

# Chapter – 01

## The Anatomy of a Java Server Page

### ❖ The JSP life cycle

#### Translation:

- After checking the JSP syntax the JSP engine will translate the JSP page into its page implementation class.
- `jspInit()` method is automatically generated during the translation phase.
- This method is used for initializing the implementation servlet.

#### Initialization:

- The class file is loaded and creates an instance of the servlet.

#### Execution:

- `_jspService()` method provides all the functionality for handling a request and returning response to the client.

#### Finalization:

- `jspDestroy()` method called by the servlet container when the page implementation servlet is about to be destroy.

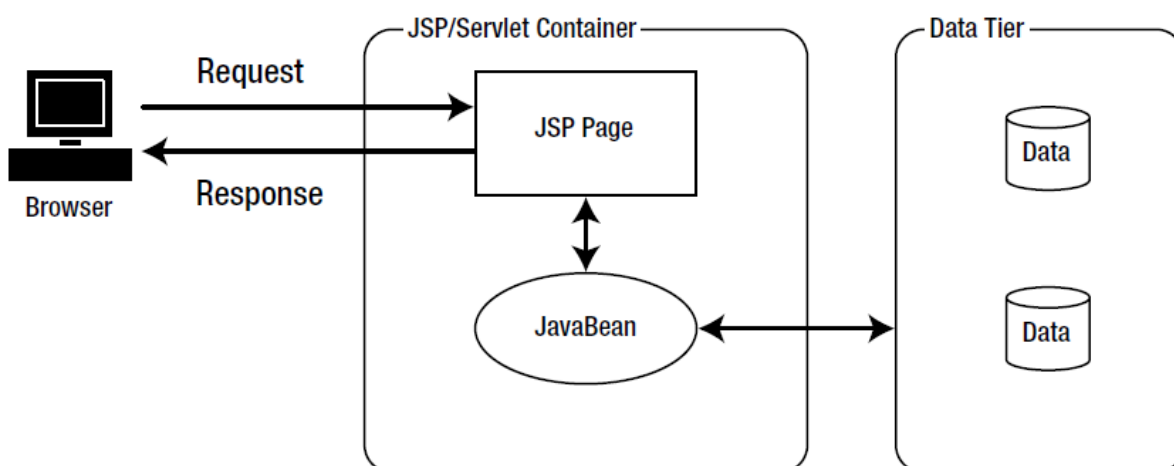
### ❖ Benefits of JSP

- Reusability
- Readability
- Maintainability

### ❖ JSP application Architecture:

#### Model 1 Architecture

- JSP page is expected to deal with its request entirely by itself. Thereby generating a response and sending it back to the client and for this reason it is often known as page-centric.

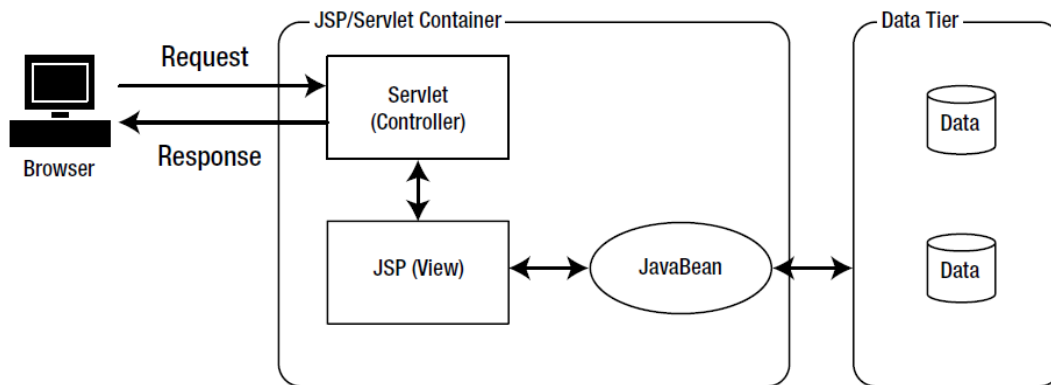


### ❖ Problem of Model -1 Architecture:

- Maintainability problems
- Reusability problems

✚ Security problems

## ❖ **Model 2 Architecture(MVC):**



## ❖ **Advantages:**

✚ **Maintainability:**

By separating application logic from its presentation logic by using MVC components it is far easier to develop cleaner code that resulting ultimately in a more flexible and maintainable

✚ **Security:**

✚ **Extensibility:** One of the best points about the mdel-2 architecture is that al the processing logic is centralized.

## ❖ **Tomcat 5.5**

✚ The bin directory contains all the necessary scripts required to start the container.

✚ The cont directory contains the entire XML based configuration file used by tomcat.

