n"));
    for (int i=0; i<n; i++)
    out.println("<li>" + i + "</li>");
    %>
</body>
</html>
```

Implicit Objects (Predefined Variables)

- The following predefined variables can be used:
 - request, the HttpServletRequest
 - response, the HttpServletResponse
 - session, the HttpSession associated with the request (if any)
 - out, the PrintWriter (a buffered version of type JspWriter) used to send output to the client
- Predefined variables are also called as Implicit Variables
- The previous scriptlet example can be rewritten like this using implicit objects

```
<% if(Math.random()<0.5) %>
{
    out.println("HI");
}
else
{
    out.println("BYE");
}
%>
```

Implicit Objects (Predefined Variables)

- As we have seen before, there are variables that can be used in the code
- There are eight automatically defined variables, sometimes called implicit objects
- The available variables are

Variable Name

class

- | | |
|---------------|-----------------------------------|
| ▪ request | javax.servlet.HttpServletRequest |
| ▪ response | javax.servlet.HttpServletResponse |
| ▪ out | javax.servlet.jsp.JspWriter |
| ▪ session | javax.servlet.http.HttpSession |
| ▪ application | javax.servlet.ServletContext |
| ▪ config | javax.servlet.ServletConfig |
| ▪ pageContext | javax.servlet.jsp.PageContext |
| ▪ page | SS1
java.lang.Object |
| ▪ Exception | java.lang.Throwable |

Slide 22

SS1

to change alignment

Sanjana Sane, 9/8/2016

Implicit Objects

request object

- This is the `HttpServletRequest` associated with the request
- It lets you
 - look at the request parameters (via `getParameter`),
 - the request type (GET, POST, HEAD, etc.), and
 - the incoming HTTP headers (cookies, etc.)

response object

- This is the `HttpServletResponse` associated with the response to the client
- The output stream is buffered,
- Thus, it is legal to set HTTP status codes and response headers, even though this is not permitted in regular servlets once any output has been sent to the client

Implicit Objects

out object

- This is the PrintWriter used to send output to the client
- This is a buffered version of PrintWriter called JspWriter
- You can adjust the buffer size, or even turn buffering off by using the buffer attribute of the page directive

session object

- This is the HttpSession object associated with the request
- Sessions are created automatically, so this variable is bound even if there was no incoming session reference - unless session was turned off using the session attribute of the page directive

Implicit Objects

application object

- This is the ServletContext as obtained via `getServletConfig().getContext()`
- Servlets and JSP pages can hold constant data in the ServletContext object
- Getting and setting attributes is with `getAttribute` and `setAttribute`
- The ServletContext is shared by all the servlets in the server

config

- we can also pass configuration parameters that are specific to a JSP page,
- which the page can retrieve using the implicit variable `config`

Implicit Objects

page

- The implicit variable page is of class `java.lang.Object`, and it refers to the
- instance of the generated servlet from jsp

pagecontext

- The `pageContext` variable is of type `javax.servlet.jsp.PageContext`.
- The `PageContext` class is an abstract class, and the JSP engine vendor provides its concrete subclass

exception

- This implicit variable is of type `java.lang.Throwable`. The exception variable
- is available to the pages that act as error handlers for other pages.
- that error-handler pages have the page directive attribute `isErrorPage` set to `true`.

JSP Actions

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine
- You can
 - dynamically insert a file
 - reuse JavaBeans components
 - forward the user to another page
 - generate HTML for the Java plugin

Action Tags

- Available actions include
 - jsp:include - Includes a file at the time the page is requested
 - jsp:useBean - Finds or instantiate a JavaBean
 - jsp:setProperty - Sets the property of a JavaBean
 - jsp:getProperty - Inserts the property of a JavaBean into the output
 - jsp:forward - Forwards the requester to a new page

The jsp:include Action

- This action lets you insert files into the page being generated
- The file inserted when page is requested
- Syntax

