Java Server Page (JSP)

- Java Server Pages (JSP)
  - JSP Scripting Elements
  - Implicit objects
  - JSP Tags
  - JSTL

- **Building Web Applications using MVC 2 Architecture**

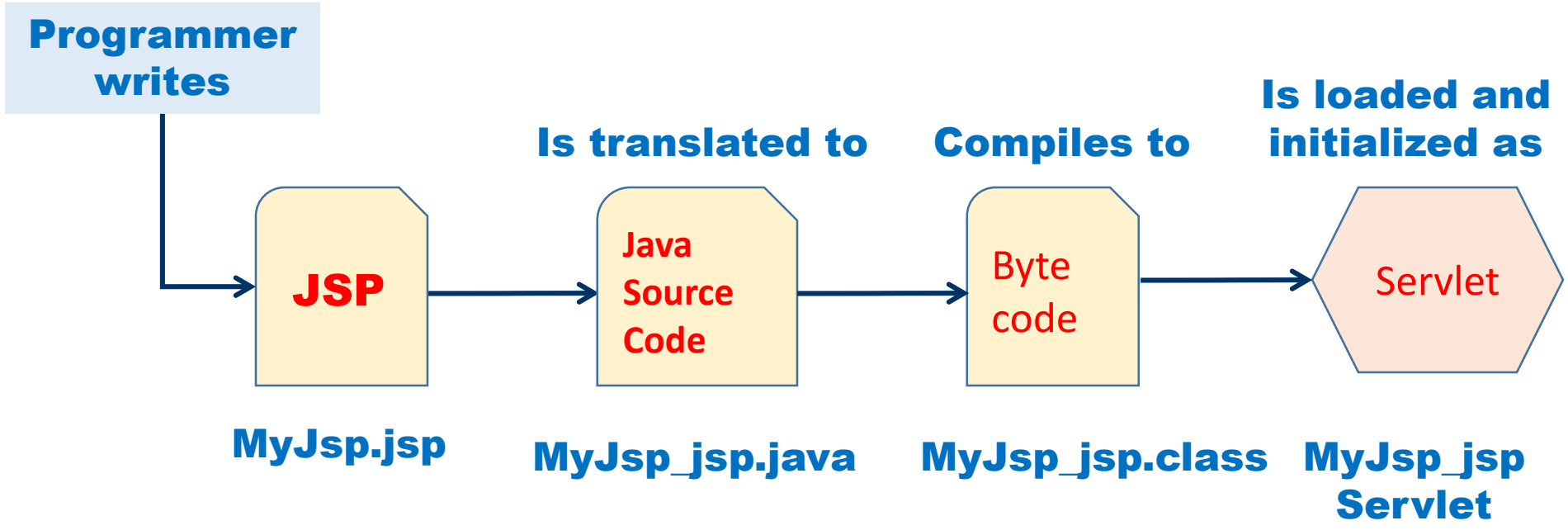
JSP Introduction

- **JavaServer Pages Technology**
- JavaServer Pages (JSP) technology allows you to easily create web content that has both static and dynamic components. JSP technology makes available all the dynamic capabilities of Java Servlet technology but provides a more natural approach to creating static content.
- *The main features of JSP technology are as follows:*
  - A language for developing JSP pages, which are text-based documents that describe how to process a request and construct a response
  - An expression language for accessing server-side objects
  - Mechanisms for defining extensions to the JSP language

- **What Is a JSP Page?**
- A **JSP page** is a text document that contains two types of text: static data, which can be expressed in any text-based format (such as HTML, SVG, WML, and XML), and JSP elements, which construct dynamic content.

JSP Processing

- Internally, JSP gets converted into a Servlet
- When the user requests for a .jsp file for the first time, the JSP Container will create a Servlet that would produce the output that the .jsp file is supposed to produce.
- Starting from the second request, there is no overhead of compilation

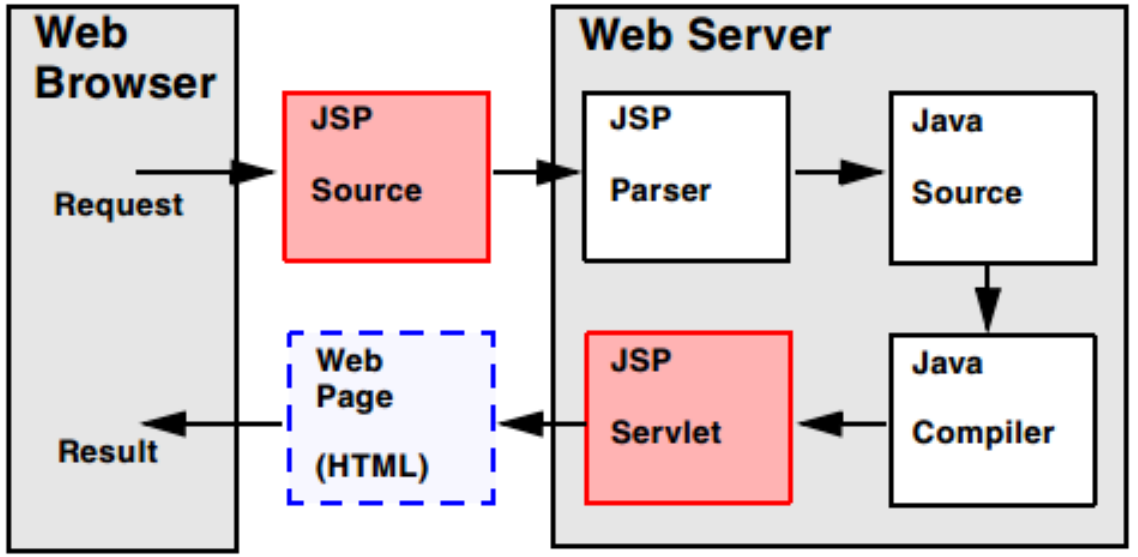


In the end, a JSP is just a Servlet

Servlet Vs JSP

Servlet	JSP
Handles Dynamic data	Handles Dynamic data
Handles business logic	Handles presentation logic
Lifecycle methods init() : can be overridden service() : can be overridden destroy() : can be overridden	*Lifecycle methods* jspInit() : can be overridden _JspService(): cannot be overridden jspDestroy() : can be overridden
Html within java out.println(“<html><body>”); out.println(“Time is “+ new Date()); out.println(“</body></html>”);	**Java within html** <html><body> Time is <% new Date() %> </body></html>
Runs within a Web Container	Runs within a Web Container