## ❖ **Java Server Page(JSP)**

✚ The programmatic logic maybe classified in to following JSP elements

- Scripting elements
- Directives
- Action element

## ❖ **Template Text:**

✚ Any non JSP code located inside a JSP page is known as template text

✚ The most common form of template text is markup such as HTML or XML

## ❖ **Scripting Elements:**

Scripting elements are used within a JSP page to manipulate objects and perform computations that enable the generation of dynamic content.

✚ They classified into the following

- Comments
- Declaration
- Scriptlets
- Expressions
- Expression language Expressions

✚ **Comments:** JSP comments is

<% -- this is a JSP comment --%>

HTML comment


<!-- this is HTML comment -->

#### **Declaration:**

<%! Date now = new Date(); %>

#### **Scriptlets:**

<% %>

 **Expression:** Expression is similar to scriptlets but as their name suggests. They evaluate tearegular Java expression and return a result. This result must be a string. Otherwise an exception will be raised during the translation phase as at runtime

<%= expression %>


#### **Expression language Expression**


JSP 2.0 introduce an EL that is based on EDMA script and XML path language(X-path)

`${ object.property }`


### ❖ JSP implicit objects:

Object name	interface
<ul style="list-style-type: none"><li>➤ Request</li><li>➤ Response</li><li>➤ Out</li><li>➤ Session</li><li>➤ Config</li><li>➤ Application</li><li>➤ Page</li><li>➤ PageContext</li><li>➤ Exception</li></ul>	<ul style="list-style-type: none"><li>➤ javax.servlet.http.HttpServletRequest</li><li>➤ javax.servlet.http.HttpServletResponse</li><li>➤ javax.servlet.jsp.jspWriter</li><li>➤ javax.srvlt.http.HttpSession</li><li>➤ javax.srvlt.ServletContext</li></ul> <p>&lt;%@ page isErrorpPage= “true” %&gt;</p>

 The exception objection instance of java.lang. Throwable represents a runtime error that occur during the request process.


 A pageContext instance provides the JSP developer with acces to all the available JSP scopes and to several useful page attributes, such as the current request and response the servletContext. HttpSession and servletConfig

### ❖ JSP Directives

 Directives are used for passing important information to the JSP engine

There are three types of JSP Directives

- Page Directives
- Include Directives
- Taglib Directives

 The page directives

<%@ page import= “java.util.Date” %>

<% @ page isThreadSafe = “false” %>

#### Attribute for the page directive:

Language	isThreadSafe
Extends	info
import	isErrorpage
session	errorPage
buffer outFlush	contentType
isELIgnored	pageEncoding

## Deferred Syntax Allowed as Literal Time Directive Whitespaces

### **They include Directive:**

`<%@ include files = "relative URL"%>`

`<%@ include file = "/copyright.html"%>`

### **The taglib Directive:**

`<%@ taglib uri = "/taglibrary URI" %>`

### **Attributes for the taglib Directive:**

- Uri
- Tagdir
- Prefix

## ❖ **Action Elements**


There are three types of Action Elements

 Standard action


 Custom action


 JSTL action

## ❖ **The standard actions**

 `<jsp:include>`

Provides simple mechanism for including the content of separate web content of a separate web component into a declaring JSP page at translation time

 `<jsp:include page = "relative URL" flush = "true"/>`

 `<jsp:useBean>`


`<jsp:useBean id="name" scope="session">`

 `<jsp:getProperty>`

Used to retrieve or access the existing properties of a javaBean interface

 `<jsp:setProperty>`

Used to set the value of an attribute inside a javaBean

 `<jsp:forward>`

Use to forward the current request to another resource

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

## Chapter 02

### Servlets and Deployment

## ❖ **What is a Servlet?**


A servlet is a server side component that is capable of dynamically processing requests and constructing responses in a protocol independent manner.


## ❖ **Why use Servlet?**

Although JSP page is far easier to create than servlets it is the best use in situations where a great deal of programmatic control is required such as decision making database quering or accessing offer enterprise resources


If we attempt to perform these types of operations within a JSP page, the following problem to be faced

- Maintainability
- Reusability

 The javax.servlet package provides the contract between the servlet or web application and the web container.

 The javax.servlet.Servlet interface defines the core structure of all sevllets.

## ❖ **The javax.servlet Interface**

 The javax.servlet package is composed of 14 interface

✚ The web container implements these 7 interfaces

- ServletContext
- ServletConfig
- ServletResponse
- ServletRequest
- RequestDispatcher
- FilterChain
- FilterConfig

❖ The web application developer implements the rest 7 interfaces to provide the application functionality

- Servlet
- ServletContextListener
- ServletContextAttributeListener
- ServletRequestAttributeListener
- ServletRequestListener
- SingleThreadModel
- Filter

✚ The Servlet interface is key in developing servlets. This interface defines the life cycle methods of a basic servlet

- Initialization
- Service
- Destruction

✚ The servletConfig interface use to pass initialization information to a servlet via the getServletContext() method.

