

# Module 3

## ❖ JSP: ( Introduction)

The first JavaServer Pages specification was released in 1999. Originally JSP was modeled after other server-side template technologies to provide a simple method of embedding dynamic code with static markup. When a request is made for the content of a JSP, a container interprets the JSP, executes any embedded code, and sends the results in a response.

- Jsp enables to mix static HTML with dynamic generated content.
- Jsp as HTML with java code inside.
- Jsp pages are translated into servlets.
- The Servlets are compiled and at the request time the compiled servlet are executed.
- In writing jsp pages is really another way of writing servlet.

## Need For JSP (Disadvantages of Servlet)

- ✚ Servlets are not good at presentation.
  - It is hard to write and maintain HTML i.e using print statement to generate HTML.
  - Changing the Look and feel the application requires the servlet code to be updated is recompiled.
- ✚ We can't use standard HTML tools.
  - HTML is inaccessible to non java developers. Web developers expert who doesn't know JAVA Programming.

## Benefits of JSP:

- ✚ Easier to write and maintain HTML.
- ✚ We can use website development tools (Dreamweaver)
- ✚ We can divide our development team (presentation and Business logic)

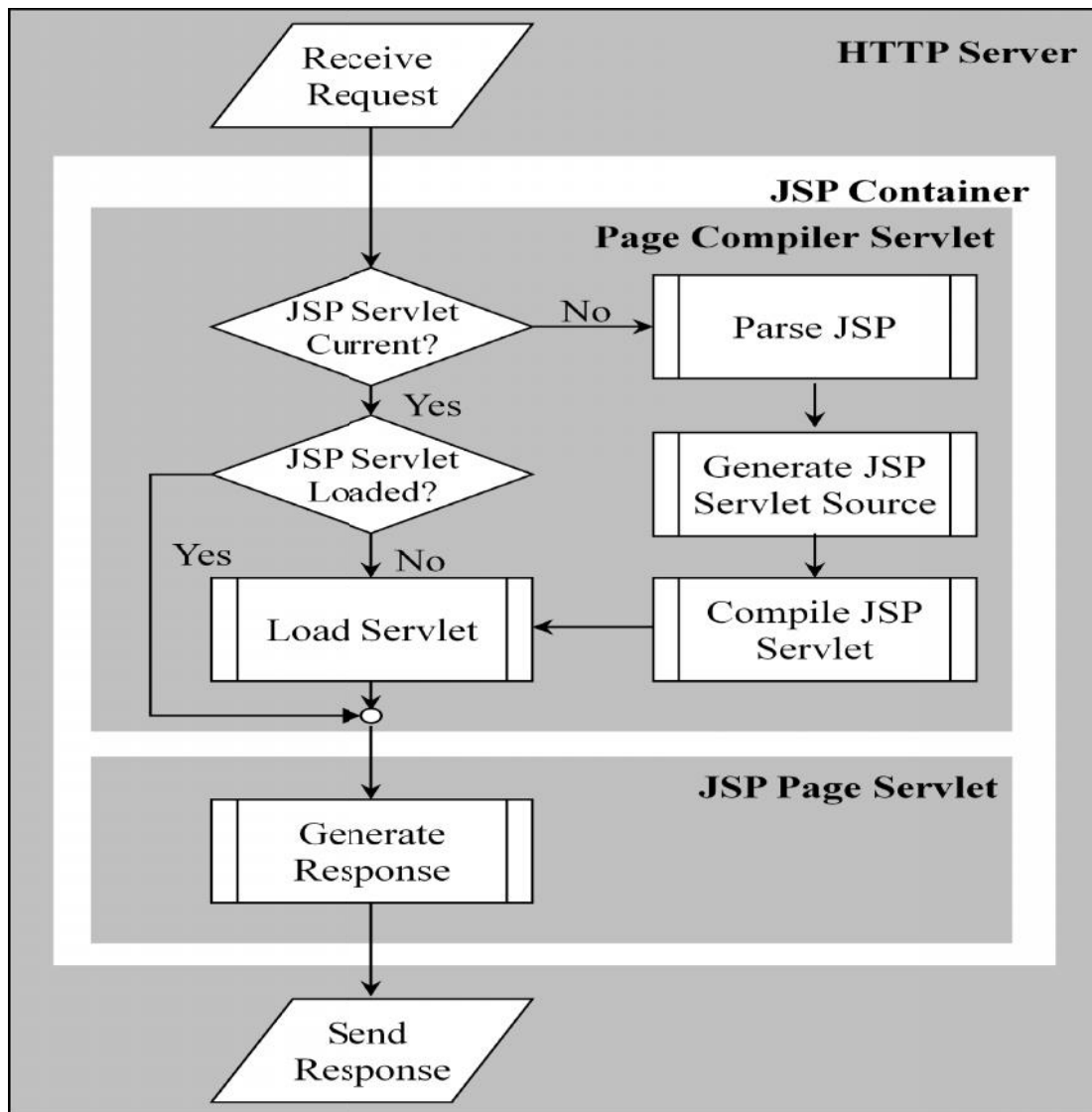
## ❖ JSP Architecture

✚ JSPs run in two phases

- Translation Phase
- Execution Phase

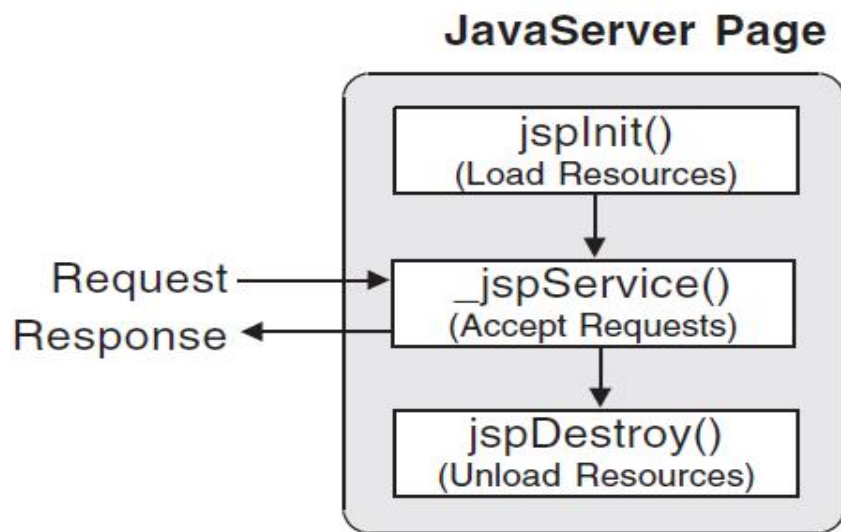
✚ In translation phase JSP page is compiled into a servlet called JSP Page Implementation class

✚ In execution phase the compiled JSP is processed



## ❖ JSP Life Cycle

JSP follows a three-phase life cycle: initialization, service, and destruction as shown in the fig below.



- ✚ While a JSP does follow the Servlet life cycle, the methods have different names. Initialization corresponds to the `jspInit()` method, service corresponds to the `_jspService()` method, and destruction corresponds to the `jspDestroy()` method.