JSP Life Cycle



The JSP processing life-cycle on first-time invocation

Following are the steps followed by a JSP Container:

- Phases
  - Translation Phase : Parsing the JSP i.e. ***xxx.jsp -> xxx.java***
  - Compilation Phase : ***xxx.java->xxx.class***
  - Execution Phase *: **xxx.class -> instantiation-> servlet***
    - Initialization (*invokes the **jspInit()** method*)
- For Each request , *the **_jspService()** method* is invoked
- Cleanup *(The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container, **jspDestroy()** method is invoked)*

JSP Tags

Tags in JSP can be categorized into:

- Comments
- Scripting Elements
  - Declarations
  - Expressions
  - Scriptlets
- Directive Elements
- Action Elements

JSP Comments:

JSP comment marks text or statements that the JSP container should ignore.

Following is the syntax of JSP comments:

<%-- This is JSP comment --%>

A **scriptlet** is a fragment of java code which is placed within **<%** and **%>** delimiters and is executed when the user requests for the page.

- Java code placed within **<%** and **%>**

JSP Scripting Elements

Scripting elements are elements in the JSP page that contains java code. JSP contains 3 types of scripting elements as shown below:

1. Declarations

– Format: **<%! code %>**

– Inserted into the body of the servlet class, outside of any existing methods

- Variables declared within <%! and %> will be a data member of the generated Servlet class and all the requests will use the same copy of this variable.
- Methods, if required, should be declared within <%! and %> and they become a method of the generated Servlet.

2. Expressions

– Format: **<%= expression %>**

- Expression is evaluated and inserted into the servlet's output i.e., results in something like ***out.print(expression)*** *i.e.* expression placed in *_jspService* inside *out.print()*

- The expression element **should not end an expression with semicolon.** **Ex.** <%= new java.util.Date() %>

3. Scriptlets

– Format: **<% code %>**

– Inserted into the servlet's _jspService() method.

Scriptlets

- **Format**
  <% Java Code %>

- **Result**

– Code is inserted into servlet's_jspService() method

- **Example**

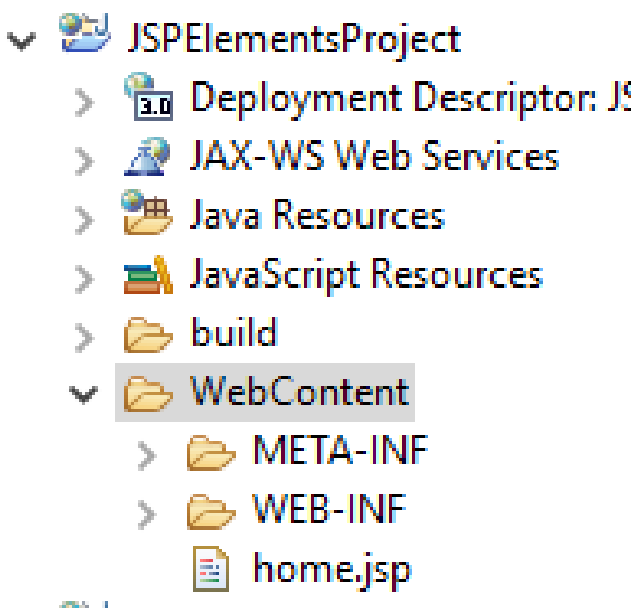
```
<%
    String queryData = request.getQueryString();
    out.println("Attached GET data: " + queryData);
%>
<% response.setContentType("text/plain"); %>
```