```
<jsp:include page="relative URL"
```

```
flush="true" />
```

The jsp:forward Action

- Forwards request to another page
- Syntax:
`<jsp:forward page="relative URL"/>`
- It has a single attribute, page, which should consist of a relative URL
- This could be a static value, or could be computed at request time
- Examples:
`<jsp:forward page="/utils/errorReporter.jsp" />`
`<jsp:forward page="<%= someJavaExpression %>" />`

Using Beans

- JavaBeans are reusable software components that can be manipulated visually in a builder tool
 - Introspection – analyze how the bean works
 - Customization – the user can customize the look of the bean
 - Events support
 - Properties support
 - Persistence – allowing saving the bean on a persistent mechanism and loading it

Bean Class ... facts

- The bean class should include a constructor with no arguments
- The bean class should not include public variables (fields)
- Access to the attributes of the bean is through methods that look like
 - getName / setName for non-boolean fields
 - isName / setName for boolean fields

Using a Bean

```
<jsp:useBean id="name" class="package.class" />
```

- Create an object of the given class
- Bind the object to a variable with the given name

Using a Bean

```
<jsp:useBean id="db"  
class="dbiClasses.WebDatabase" />
```



```
<% dbiClasses.WebDatabase db = new  
dbiClasses.WebDatabase(); %>
```

Using a Bean

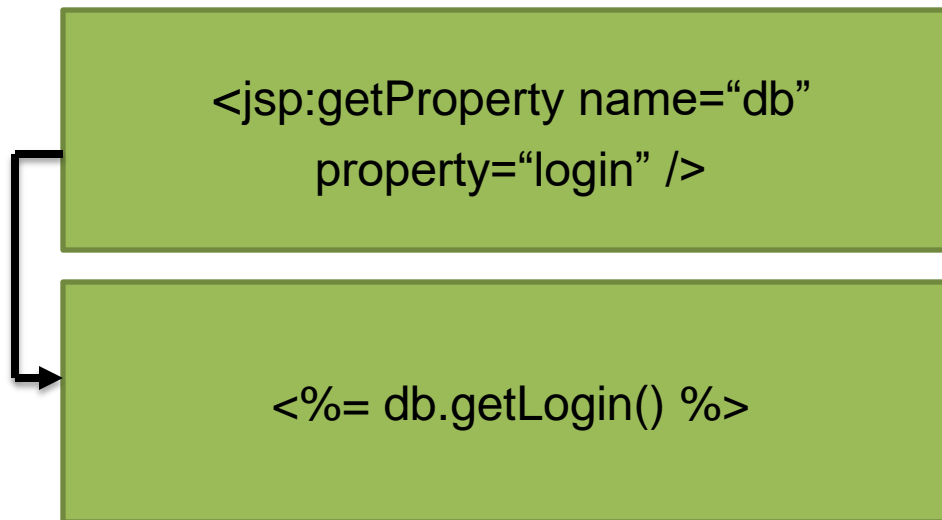
```
<jsp:useBean id="handler"  
class="ConnectionHandler" type="Runnable"/>
```



```
<% Runnable handler = new  
ConnectionHandler(); %>
```

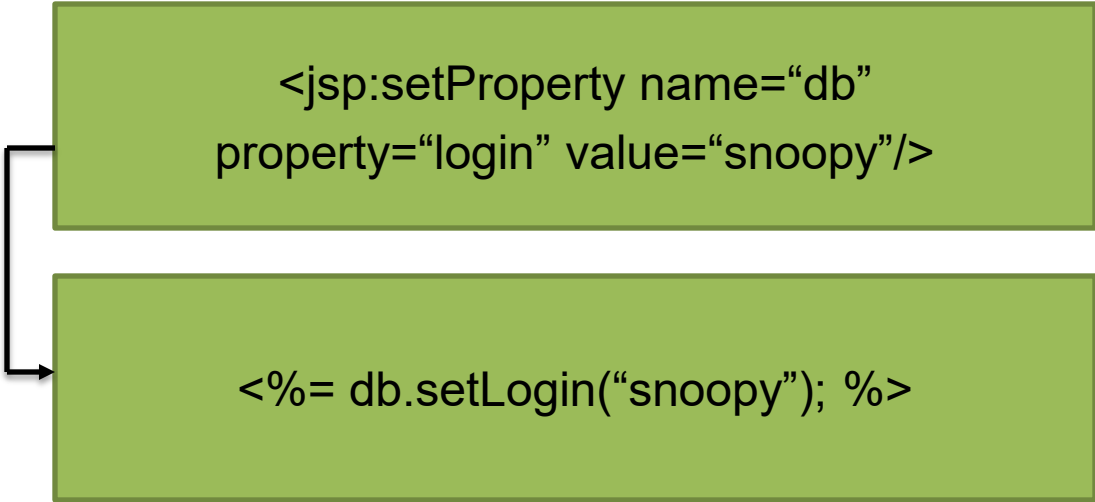
Getting the Properties of a Bean

- Use `jsp:getProperty` to get a property of a bean
- Use the attributes `name` and `property` to get the requested property



Setting the Properties of a Bean

- Use `jsp:setProperty` to set a property of a bean
- Use the attributes `name`, `property` and `value` to set the value to the property