- ✚ The three phases are all used the same as a Servlet and allow a JSP to load resources, provide service to multiple client requests, and destroy loaded resources when the JSP is taken out of service.
- ✚ JSP is designed specifically to simplify the task of creating text producing `HttpServlet` objects and does so by eliminating all the redundant parts of coding a Servlet.
- ✚ Unlike with Servlets there is no distinction between a normal JSP and one meant for use with HTTP. All JSP are designed to be used with HTTP and to generate dynamic content for the World Wide Web.
- ✚ The single JSP `_jspService()` method is also responsible for generating responses to all of the HTTP methods.

HelloWorld.jsp:

```
<html>
<head>
<title>Hello World!</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

Web Application automatically deploys any JSP to a URL extension that matches the name of the JSP. HelloWorld.jsp is also actually compiled into equivalent Servlet code.

This is done in what is called the translation phase of JSP deployment and is done automatically by a container.

JSP translation both is and is not something of critical importance for a JSP developer to be aware of. JSP translation to Servlet source code is important because it explains exactly how a JSP becomes Java code. While it varies slightly from container to container, all containers must implement the same JSP life cycle events.

Understanding these life cycle methods helps a JSP developer keep code efficient and thread-safe. However, JSP translation is not of critical importance because it is always done automatically by a container.

#### ❖ Four different elements are used in constructing JSPs

- Scripting Elements
- Implicit Objects
- Directives
- Actions

## ❖ Scripting Elements

There are three types of Scripting elements.

1. Declaration
2. Scriptlets
3. Expression

### 1. Declaration :

✚ JSP declaration allows us to define methods and variables that get inserted into main body of servlet class and outside the `_jspService()` .

✚ It doesn't produce any output that is sent to the client.

✚ Syntax:-