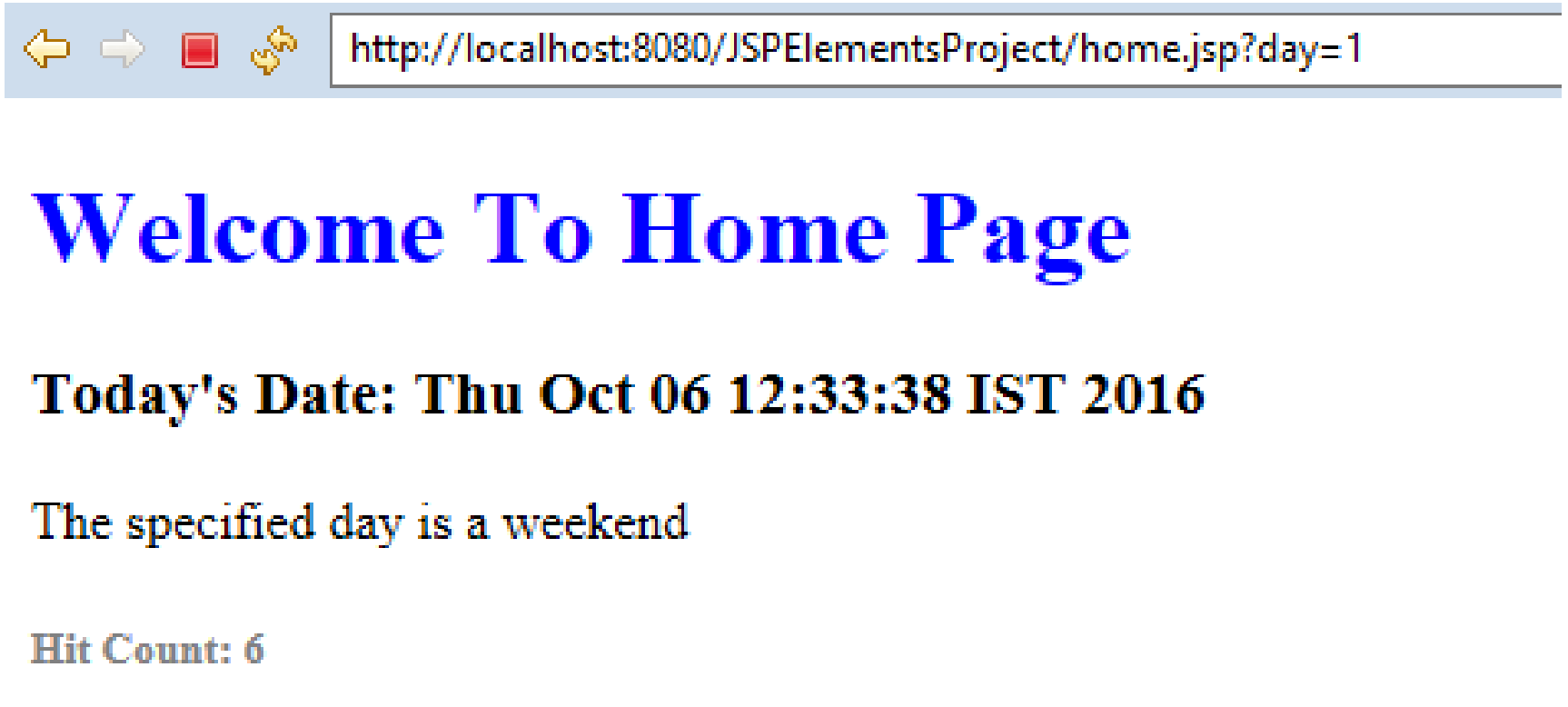
The **if...else** block starts out like an ordinary scriptlet, but the scriptlet is closed at each line with HTML text included between scriptlet tags

```
<%! int day = 3; %>
<html> <head><title>if...else example</title></head>
 <body>
      <% if (day == 1 | day == 7) { %>
           <p> Today is weekend</p>
      <% } else { %>
           <p> Today is not weekend</p>
      <% } %>
</body>
 </html>
```

- A scriptlet contains Java code that is executed every time the JSP is invoked.
- By itself a scriptlet does not generate HTML.
- If a scriptlet wants to generate HTML, it can use **implicit variable** called "out".

Generate the following output using JSP





JSP implicit objects

JSP supports **nine automatically defined variables**, which are also called implicit objects.

Objects	Description
request	This is **HttpServletRequest** object associated with the request. Request objects are passed as parameters to _jspService() method when a client request is made.
response	This is **HttpServletResponse** object associated with the response to the client. Response objects carry the response of a client request after the _jspService() method is executed.
out	This is **PrintWriter** object used to send output to the client.
session	This is **HttpSession** object associated with the request. The session object helps access session data.
application	The application object refers to the entire environment of a web application to which a JSP page belongs. The ServletContext (for sharing data) as obtained via getServletContext().

JSP implicit objects

Objects	Description
config	The config object specifies the configuration of the parameters passed to a JSP page. This is the **ServletConfig** object associated with the page.
pageContext	The pageContext object represents the context of the current JSP page. This encapsulates use of server-specific features like higher performance **JspWriters**.
page	This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
exception	The **Exception** object allows the exception data to be accessed by designated JSP.

JSP Directives

JSP Action Tags

JSP Directives

JSP Directives:

A JSP directive elements provide information to JSP container about the page.

```
<%@ directive attribute="value" %>
```

There are three types of directive tags:

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, to include third-party tag libraries such as JSTL, Spring form tags etc. or custom tag libraries.

JSP Page Directives

The page directive has the following form:

<%@ page *attributes* %>

Ex.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" import="com.cgs.businesstier.Employee"%>
```

Attribute	Purpose
contentType	Defines the character encoding scheme.
errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
isErrorPage	Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
extends	Specifies a super class that the generated servlet must extend
import	Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
isThreadSafe	Defines the threading model for the generated servlet.
language	Defines the programming language used in the JSP page.
session	Specifies whether or not the JSP page participates in HTTP sessions

errorPage & isErrorPage example

sample.jsp

```
<%@ page errorPage="error.jsp" language="java"
contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
……….
<body>
<%!
      String string = null;
%>
The length of the string is <%= string.length() %>
</body>
</html>
```

Note: Run sample.jsp on external browser

error.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"    pageEncoding="ISO-8859-1"  isErrorPage="true"%>
…….
<body>
      <h2>Error Encountered</h2>
      <b><font color="red"><%= exception %></font> has
      encountered. </b>String variable has null.
</body>
</html>
```

JSP directive - include

The include directive can be used to include the contents of some other file in a JSP.

Example:

<%@ include file="../header.html" %>

The contents of the included file will be pasted as a part of the JSP

JSP Action Tags

- JSP actions or action tags are the constructs that follows an XML syntax.

  - The behavior of the servlet engine is controlled by these action. Using JSP actions, a file can be inserted into another page dynamically, reuse a bean component, or forward a user from one page to another page.

    The following are the JSP action tags.

    - inserting other page resources: <jsp:include page="">
    - forwarding the request to another page: <jsp:forward page="">
    - To append parameters while forwarding request object from source page to target page: <jsp:param>
    - creating and locating the java bean instances: <jsp:useBean ….. >
    - setting and retrieving bean properties, in JSP pages: <jsp:getProperty ….> & <jsp:setProperty ..>

Note : Instead of writing Java code in a scriptlet, developers use special JSP action tags, for example to link to a Java bean to set its properties, or get its properties.

include action tag

The include action tag allows a static or dynamic resource such as HTML or JSP pages, specified by a URL, to be included in the current JSP while processing a request.