✚ The ServletContextListener interface is a life cycle interface that programmer can implement to list for changes to the state of the servletContext object. This allows the developer to perform application start and shutdown type functionality.

✚ The ServletContextAttributeListener interface can perform similar functionality but the events that they are notified about relate to the modification.

✚ The RequestDispatcher interface manages client requests by directing them to the appropriate resources on the server. Developer can use this interface to redirect the application to different pages and servlet.

✚ The ServletRequest interface encapsulates all the interface that is transmitted to a servlet through its service() method during a single client request

✚ The getParameter() method will return the parameter value with the given name or null if the parameter does not exist.

✚ The getParameterNames() method is used to retrieve a java.util.Enumeration of string objects containing the names of all parameters found in the request.

✚ The getParameterMap() method returns a java.util.Map containing all the parameters found in the request.

❖ **The javax.servlet classes**

Classes within the javax.servlet package

Class name	Use
Abstract class: GenericServlet	Used to develop protocol independent servlets and requires only that subclasses implement its service () method.
Event class: ServletContextEvent Servlet ContextAttributeEvent	Used for nbotification about changes to the ServletContext object and itsattributes
Abstract class: ServletInputStream ServletOutputStream	Provides the ability to read and write binary data from and to the client java.io.InputStream.read() Java.io.OutputStream.write()
Event class: ServletRequestEvent ServletRequestAttributeEvent	Used for notification about changes to the servletRequwst object and its attributes
Wrapper class ServletRequestWrapper ServletResponseWrapper	Provide useful implementation of the servletRequest and SevletResponse interfaces

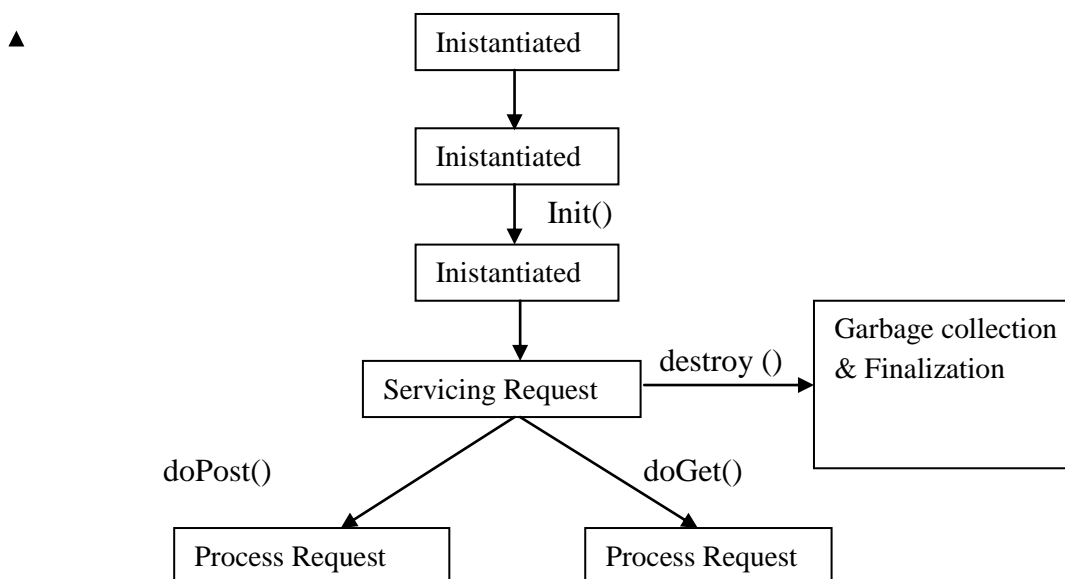
### ❖ The Life Cycle of a Servlet

The javax.servlet.Servlet interface defines methods that are known as Lifecycle methods:

- Init()
- Sevice()
- Destroy()

🎨 The life cycle methods are each called at separate times during the life span of a servlet. These methods are called in the folloeing order

- When the Servlet is constructed it is initialized with the init() method.
- Any requests from clients are handled initially by the service () method before deleting to the doXXX() methods in the case of an HttpSdervlet. The service() method is responsible for processing the response
- When the servlet needs to be removed from service it destroyed with the destroy() method then garbage collection and finalized



**Fig: The Servlet lifecycle**

## ❖ HTTP Servlets

- ✚ The main class is the javax.Servlet.http package is the HttpServlet abstract class
- ✚ This class extends from the javax.servlet.GenericServlet class
- ✚ The doGet() method is intended to retrieve an entity from the server as reference by a request url
- ✚ The doHead() method is simply a get request that is intended to return only the Http header information
- ✚ The doPost() method is intended to allow –posting of information (forms and so on ) to the server
- ✚ The doPut() method is used to upload a file to a server in a manner similar to the FTP.
- ✚ The doOption() and doTrace() method allow us to override the behavior of HTTP