```
<jsp:setProperty name="db"  
property="login" value="snoopy"/>
```

The diagram consists of two green rectangular boxes stacked vertically. A black arrow originates from the left side of the top box and points to the left side of the bottom box, indicating a comparison or equivalence between the two code snippets.

```
<%= db.setLogin("snoopy"); %>
```

Example

```
package dbiTests;

public class SimpleBean {

    private String message="No message specified";

    public String getMessage() {
        return(message);
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```

JSP Code

```
<HTML>
<HEAD> <TITLE>Reusing JavaBeans in JSP</TITLE>
</HEAD>

<BODY>
<CENTER> <TABLE BORDER=5>
<TR><TH CLASS="TITLE"> Reusing JavaBeans in JSP
</TABLE> </CENTER>
<P>
<jsp:useBean id="test" class="dbiTest.SimpleBean" />
<jsp:setProperty name="test" property="message"
value="Hello WWW" />
<H1>Message: <I>
<jsp:getProperty name="test" property="message" />
</I></H1>
</BODY>
</HTML>
```


Getting the Values from the Request

```
<jsp:setProperty name="db"  
property="login"  
value='<%=  
request.getParameter("login")%>'  
>
```

Getting the Parameter Automatically

- If we want that the value of the parameter dblogin of the request will be set to the bean db automatically we can use

```
<jsp:setProperty name="db"  
property="login"  
value="dblogin" />
```

Shared Beans

- So far, we assumed that object that were created using `jsp:useBean` were bound to to the local variables in the `_jspService` method
- The `scope` attribute of `jsp:useBean` gives us more possibilities
- Scope of the bean can be
 - page
 - request
 - session
 - application

Object scopes

Scope describes what entities can access the object. Objects can be defined in any of the following scopes:

- **Page**

- Objects with page scope are accessible only within the page where they are created.
- References to objects with page scope are stored in the pageContext object.

- **Request**

- Objects with request scope are accessible from pages processing the same request where they were created.
- References to objects with request scope are stored in the request object and are released after the request is processed.
- In particular, if the request is forwarded to a resource in the same runtime, the object is still reachable.

Object Scope cont....

■ Session

- Objects with session scope are accessible from pages processing requests in the same session as the one in which they were created.
- It is illegal to define an object with session scope from within a page that is not session-aware.
- All references to the object are released after the associated session ends.

■ Application

- Objects with application scope are accessible from pages processing requests in the same application as the one in which they were created.
- The application object is the servlet context obtained from the servlet configuration object.

JSP Vs Servlet

TLS

Rewards and Recognition



Init and Destroy

- JSP pages, like regular servlets, sometimes want to use init and destroy
- **Problem:** the servlet that gets built from the JSP page might already use init and destroy
 - Overriding them would cause problems
 - Thus, it is illegal to use JSP declarations to declare init or destroy
- **Solution:** use `jspInit` and `jspDestroy`
 - The auto-generated servlet is guaranteed to call these methods from init and destroy
 - The standard versions of `jspInit` and `jspDestroy` are empty (placeholders for you to override)

Difference between JSP and Servlets

- JSP is Java code embedded in HTML
- Servlets are compiled pure Java class files
- In servlets both the Presentation and business logic are placed together.
- In jsp both are separated by defining java beans
- Servlet can support HTTP, FTP, and SMTP protocol
- Jsp only supports HTTP protocol
- Both require a container on the server, such as Tomcat, which provides the environment and VM for the Java program.

JSP & Servlets

- But JSP pages are converted to Servlets? Aren't they the same?