General syntax of include action Tag:

<**jsp:include page**="{relativeURL | <%= expression %> }” />

- **include** action in JSP is used to include static or dynamic resource in JSP file
- Elements of this action is processed at the time of the JSP page execution.
- Here in case of the static resource, contents are inserted into the calling JSP file.
- In case of dynamic resource, the included resource receives the request object and gets executed. The result is included in the calling JSP file.

Syntax for a dynamic resource:

```
<jsp:include page="{relativeURL | <%= expression %>}"  >
    <jsp:param name="parameterName" value="{parameterValue | <%= expression %>}" />
</jsp:include>
```

forward action tag

The **jsp:forward** tag forwards the request to another resource. The resource can be dynamic or static.

Syntax

```
<jsp:forward page="URL" />

<jsp:forward page="URL" >
      <jsp:param name="ParamName1" value="ParamValue1" />
       <jsp:param name="ParamName2" value="ParamValue2" />
</jsp:forward>
```

This action tag compliments the forward() method of RequestDispatcher object.

Note: One factor that you need to keep in mind when using this tag is *its interaction with output buffering*. When the processing of a page encounters a <jsp:forward/> tag **all output generated so far will be cleared**

Difference Between Include Directive and Include action

include Directive: **<%@ include file= "index.jsp" %>**

- At JSP page translation time, the content of the file given in the include directive is ‘pasted’ as it is, in the place where the JSP include directive is written in the source page. Then the source JSP page is converted into a java servlet class.
- **The original JSP file will not get affected if there is a change in the included JSP file.**

Include Action: **<jsp:include page= "index.jsp" >**

The *jsp:include* action element is like a function call. At runtime, the included file will be ‘executed’ and the result will be included with the source JSP page. *When the included JSP page is called, both the request and response objects are passed as parameters.*

Calling a servlet from a JSP : html form action

```
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JSP Calling servlet through form action tag of html</title>
</head><body>
<H1> Call Servlet from JSP </H1>
<FORM method="POST" action="/JSPProject/CounterServlet">
      <P> <INPUT type="submit" name="CALL_SERVLET" value="Call the Servlet">
</FORM>
</body></html>
```

Calling a servlet from a JSP : jsp:forward

Invoking a Servlet from a JSP directly through the **jsp:include** or **jsp:forward** tags.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Call the servlet through jsp:forward tag</title>
</head>
<body>
<p> Calling Servlet from JSP </p>
<jsp:forward page="/JSPProject/CounterServlet"></jsp:forward>
</body>
```

In reality, It is other way , i.e. A Servlet that acts as a controller invokes JSP

Invoking a JSP from a JSP

To invoke a JSP file from another JSP file, you can perform one of the following:

- Specify the URL of the second JSP file on the FORM ACTION attribute:

  <FORM action=*"JSP_URL"*>

- Specify the URL of the second JSP file in an anchor tag HREF attribute:

  ** reference-text **

- Using **<jsp:forward >** or **<jsp:include>** action tags.

Java EE

- Invoking JSP from a Servlet
- Java Bean
- JSP Action Tags: useBean, getProperty & setProperty
- Expression Language (EL)
- JSP Standard Tag Library (JSTL)

Invoking a JSP Page from a Servlet

- You can invoke a JSP page from a servlet through functionality of the standard javax.servlet.RequestDispatcher interface.

Complete the following steps in your code to use this mechanism:

- Get a servlet context instance from the servlet instance:

```
ServletContext sc = this.getServletContext();
```

- Get a request dispatcher from the servlet context instance, specifying the absolute path of the target JSP page as input to the getRequestDispatcher() method:

```
RequestDispatcher rd = sc.getRequestDispatcher("/jsp/mypage.jsp");
```

- Invoke the include() or forward() methods of the request dispatcher, specifying the HTTP request and response objects as arguments.

```
rd.include(request, response);    or   rd.forward(request, response);
```

- The functionality of these methods is similar to that of *jsp:include* and *jsp:forward* actions.

Note : MVC Architecture is implemented using the above technique

About Java beans

A JavaBean is a Plain Old Java object (POJO) that is Serializable, has a 0-argument i.e. *no-arg* constructor, and allows access to its properties using getter and setter methods.

Java Beans are Java classes that obey the following conventions:

– Must have a zero-argument (empty) constructor

• You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors

– Should have no public instance variables (fields)

– Persistent values should be accessed through methods called getXxx

- If class has method *getTitle()* that returns a String, class is said to have a String property named *title*
- Boolean properties may use *isXxx* instead of *getXxx*

Note : It is the name of the method, not instance variable that matters!

Using Beans : Basic tasks

jsp:useBean

– In the simplest case, this element instantiates a java bean, or to locate an existing bean instance, and assign it to id.

It is used as:

<jsp:useBean id="beanName" class="package.Class " scope= " request" />

jsp:setProperty

-This element modifies a bean property (i.e., calls a setXXX method).

It is used as:

<jsp:setProperty name="beanName"
property="propertyName"
value="propertyValue" />

jsp:getProperty

– This element reads and outputs the value of a property.

It is used as:

<jsp:getProperty name="beanName"
property="propertyName" />

Example:

<jsp:useBean id="book1" class= " com.lnt.businesstier.Book" />

is equivalent to the scriptlet

<% org.asr.beans.Book book1 = new com.lnt.businesstier.Book.Book(); %>

– But **jsp:useBean** has two additional advantages:

- **It is easier to derive object values from request parameters**
- **It is easier to share objects among pages or servlets**

Attributes of <jsp:usebean> tag

Attributes of the <jsp:useBean> Tag:

- Id: The id attribute of the jsp:useBean action tag represents the variable name assigned to the id attribute of the jsp:useBean.

- scope: The scope attribute represents the scope in which the bean instance has to be located or created. scopes can be page, request, session or application.

  - **page (default) :** It means that we can use the Bean within the JSP page.
  - **request**: We can use the bean from any JSP page processing the same request.
  - **session**: We can use the bean from any JSP page in the same session as the jsp page that created the bean.
  - **application**: We can use the bean from any page in the same application as the jsp page that created the bean.

- class: It takes the *qualified class name (packagename.classname)* to create a bean instance if the bean instance is not found in the given scope.

Setting Java Bean properties from request parameters

- To set the Java Bean properties from request parameters, we can use param attribute as shown below:

```
<jsp:useBean id="myBeanAttribute" class="com.lnt.businesstier.MyBean" scope="session">
        <jsp:setProperty name="myBeanAttribute" property="id" param="empID" />
</jsp:useBean>
```

- If property and param attribute values are same, we can skip the param attribute. For example if request parameter name is also *id* then we can simply write:

```
<jsp:useBean id="myBeanAttribute" class="com.lnt.businesstier.MyBean" scope="session">
        <jsp:setProperty name="myBeanAttribute" property="id" />
</jsp:useBean>
```

- If all the request parameter names matches with the java bean properties, then we can simply set bean properties as shown below.

```
<jsp:useBean id="myBeanAttribute" class=" com.lnt.businesstier.MyBean " scope="session">
        <jsp:setProperty name="myBeanAttribute" property="*" />
</jsp:useBean>
```

Using Expression Language (EL) in a JSP Page

An EL expression is delimited in a JSP page using the notation:

${el-expression-goes-here}

When the JSP is executed, the value of the expression is determined and the delimiters, along with the enclosed expression, is replaced with the text evaluation of the result.

For example, the EL expression **${ 3 + 4 }** will be replaced with the text 7 in the response output or attribute value.

So, When the JSP compiler sees the ${} form in an attribute, it generates code to evaluate the expression and substitutes the value of expression.

Accessing Java Bean properties

While strings and numbers are useful for representing scalar values, more frequently, complex objects in the form of JavaBeans are what we have to deal with, so the EL is specifically designed to make it easy to access the properties of JavaBeans.

For example, If an instance of this class were to be created as a scoped variable named person, we can use the EL's property operator to reference the property values.

This operator has two forms:

1. The simplest and most commonly-used form is the dot (period) character.

For example, to reference the bean's title property:

```
${person.title}
${person.firstName}
```

2. The other form of the property operator consists of square bracket characters([]) which contain an expression whose string evaluation is taken as the property to be referenced.