## ❖ HTTP Header Names

Date	Http header
Accept	Http header stores a wide range of
Accept Encoding	information about the user and the
Connection	request and transmitted between user
Content length	web servers during each request.
Cookie	
Host	
Referrer user	
User Agent	

## ❖ Deployment descriptor:

The deployment descriptor describes the web application to the container. The deployment descriptor file is perhaps the single most important item of web application

- ✚ For deployment descriptor to be valid for web applications using the servlet 2.5 specification several things must be true.
  - ✚ The file must conform to the web application XML schema
  - ✚ Must be well formed XML file
  - ✚ Must be named web.xml
  - ✚ Must reside at the top level of the WEB\_INF directory of web application
  - ✚ The deployment descriptor conveys the elements of and configuration information of a web application to developers, assemblers and deployers
  - ✚ Servlet api.jar file located in the common \lib directory used to set the class path.
- ❖ There are two ways we can apply to deploy a web application into Tomcat:
- ✚ Copy the application files and directories directly in tomcat's webapps directory.
  - ✚ Create a distributable web Archive(WAR) file
- ❖ The setContentType() method is used to set the (MIME,RFC2045 and 2046) type of the response
- ❖ The setContentType() method must be called before calling the getWriter() method.
- ❖ Send Redirect() method provides by the
- ✚ javax.servlet.http.HttpServlet interface
  - ✚ This method used to switch the application from one URL to another
  - ✚ <Servlet> tag
  - ✚ Contains several child tags that give information about the declaration of the servlet.
  - ✚ <error-page> tag contains two child element
    - <exception-type>
    - <location>
  - ✚ <jsp-config> used to
    - Enable or disable EL evaluation
    - Enable or disable scripting elements
    - Indicate page encoding information
    - Automatically include preludes and codas

## Chapter-3

### The Java Server Pages Expression language (JSPEL)

#### ❖ JSP expression language:

JSP expression language is an intentionally simple language that is to a large extent independent from jsp

There are many problems associated with using java code in the form of scriptlet in JSP pages

✚ It is very common for non-java programmers to create the user interface for a system

✚ Maintainability problem

For all these reasons the JSP 2.0 Specification introduce or expression language (EL) that can do pretty much everything that scriptlets can do.

EL is simpler to understand and very similar to java Script

#### ❖ Basic Syntax:

EL can be used in anywhere of the code

In JSP page `${expr}`

Deferred expressions `# {expr}`

#### ❖ Use

```
<jsp:UseBean id= "bean" class= "myBean" />
```

```
${bean.name}
```

#### ❖ Literal and their values

Literal	Valid values
Boolean	True or false
Integer	-10 <sup>2</sup> 0 21 21234
Floating Point	-1.09 1.0E 10 1 -10.0 0.1
String	"A string" or 'a string'
Null	null

#### ❖ ArithmeticException `#{2/0}`

#### ❖ Using the Expression language

#### ❖ Within attribute values for jsp standard and custom tags

✚ Using EL : `<myTaglibrary:myTag counter = "<%=1+1%>" />`

```
<myTagLibrary: myTap counter = "${1=1}" />
```

✚ Using the EL writer JSP Template Text

```
<table>
```

```
<tr>
```

```
<td>Hello${param['name']}</td>
```

```
</tr>
```

```
<tr>
```

```
<form action = "templetText.jsp" method = "post">
```

```
<td><input type = text name = "name"></td>
```

```
<td><input type = "submit"></td>
```

```
</form>
```

```
</tr>
```

```
</table>
```



## ❖ Reserved Words

and	true	lt	eq	null	if
or	false	gt	ne	div	instanceof
not			ge	mod	empty

## ❖ Disabling the EL

We can disable EL evaluation in two ways

- ✚ Individually on each page by using the page directive  
`<%@ page isELIgnored="true"%>`
- ✚ Within the web.xml file by using a JSP configuration element  
`<jsp-property-group>`  
`<uri-pattern>*.jsp</uri-pattern>`  
`<el-ignored>true </el-ignored>`  
`</jsp-property-group>`
- ✚ Arithmetic Evaluation using EL  
 The precedence is as follows
  - ()
  - -(unary) `${10-5}`
  - \*/ div mod % `${10/3}`
  - +-(binary)
- ✚ Comparisons in the EL:
  - == or eq
  - != or ne
  - < or lt
  - > or gt
  - <= or le
  - >= or ge
- ✚ Logical operation in the EL
  - && or and
  - || or or
  - ! or not

## ❖ Expression Language Implicit object

- Application Scope
- Cookie
- Header
- HeaderValues
- initParam
- pageContext
- pageScope
- param
- paramValues
- requestScope
- sessionScope