- **Similarities**

- Provide identical results to the end user
- JSP is an additional module to the Servlet Engine

- **Differences**

- Servlet: "HTML in Java code"
 - HTML code inaccessible to Graphics Designer
 - Everything accessible to Programmer
- JSP: "Java Code Scriptlets in HTML"
 - HTML code accessible to Graphics Designer
 - Java code accessible to Programmer

- **Best Practices** : It is recommended to avoid pure java code inside JSP and use tags as much as possible due to maintenance issue of java code.
- For achieving this we can use JSTL or scripting language or custom tags.
- JSL or custom tags also serves the Reusability feature to java programs.



JSP Standard Template Library

TLS

Rewards and Recognition



JSTL

- The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications.
- JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.
- The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:

JSTL Core Tags

- The core group of tags are the most frequently used JSTL tags.
- Following is the syntax to include JSTL Core library in your JSP:

```
<%@ taglib prefix="c"  
    uri="http://java.sun.com/jsp/jstl/core" %>
```

JSTL Core Tags

- Following are Core JSTL Tags

Tag	Description
<c:out >	Like <%= ... >, but for expressions.
<c:set >	Sets the result of an expression evaluation in a 'scope'
<c:catch>	Catches any Throwable that occurs in its body and optionally exposes it.
<c:if>	Simple conditional tag which evaluates its body if the supplied condition is true.
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<c:when>	Sub-tag of <choose> that includes its body if its condition evaluates to 'true'.

JSTL Core Tags (Cont.)

Tag	Description
<code><c:otherwise ></code>	Subtag of <code><choose></code> that follows <code><when></code> tags and runs only if all of the prior conditions evaluated to 'false'.
<code><c:import></code>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
<code><c:forEach ></code>	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .
<code><c:forEachToken s></code>	Iterates over tokens, separated by the supplied delimiters.
<code><c:param></code>	Adds a parameter to a containing 'import' tag's URL.
<code><c:redirect ></code>	Redirects to a new URL.

<c:out> Tag

- The <c:out> tag displays the result of an expression, similar to the way <%= %> works
- The difference is that <c:out> tag lets you use the simpler "." notation to access properties.
- For example,
 - To access customer.address.street just use
 - <c:out value="customer.address.street"/>
- **Attribute:**

Attribute	Description	Required
value	Information to output	Yes

<c:out> Tag

- **Example:**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

- `<html>`
- `<head>`
- `<title><c:out> Tag Example</title>`
- `</head>`
- `<body>`
- `<c:out value="${<tag> , &}"/>`
- `</body>`
- `</html>`

- This would produce following result:

`<tag> , &`

<c:set> Tag

- The <c:set> tag is JSTL-friendly version of the setProperty action
- The tag is helpful because it evaluates an expression and uses the results to set a value of a JavaBean

- **Attribute:**

Attribute	Description	Required
value	Information to save	No
var	Name of the variable to store information	No
scope	Scope of variable to store information	No

<c:set> Tag

▪ Example:

- `<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>`
- `<html>`
- `<head> <title><c:set> Tag Example</title> </head> <body>`
- `<c:set var="amt" scope="session"`
`value="${2000*2}"/>`
- `<c:out value="${amt}"/>`
- `</body>`
- `</html>`

▪ This would produce following result:

- 4000

<c:if> Tag

- The <c:if> tag evaluates an expression and displays its body content only if the expression evaluates to true.
- **Attribute:**

Attribute	Description	Required
test	Condition to evaluate	Yes
var	Name of the variable to store the condition's result	No
scope	Scope of the variable to store the condition's result	No

<c:if> Tag

- **Example:**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
    prefix="c" %> <html>
```