```
${person['title']}
```

This expression is identical to ${person.title}

Building Web Applications using MVC 2 Architecture

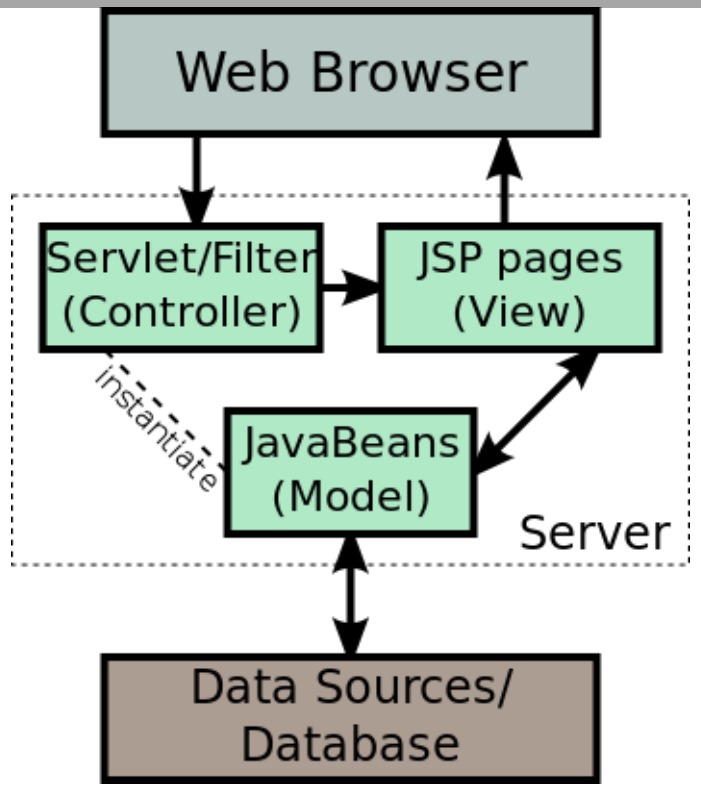
Building Web Applications

Simple Application

Complex Application

- Scripting elements calling servlet code directly
- Scripting elements calling servlet code indirectly (by means of utility classes)
- Beans
- Servlet/JSP combo (MVC 2 Architecture)
- MVC with JSP expression language (JSTL)
- Custom tags
- MVC with beans and a framework like Struts or JSF or Spring

MVC Model 2 Architecture



MVC based frameworks are: **Struts 2 , JSF , Spring etc.**

In JSP Model 2 architecture , **JSP** is used for creating the **view** for the application.

A centralized Servlet is used to handle all the request for the application. *The* **Servlet** *works as the* ***controller*** *for the application*. It then uses the **Java beans** for processing the business logic and getting the ***data (Model)*** from the database.

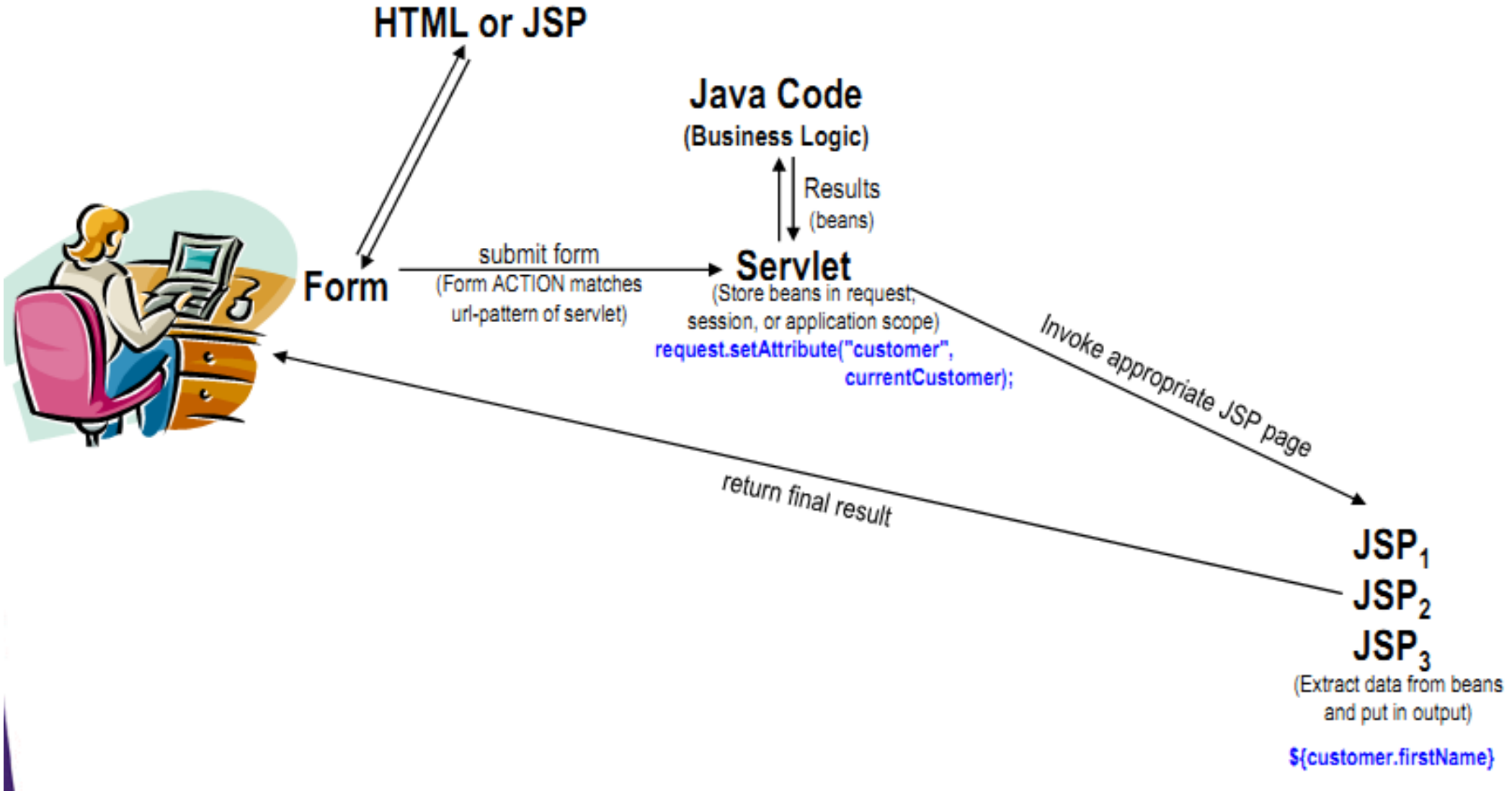
Finally it uses the JSP to render the view which is displayed to the user.

Why MVC ?

Combination (MVC architecture). Needed when:

- A single request will result in multiple substantially different-looking results.
- You have a large development team with different team members doing the Web development and the business logic.
- You perform complicated data processing, but have a relatively fixed layout.

MVC Flow of Control



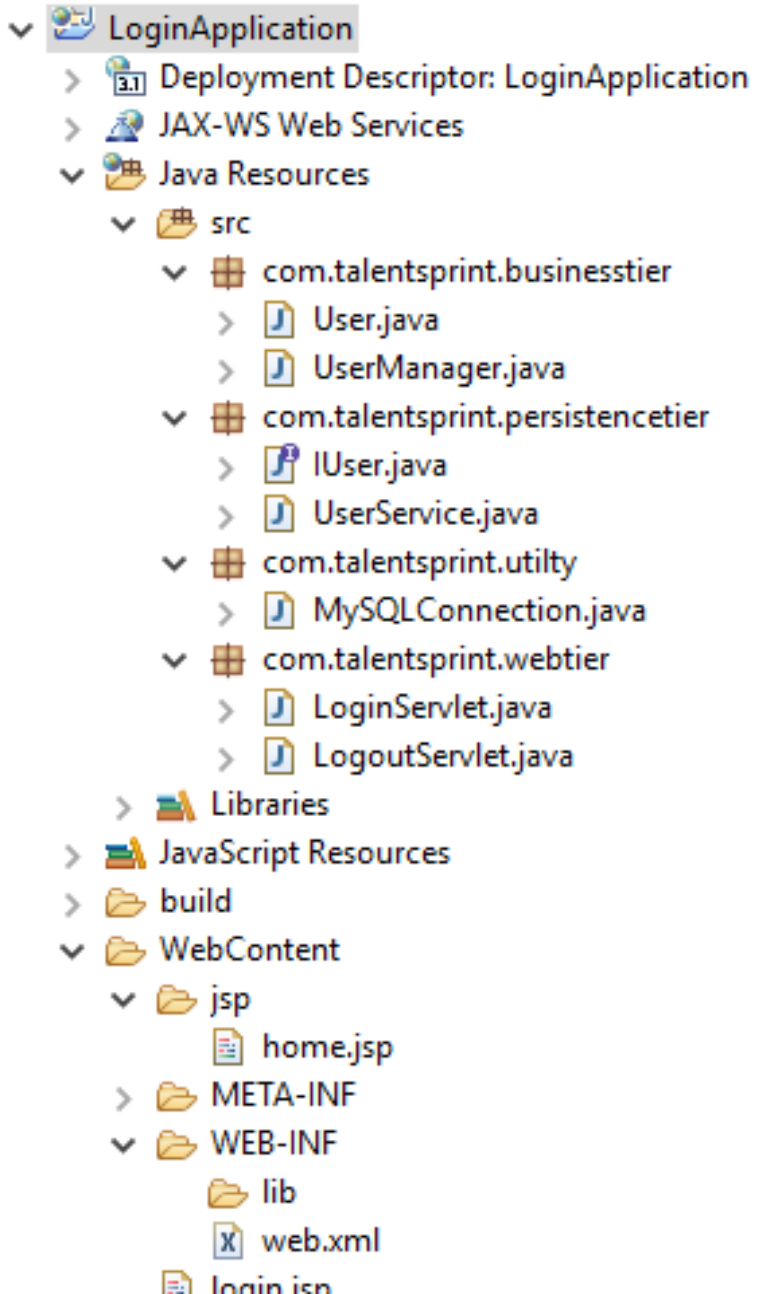
Implementing MVC Architecture with Servlet and JSP API

1. **Define beans to represent result data**
   - Ordinary Java classes with at least one get*Blah* method
2. **Use a servlet to handle requests**
   - Servlet reads request parameters, checks for missing and malformed data, calls business logic, etc.
3. **Obtain bean instances**
   - The servlet invokes business logic (application-specific code) or data-access code to obtain the results.
4. **Store the bean in the request, session, or servlet context**
   - The servlet calls setAttribute on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.

Implementing MVC Architecture with Servlet and JSP API

5. **Forward the request to a JSP page.**
   - The servlet determines which JSP page is appropriate to the situation and uses the forward method of RequestDispatcher to transfer control to that page.
6. **Extract the data from the beans.**
   - JSP 1.2: the JSP page accesses beans with jsp:useBean and a scope matching the location of step 4. The page then uses jsp:getProperty to output the bean properties.
   - JSP 2.0: the JSP page uses ${nameFromServlet.property} to output bean properties
   - Either way, the JSP page does not create or modify the bean; it merely extracts and displays data that the servlet created.

Login Application