## ❖ Expression language functions

- ✚ An EL function is mapped to a static method of a Java class. This mapping is specified within a tag library descriptor (TLD)
- ✚ Functions must always have a namespace  
`<%@ taglib uri="/WEB-INF/taglib.tld" prefix="myFunctions"%>`
- ✚ A TLD is an XML file that declares a tag library

- ✚ Each function is ever a name and a specific method in a java class that will implement the function

- Method must be public static on a public class
- Function must be unique if not translation time error will occur

#### ❖ **EL function Vs custom Tags**

Functions are much simpler then tag to writer

EL function	Tag
	Provides iterative behavior over a body
Provides to reuse existing java code in a web content	

#### ❖ **Scope**

Scope is the idea that an object belongs to a certain part of an application the five main scopes of a web application is request, response, page, session and application.

```
<jsp:useBean id = "person" class = "com.apress.projsp.person" scope = "request" >
```

```
<jsp:set Property name = "person " peoperty = "*" />
```

```
</jsp:useBean>
```

- ✚ When we use property= "\*" this tells the page implementation class to find each request parameter with the same name as a javaBeanproperty.

- ❖ When a lot of display formatting is required we use JSP. In this case if we were to use a servlet, it would contain lots of lines such as the following

```
Out.println("<a href= \"cart.jsp\">cart</a>");
```

## Chapter-4

### Java server pages standard Tag library

- ✚ Using too many scriptlet on a page reduces its readability and therefore its maintainability and generally makes a jsp page look ugly.
- ✚ More recently a better approach has been for java developers to create their own custom actions (often known as custom tags ) and make them available to web designers via tag libraries
- ✚ One obvious drawbacks of custom tags is that
  - The must be coded
  - Packaged and
  - Tested before use
- ✚ The javaServer pages Tag library (jstl) specification 1.0 was first released in june 2002

#### ❖ **Installing the jstl**

To able to use the jstl we must have the following

- At least a servlet 2.3 and jsp 1.2 complaint container
- An implementation of the jstl specification

#### ❖ **Two JAR file well be needed in the Tomcat/common/lib directory**

- Jstl.jar
- Standard.jar

#### ❖ **Lib directory:**

The lib directory is where we place any JAR files that the web application requires

#### ❖ **Tld directory:**

The tld directory holds the tag library descriptor (TLD) files for the tag libraries

✚ The deployment descriptor contains 4 taglib elements that describe the TLD files available for the test application

1. C.tld
2. Fmt.tld
3. X.tld
4. Sql.tld

## ❖ Understanding JSTL

The JSTL is often referred to as a single tag library in fact it is a collection of 4 tag libraries

➤ Each tag library provides useful actions (or tags) based on the following functional areas:

1. Core
2. Internationalize (I18n) and formatting
3. Relational database access
4. XML processing

➤ One of the primary goals of the JSTL is to simplify the lives of JSP page authors

### 1. The core tag library

Simple tasks such as displaying content based on runtime condition of items as well as a host of URL manipulation features can be achieved via the action provided by the core library

### 2. The internationalization and formatting tag library

Provides a series of actions to aid in the use of three key components associated with internationalization

- Locales
- Resource bundles and
- Basenames

### 3. The SQL tag library

Generally preferable to use MVC architecture

### 4. The XML Tag library

## ❖ Using the JSTL

The core tag library

- It provides jsp page authors with a set of reusable actions
- The JSTL core library can be split further to expose its main functional areas
  1. Scoped variable manipulation
  2. Conditionals
  3. Looping and iteration
  4. URL manipulation

### 1) Scope variable manipulation

**<c:out>Action**

- This action evaluates an expression and outputs it to the current jspWriter
- It is equivalent to the jsp syntax

```
<%=expression%>
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<c:out value="Good Afternoon:"/>
<c:out value="$ {book.author.name}" default="unknown"/>
```

**<c:set>Action**

- Use to set a variable value on a particular web application scope
- `<c:set var="browser" value="$ {header['User Agent']}" scope="session"/>`
- Setting the var =null has the effect of removing the object referred to by var

- Another use of <c:set> is to set property of a second object
- <c:set target = "person " property ="firstName" value="Soundra"/>  
     <c:remove>Action
- It removes a variable from a specific application scope
- <c:removevar= "level" scope="session"/>

<c:catch>Action

It provides a simple mechanism for catching any java.lang.Throwable exceptions that are thrown by any nested actions.

<c:catchvar ="exception">.

## 2) Conditionals

<c:if>action

<c:if> decides to evaluate its body content depending on the value of an evaluated Boolean attribute.

<c:choose><c:when><c:otherwise>action

Similar to if, elseif, else blocks or case/switch statement.