```
<head> <title><c:if> Tag Example</title>
```

```
</head>
```

```
<body>
```

```
    <c:set var="amt" scope="session"
        value="${2000*2}"/>
```

```
    <c:if test="${amt > 2000}">
```

```
        <p>My amt is: <c:out value="${amt}"/><p>
```

```
    </c:if>
```

```
</body>
```

```
</html>
```

- This would produce following result:

- My amt is: 4000

<c:choose>, <c:when>, <c:otherwise>

- The <c:choose> works like a Java **switch** statement
- Lets you choose between a number of alternatives.
- Like the **switch** statement has **case** statements, the <c:choose> tag has <c:when> tags.
- Like a switch statement has **default** clause to specify a default action, similar way <c:choose> has <c:otherwise> as default clause.

<c:choose>, <c:when>, <c:otherwise>

- **Attribute:**

- The <c:choose> tag does not have any attribute.
- The <c:when> tag has one attributes which is listed below.
- The <c:otherwise> tag does not have any attribute.

- The <c:when> tag has following attributes:

Attribute	Description	Required
test	Condition to evaluate	Yes

<c:choose>, <c:when>, <c:otherwise>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<html>
```

```
<head> <title><c:choose> Tag Example</title> </head>
```

```
<body>
```

```
    <c:set var="amt" scope="session" value="${2000*2}"/>    <p>Your Bill Amount is : <c:out  
value="${amt}"/></p>    <c:choose>
```

```
    <c:when test="${amt <= 0}">
```

```
        Low Bill Amount
```

```
    </c:when>
```

```
    <c:when test="${amt > 1000}">
```

```
        Bill amount is very good
```

```
    </c:when>
```

```
    <c:otherwise> No comments.. </c:otherwise>
```

```
</c:choose>
```

```
</body>
```

```
</html>
```

- This would produce following result:

Your amt is : 4000 Bill amount is very good.

<c:forEach> Tag

- This tag exist as a good alternative to embedding a Java **for**, **while**, or **do-while** loop using scriptlet
- The <c:forEach> tag is the more commonly used tag because it iterates over a collection of objects

Attribute	Description	Required
begin	Element to start with (0 = first item, 1 = second item, ...)	No
end	Element to end with (0 = first item, 1 = second item, ...)	No
var	Name of the variable to expose the current item	No

<c:forEach> Tag

▪ Example:

- `<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %> <html>`
- `<head> <title><c:forEach> Tag Example</title> </head>`
- `<body>`
- `<c:forEach var="i" begin="1" end="5">`
- `Item <c:out value="{i}"/><p>`
- `</c:forEach>`
- `</body>`
- `</html>`

▪ This would produce following result:

- Item 1
- Item 2
- Item 3
- Item 4
- Item 5

<c:redirect> Tag

- The <c:redirect> tag redirects the browser to an alternate URL by providing automatically URL rewriting

Attribute	Description	Required
url	URL to redirect the user's browser to	Yes
context	/ followed by the name of a local web application	No

<c:redirect> Tag

- Example :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title><c:redirect> Tag Example</title>
  </head>
  <body>
    <c:redirect url="http://www.mybeat.techmahindra.com"/>
  </body>
</html>
```

- Above example would redirect request to
<http://www.mybeat.techmahindra.com>

Participants Task

Explore following tags –

- `<c:remove >`
- `<c:import>`
- `<c:forTokens>`
- `<c:param>`

Summary

In this session, we have covered,

- Need for JSP
- JSP Architecture
- JSP Tags
- Implicit Objects
- Action Tags
- useBean Action Tag
- Bean's Scope
- JSP vs Servlets
- Core JSTL



Thank You

Disclaimer

Tech Mahindra Limited, herein referred to as TechM provide a wide array of presentations and reports, with the contributions of various professionals. These presentations and reports are for informational purposes and private circulation only and do not constitute an offer to buy or sell any securities mentioned therein. They do not purport to be a complete description of the markets conditions or developments referred to in the material. While utmost care has been taken in preparing the above, we claim no responsibility for their accuracy. We shall not be liable for any direct or indirect losses arising from the use thereof and the viewers are requested to use the information contained herein at their own risk. These presentations and reports should not be reproduced, re-circulated, published in any media, website or otherwise, in any form or manner, in part or as a whole, without the express consent in writing of TechM or its subsidiaries. Any unauthorized use, disclosure or public dissemination of information contained herein is prohibited. Unless specifically noted, TechM is not responsible for the content of these presentations and/or the opinions of the presenters. Individual situations and local practices and standards may vary, so viewers and others utilizing information contained within a presentation are free to adopt differing standards and approaches as they see fit. You may not repackage or sell the presentation. Products and names mentioned in materials or presentations are the property of their respective owners and the mention of them does not constitute an endorsement by TechM. Information contained in a presentation hosted or promoted by TechM is provided "as is" without warranty of any kind, either expressed or implied, including any warranty of merchantability or fitness for a particular purpose. TechM assumes no liability or responsibility for the contents of a presentation or the opinions expressed by the presenters. All expressions of opinion are subject to change without notice.