```
CREATE TABLE university.users (
userid INT NOT NULL,
username VARCHAR(30) NULL,
password VARCHAR(20) NULL,
status VARCHAR(20) NULL,
PRIMARY KEY (userid)
);
```

http://bootsnipp.com/

Login Application

- **login.jsp accepts credentials and passes it on to LoginServlet.**
- **LoginServlet_i**nvokes ***verifyCredentials()*** method of **UserManager** class to validate ***username*** and ***password*** *entered by the user.*
- ***The verifyCredentials()*** method of **UserManager** class invokes ***verifyCredentials()*** method of **UserService class** which in turn connects to database and verifies whether the credentials are registered in the ***users*** table, returns true/false.
- If credentials are valid , **create session object** and render home page(***home.jsp***) to the browser else render ***login.jsp.***
- ***home.jsp*** provide a hyperlink to logout(invokes **LogoutServlet)**
- **LogoutServlet** invalidates the session and send html content **" You have logged out of the application. Than Q" and** also provides an hyperlink to login.

Login Application

Dynamic Web Project: **LoginApplication**

Package: **com.cgs.webtier**

Servlets: LoginServlet , LogoutServlet

Package: **com.cgs.businesstier**

class: **UserManager**

instance method:

public Boolean verifyCredentials(User user)

Note: Invokes *verifyCredentials()* method of UserService class

Package: **com. cgs.persistencetier**

class: UserService

instance method:

public Boolean verifyCredentials(User user) : This method connects to database and verifies whether the credentials are registered in the ***users*** table, returns true/false

Package: **com. cgs.utility**

class: **MySQLConnection**

Create a folder, *jsp* under WebContent : Files in this folder are home.jsp , login.jsp

login.jsp

```
<h1>Login Screen</h1>
   <form action="LoginServlet" method="POST">
                Enter username:<input type="text" name="username“ size=“20” >
                Enter password: <input type="password" name="password“ size=“20” >
                <input type="submit" value="Submit“>
      </form>
```

BigPartyWebApp.rar

LoginServlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                String username= request.getParameter("username");
                String password=request.getParameter("password");
                UserManager manager=new UserManager();
                Boolean flag=manager.verifyUsernameAndPassword(username, password);
                if(flag==true){
                        HttpSession session=request.getSession(true);
                        session.setAttribute("username", username);
                        String uri= "views/main_menu.jsp"
                         request.getRequestDispatcher(uri).forward(request, response);
                        //out.println("<html><body"+"<h1>Hi! Welcome "+username+"</h1></body></html>");
                }else{
                        out.println("<html><body"+"<h1>"+"Invalid Credentials"+"</h1>"+"</body></html>");
                        out.println("<a href=\"login.jsp\">"+"Login"+"</a>");
                }

        }
```

LogoutServlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        HttpSession session=request.getSession(false);
        PrintWriter out=response.getWriter();
        if(session!=null){
                session.removeAttribute("username");
                session.invalidate();
                out.println("<a href=\"login.jsp\">"+"Login"+"</a>");
        }else{
                out.println("Please Login<br>");
                out.println("<a href=\"login.jsp\">"+"Login"+"</a>");
        }
}
```

Prevent the browsers from caching

The root of the problem is the Back button that exists on most modern browsers. When the Back button is clicked, the browser by default does not request a page from the Web server. Instead, the browser simply reloads the page from its cache

To leverage the HTTP headers' cache directives, the following code snippet should be placed at the top of all protected JSP pages.