## 3) Looping and iteration

<c:forEach>(for generating data)

It has two alternate syntax

Syntax: if we want to iterate over a collection of object

<c:forEachvar ="varName" items="collection" varStatus="varStatusName">

Body content

<c:forEach>

Syntax: if we want to iterate a set number of times

<c:forEachvar ="varName" varStatus="varStatusName" begin="begin" end="end" step="step">

Body content

<c:forEach>

The object referenced by the items variable can be any one of the following data types:-

- a) An array
- b) An implementation of java.util.Collection
- c) An implementation of java.util.Iterator
- d) An implementation of java.util.Enumeration
- e) An implementation of java.util.Map
- f) A String of comma-separated values.

### Attributes for the <c:forEach>tag:

- Items
- Begin
- End
- Step
- Var
- varStatus

<c:forTokens> for generating String token

java.util.StringTokenizer class

<c:setvar="queryResult" value="Dan, Jeep, Male, 26" scope="request"/>

<c:forTokens items="\$ {queryResult}" delims="," var="token">

<c:out value="\$ {token}"/>

</c:forTokens>

### **Attribute for the <c:forTokens> tag**

items	end	varstatus
delims	step	
begin	var	

### **4) URL-Related Actions**

<c:import>Action

Simplest way to retrieve a file containing XML or XSLT

<c:importurl=http://mybookstore.com/book.xmlvar="url">

Similar to <jsp:include>

<c:url>Action

Provides a handy way of constructing correctly formatted URLs that have the correct URL rewriting rules applied.

<c:url value="http:-----"/>

<c:redirect>Action

This action sends on HTTP redirect to a client.

<c:redirecturl=http://www.mynewurl.com/>

<c:param>Action

### **3. The internationalization and Formatting Tag Library:**

#### **1. Setting the Local**

<fmt:setLocal>Action

This action can be used to provide client specified locale for the processing of a JSP page.

#### **2. Messaging Actions**

<fmt:bundle> and <fmt:setBundle> Action

Can be used to specify a resource bundle, and they're identified by the basename.

<fmt:message>Action

Localized message are retrieved from a resource bundle by using this action.

#### **3. Formatting Actions**

Different locals have different standards regarding the following:

- a) Date and time formats
- b) Number formats
- c) Currency formats
- d) Colours
- e) Page layouts
- f) Address Standards (Zip codes)

<fmt:timeZone> and <fmt:setTimeZone>

<fmt:formatDate>Action is highly flexible and provides the ability to display dates and times in predefined or custom formats by using the conventions as set out by the Java.text.DateFormat class.

### **4. The SQL Tag Library:**

<sql:setDataSource>

Used to configure a data source that represents an underlying physical data store and expose it as either a scoped variable or the data source configuration object.

Javax.servlet.jsp.jstl.sql.DataSource

<sql:query>

<sql:queryvar="users" dataSource="{dataSource}">

Select \* from user where username="den"

</sql:query>

## 5. The XML Processing Tag Library:

### 1. XML core actions-

<x:parse>,<x:out>,<x:set>

### 2. XML Flow Control Action-

<x:if>

<x:choose>,<x:when>,<x:otherwise>

<x:forEach>

### 3. XML Transformation Action-

<x:transform>.

## Chapter-5

### Java server Faces (JSF)

JSF makes easy to build user interface components and pages and to connect those components to business objects. It also automatic the process of java Bean usage a page navigation.

One of the goals of JSF as stated in the specification proposal was to create a standard for Java server application GUIs and to identify synergies in existing frameworks that JSF could take advantages of—

#### Introducing to JSF:

The JSF specification lists the following ways that JSF helps web-application developers to create user interface (UIS)-

- Makes it easy to construct a UI from set reusable UI components.
- Simplifies migration of application data to and from the UI.
- Helps manage UI state across server requests.
- Provides a simple model of wiring client generated events to server side application code.
- Allows custom UI components to be easily built and reused.
- The JSF implementation automatically manages state across user requests. So we do not need to manage or implement any session handling.
- JSF provide an easy way to manage event handling.
- JSF event handling model is similar to those used other UI frame work, such as Java swing. This means that multiple event listener can respond to a single event.
- JSF is based on reusable components; it provides a design that allows us to easily create and integrate our own components on third party components into our JSF enabled applications.

#### The Relationship between JSF and other Java EE Technologies

- JSF is a supporting technology. We use it in conjunction with JSP pages, Servlet or other presentation technologies.
- The primary design pattern of JSF in the MVC pattern.
  - The model is the abstraction of all the domain data in the system.
  - The view is the visualization of use model. In web application the view consist of HTML pages.
  - Controller is the set of components that manage the communication between model and view.

#### Request Processing Life Cycle

There are three request/response pairs-

1. Non-JSF request generates JSF response.
2. JSF request generate JSF response.
3. JSF request generates Non-JSF.

- The page class processes the request and then writes the response back to the client.
- The delivery of user actions or page events is delayed until a new connection is established.
- In JSF the model is composed of business objects that are usually implemented as JavaBeans.
  - The controller is the JSF implementation.
  - UI components are the view
- The JSF life cycle has six phases—
  1. **Pe-store view**: JSF implementation restores the object and data structure that represents the view of the request.
  2. **Apply Request Values**:
    - Any data that was sent as a part of the request is passed to the appropriate UI components with the new data.
    - The data is not yet update, it updates only the UI components with the new data.
  3. **Process Validation**: The data that was submitted with the form is validated.
  4. **Update Model Values**: After all validation are completed the business of objects that make up the application are updated with the validated data from the request.
  5. **Invoke Application**: In this phase the action method of any command button or link that was activated is called.
  6. **Render Response**: The response UI components are rendered and the response is sent to the client.

### **Installing JSF—**

There are two ways that we can make the JSF and JSTL libraries available to the web application running are Tomcat.

#### **1. Six JSF JARS:**

- commons-beanutils.jar
- commons-collection.jar
- commons-digester.jar
- commons-logging.jar
- jsf-api.jar
- jsf-impl.jar

#### **2. Two JSTL JARS:**

- jstl.jar
- standard.jar

### **Using JSF with JSP pages:**

>>The JSP implementation from sun comes with libraries of customer action—

#### **1. HTML custom action category:**

- Input
- Output
- Selection
- Commands
- Miscellaneous

#### **2. The core custom action category:**

- Converters
- Listeners

- Miscellaneous
- Selection
- Validations
- View

#Tomcat 5.5 needs all jar file of version 1.1

In JSF

Model: FlightSearch.java

View: searchFrom.jsp

Controller: faces-config.xml

#JSF use standard JavaBeans which has no explicit constructor. So the compiler provides a default no-argument constructor

#The FlightSearch.java (bean file) use classes from only the java.lang package does not use any special API's or classes. We should be able to compile it without needing to reset CLASSPATH in any way.

#Prefix f: Provide the core JSF functionality for the page and tag.

#Prefix h: Provide HTML elements for the page.

# There is one JSF core tag in the page: the <view> tag. Any page that includes JSF elements must have the <view> tag as the outer most JSF tag. <view>

# The<from> tag creates the HTML form.

<h:form>

# The<input> tag create input text fields in the form.

<h:inputText value="#{flight. Destination}"/>

#The<commandButton> tag creates a button in the form:

<h:commandButton value='search' action="submit"/>

#The action attribute tells the web browsers where to submit the form data.

#The method attribute tells the browsers where to submit a GET or POST request. The JSF tag does not use either of these attribute.

#The specification also requires that all JSF forms use the POST method for submitting form data to web applications.

#The<outputText> tags are obviously used to output text to the page.

<h:outputText value="#{flight.omgination}"/>

#The <inputText> and <outputText> tags are use the #{object.property} syntax to access a property of an object in the page.

**#<managed-been> element contains three subelements----**

<managed-bean>

<managed-bean-name>object name</managed-bean-name>

<managed-bean-class>com.appress.projsp.FlightSearch</managed-bean- class>

<managed-bean-scope>session</managed-bean-scope>

</managed-bean>

#<navigation-rule> element identifies the start page, a condition and the page to navigate to where the condition occurs.

#<navigation-rule> also contain an empty <redirect/> element with this element a response is created by causing the browser to redirect to the searchResults.jsp page.

#The deployment description identifies the controller servlet( Faces Servlet) for the application specifies a servlet mapping indicating which request should be send to the controller servlet and designate the welcome file for the application.

<servlet-mapping>

<servlet-name>Faces Servlet</servlet-name>

<uri-pattern>\*.faces</uri-pattern>

</servlet-mapping>

#Which we control navigation through string value of the action attribute is called static navigation.



#When we control navigation through value-binding-expressions or method-binding-expressions is called dynamic-navigation.

### Using standard converters:

\*The JSF implementation comes with two standard converters, one for numbers

1. <ConvertNumber> **converts String to number**  
One for dates and times
2. <convertDateTime> **converts String to dates or times.**

### Validating Input:

#The JSF provides three standard validations as part of the JSF implementation through the following custom tag---

1. <ValidateDouble Range>
2. <ValidateLong Range>
3. <validateLength>

### Event Handling

There are two basic types of event handling in jsp

1. Value change listeners
  2. Action listeners
1. Value change listener
- Value change listeners are attached to input element such as

- Text fields
- Radio buttons
- Check boxes
- Menus

When we want our application to respond to a value changed event we attach a value change listener

#### 2. Action listener:

Action listener are attached to the two the jsf commands elements

Commands buttons

Command links

- Action listener are provided By the jsf to make it easier to handle action event
- For value change listener The tag<f:valueChangeListener>

With an attribute type that is the class name of the listener.