```
//...
response.setHeader("Cache-Control","no-cache"); //Forces caches to obtain a new copy of the page from the origin server
response.setHeader("Cache-Control","no-store"); //Directs caches not to store the page under any circumstance
response.setDateHeader("Expires", 0); //Causes the proxy cache to see the page as "stale"
response.setHeader("Pragma","no-cache"); //HTTP 1.0 backward compatibility

String userName = (String) session.getAttribute("User");
if (null == userName) {
        request.setAttribute("Error", "Session has ended. Please login.");
        RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
        rd.forward(request, response);
}
 //...
```

JSP Standard Tag Library (JSTL)

JSTL allows us to write JSP pages using tags, rather than the scriptlet code

1. JSTL provides standard tag libraries into four groups
    1. core
    2. XML
    3. Internationalization/format
    4. SQL

The primary design goal for JSTL and the EL (Expression Language) is to simplify webpage development and implementation.

Note: Apache Tomcat 8 supports Java Servlet 3.1, Java Server Pages 2.3 and Expression Language 3.0 specifications.

JSTL is by default not available in Tomcat. So, download ***jstl-1.2.jar*** file and place it into your project's classpath.

Java Standard Tag Library (JSTL) Groups

Although JSTL is officially named the Java Server Pages Standard Tag Library, it divides its tags into four groups and makes them available as separate tag libraries.

The following table lists the different libraries, along with their URIs and prefixes:

JSTL tag library	prefix	URI	Example Tag
Core Library	c	**<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**	<c:forEach>
XML processing library	x	**<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>**	<x:forEach>
Internationalization and formatting	fmt	**<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>**	<fmt:formatDate>
Database Access (SQL)	sql	**<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>**	<sql:query>

The core group of tags are the most frequently used JSTL tags.

Note: Earlier JSTL versions used URI: http://java.sun.com/jstl/core which is now revised to above specified URI.

JSTL Core Tags

Tags	Description
c:out	It display the result of an expression, similar to the way <%=...%> tag work.
c:import	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
c:set	It sets the result of an expression under evaluation in a 'scope' variable.
c:remove	It is used for removing the specified scoped variable from a particular scope.
c:catch	It is used for Catches any Throwable exceptions that occurs in the body.
c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
c:choose,c:when, c:otherwise	It is the simple conditional tag that includes its body content if the evaluated condition is true.
c:forEach	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
c:forTokens	It iterates over tokens which is separated by the supplied delimeters.
c:param	It adds a parameter in a containing 'import' tag's URL.
c:redirect	It redirects the browser to a new URL and supports the context-relative URLs.
c:url	It creates a URL with optional query parameters.

JSTL Core Tag Examples

jstl_demo1.jsp

```
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Your income is : <c:out value="${income}"/></p>
<c:choose>
    <c:when test="${income <= 1000}">
       Income is not good.
    </c:when>
    <c:when test="${income > 10000}">
        Income is very good.
    </c:when>
    <c:otherwise>
       Income is undetermined...
    </c:otherwise>
</c:choose>
```

Note: jstl-1.2.jar file to be placed in your project's classpath

```
<%-- <c:url> is equivalent to response.encodeURL() method --%>
<c:url value="/index1.jsp" var="completeURL">
 <c:param name="trackingId" value="786"/>
 <c:param name="user" value="Ravi"/>
</c:url>
${completeURL}
```

jstl_demo2.jsp

```
<%-- <c:redirect> is equivalent to response.sendRedirect() method --%>
 <c:set var="id" value="0" scope="request"/>
 <%-- JSTL does not have elseif tag --%>
  <c:if test="${id<1}">
     <c:redirect url="http://google.com"/>
  </c:if>
  <c:if test="${id>1}">
     <c:redirect url="http://microsoft.com"/>
  </c:if>
```

<c:forEach> Example

jstl_demo3.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" import="java.util.Map,java.util.HashMap"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
…….
<body>
<%
    Map<String, String> countryCapitalMap = new HashMap<>();
    countryCapitalMap.put("United States", "Washington DC");
    countryCapitalMap.put("India", "Delhi");countryCapitalMap.put("Germany", "Berlin");
    countryCapitalMap.put("France", "Paris");countryCapitalMap.put("Italy", "Rome");

    request.setAttribute("capital", countryCapitalMap);
%>
<%-- JSP Code --%>
<table border="1">
    <tr><th bgcolor="green">COUNTRY</th><th bgcolor="green">CAPITAL</th></tr>
    <c:forEach var="c" items="${capital}">
        <tr><td>${c.key}</td><td>${c.value}</td></tr>
    </c:forEach>
</table>
</body></html>
```

<c:forEach> Example

servlet

```
……..
request.setAttribute("customerList", customerList);
uri="views/show_all_customers.jsp";
}else{
throw new Exception("No Customers in the database");
}
}
request.getRequestDispatcher(uri).forward(request, response);

…….
```

```
<c:forEach var="c" items="${customerList}">
    <tr>
      <td class="hidden-xs">${c.customerId}</td>
       <td>${c.customerName }</td>
      <td>${c.customizedBirthdate}</td>
      <td>${c.address}</td>
      <td>${c.email}</td>
      <td>${c.mobile}</td>
    </tr>
</c:forEach>
```

BigPartyWebApp.rar

<c:forEach> Example

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
……..
<head>
<link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
</head>
<body>
<jsp:useBean id="customerList" type="java.util.List<Customer>" scope="request"/>
<div class="container">
  <h2>Customer Details</h2>
  <table class="table">
    <thead>
      <tr> <th>Customer ID</th><th>Customer Name</th><th>Birthdate</th><th>Mobile</th>
         <th>Address</th> <th>Email</th> <th>Password</th> <th>Edit</th><th>Delete</th>      </tr>
    </thead>
    <tbody>
    <c:forEach var="c" items="${customerList}">
      <tr class="success">
        <td>${c.customerId}</td><td>${c.customerName}</td><td>${c.customDate}</td>
        <td>${c.mobile}</td><td>${c.address}</td><td>${c.email}</td><td>${c.password}</td>
        <td><a href="updatecustomer?custid=${c.customerId}">edit</a></td>
        <td><a href="deletecustomer?custid=${c.customerId}">delete</a></td>
      </tr>
    </c:forEach>
    </tbody>
  </table>
</div>
</body></html>
```

Thank You!