The interface that the listener must be implemented is-

Javax.faces.event.ValueChangeListener

<f:valueChangeListener type="com.apress.projsp.FlightSearch"/>

### Using messageBundle:

This provides two big advantages

- If we want change the text used in web application, we need to change only the message.properties message bundle.
- We can easily internationalize our application by providing additional message bundles for other languages.

## Chapter – 6

### Tag Files and Simple Tags

-custom tags are known as classic tags.

-classic tags get the behavior from a Java class known as a tag handler.

-when a JSP page is translated the custom tag in the page is translated into a call to the tag handler class.

-Custom tag also known as JSP tag extensions – because they extend the set of built in JSP tags.

-JSP 2.0 introduced 2 major additions to the tag extension mechanism

1) tag files

2) simple tags

1) tag files are custom tag written entirely as JSP pages

2) simply tags are similar to classic tags because they get their behavior from a tag handler class.

### **The needs for custom tags**

- Key factors benefits for the custom tags-

1. Reusability

2. Readability

3. Maintainability

### **Tag and Tag Libraries**

- A tag library (commonly known as taglib) is a collection of custom tags that are typically related each other.

For example, the JSTL core tag library contains all of those tags that help to solve some of the common problems that use encounter when building JSP.

### **Importing and using tags**

If the tag library contained a tag called copyright we simply use the tag as follows-

<tags:copyright></tags:copyright>

<tags:copyright/> (shortened form)

### **Body Content:**

Body content is defined as anything that falls between the start and end tags.

There are 4 types of body content

1) empty(<tags:copyright/>)

2) JSP(<prefix:mytag>Here is some <b>HTML content</b></prefix:mytag>)

3) scriptless(<myTagLibrary:myTag Counter ="\${1+1}" />)

4) tagdependant (the body content is treated as plain text)

### **Attributes:**

- Customtag can be customized through the use of attributes in the same way that methods can be customized through the use of parameters.
- attributes are written as name =value pairs within the tag itself as follows-  
<prefix:myTagattributeName="attribute value"/>

### **JavaBeans vs. CustomTags:**

Java Beans are reusable component and the JSP specialization provides a built-in-mechanism for integrating and utilizing the features provided by JavaBean. Although both technologies can be used to encapsulating and abstracting data away from the JSP page there are significant difference between the two-

- JavaBeans use to representing and storing information and state.
- Custom tags used to represent and implement actions that occur on those JavaBeans, as well as logic related to the presentation of information. example – JSTL

### **Difference between Simple vs Classic Tags:**

- javax.servlet.jsp.tagext.Taginterface this class containing the functionality provided the tag is called the tag handles.
- Simpletag is similar to classic tag because they get their behavior from a tag handler class.

### **Using Tag Files.**

Tag files are custom tags written entirely as jsp page. It is a very simple way for context and functionality to be abstracted away from jsp page and into reusable components.

### **Reusing context with include JSP files**

<%@include file="Copyright.jsp" %>

### **Reusing context with Tag files.**

One of the problems with include files is that with many include directives on the page. The context of the each include file can sometimes be cryptic.

### **Defining content in a Tag file.**

Extension of the file .Tag

### Using the Tag file.

```
<jsp:includepage:bou.jsp>
```

```
<jsp:param name="color" value="red">
```

### Templating with Tag files.

```
<%@attribute name="color" required="yellow" rtexprvalue="false" %>
```

### Why we use Tag files?

- Tag files provide much cleaner way to build and subsequently use templates on JSP pages.
- Using tag files to build templates is a gateway to separate the context from the presentation of that context.

### Using simple Tags.....

#### The simple Tag interface.

- Provide the simple Tag with information about its execution environment.
- Provides a method for executing the functionality encapsulated by the simple Tag handler.

#### The basic Tag life cycle.

- Setting the context: `SetJspContext(JspContext)`
- Setting the parent: `SetParent(JspTag)`
- Setting the Body: `setJspBody(JspFragment)`
- Executing the functionality: `doTag()`

#### Deploying the tag library.

- Jsp file-করে রাখতে হবে jsp example/simple tag folder-এ
- TLD file-করে রাখতে হবে WEB-INF/tld folder-এ

#### The tag life cycle with attributes

1. `setJspContent(JspContent)`
2. `setParent(jspTag)`
3. `setAttribute 1 ()`
4. `setAttribute N()`
5. `setJspbody (JspFragment)`
6. `doTag()`

\*\*the `dotag()` method is responsible for outputting the information.

\*\*The tag handler must declare a scatter method with the following signature

```
Public void setName(string s)
```

```
<prefix:myTag attributes1="abc"
```

```
Attribute N="def"/>
```