```
<%! Script code %>
```

✚ Example 1:-

```
<h1>some heading</h1>
<%!
    private String randomHeading(){
return ("<h2>" + math.random() + "</h2>");
    }
%>
<body>
    <%out.println("--" + randomHeading() + "--");%>
</body>
```

## Example 2:-

```
<%!  
    public void myFunOne(){  
        System.out.println("function one");  
    }  
    public void myFunTwo(){  
        System.out.println("function two");  
    }  
    int myVariable =123;  
%>
```

- ✚ The function and variable are available to the jsp page as well as to the servlet in which it is compiled.

## 2. Scriptlets:

- ✚ Scriptlets are used to embed java code in jsp pages.
- ✚ Content of JSP code goes into `_jspService()` method.
- ✚ Scriptlets have access to the automatically defined variables(request, response, session, out ,application etc.)
- ✚ We can't define methods inside scriptlet.

## ✚ Systax:

```
<%    scriptlet code    %>
```

## ✚ Example 1:

```
<html>  
  
<body>
```

```
<% out.println("This is my first scriplet code")%>
```

```
<%  
    int x=5,y=7;  
    int z=x+y;  
    out.println(z);  
%>  
</b>  
</html>
```

#### Example 2:

```
<% if(Math.random()<0.5){ %>  
    Good  
<% } else { %>  
    Bad  
<% } %>
```


### 3. Expression


 Expressions are used to write dynamic content back to the browser.

 JSP expression used to insert values directly into output.

 Syntax:

```
<%= expression %>.
```

 The expression is evaluated and then converted to string and inserted in the page.

 Evaluation is performed at runtime and thus has full access to information about the request.

✚ Example:-

```
<%= Math.random()%>
```

```
<%= Math.sqrt(9)%>
```

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

✚ In expression we can use a number of predefined variables(implicit objects)

## ❖ JSP implicitly Objects

### Scope

Implicit objects provide access to server side objects e.g. request, response, session etc.

There are four scopes of the objects

- Page: Objects can only be accessed in the page where they are referenced
- Request: Objects can be accessed within all pages that serve the current request.
- (Including the pages that are forwarded to and included in the original jsp page)
- Session: Objects can be accessed within the JSP pages for which the objects are defined
- Application: Objects can be accessed by all JSP pages in a given context

❖ JSP supports following implicit objects.

request: Reference to the current request

response: Response to the request

out: Object that writes to the response output stream



session: session associated with current request

application: Servlet context to which a page belongs

pageContext: Object to access request, response, session and application associated with a page

config: Servlet configuration for the page

page: instance of the page implementation class (this)

exception: Available with JSP pages which are error pages

✚ request:- This is the `HttpServletRequest` object associated with request. This is an instance of `javax.servlet. HttpServletRequest` object. Each time a client request a page , the JSP engine creates a new object to represent that request. It gives us to access to the request parameters, request type and incoming Http Header information like Cookies etc.

✚ response:-The response object is an instance of `HttpServletResponse` object. Just as the server creates the request object it also creates an object to represent the response to the client. The response object defines the interface that deals with creating new Http Headers. Through this object we can add new cookies, HTTP Status code etc.

✚ out: - the out is an implicit object is an instance of `javax.servlet.jsp.JspWriter` object. It is used to send content in a response. The initial jsp writer object is instantiated differently depending on whether the page is buffered or not. Buffering can be easily turned off by using the “buffered=false” attribute of the page directive. Its visibility scope is specific to each jsp page.

```
out.println("Sample Text"); // set text to response object
```

out.flush();

- ✚ session:- the session object is an instance of `javax.servlet.http.HttpSession` and behaves exactly same way that session object behaves under java servlet. Session object is used to track the client session between client request. Its scope is specific to each browser window. The session is declared by specifying the session attribute true in a page directive.
- ✚ application:- the variable is an instance of `javax.servlet.ServletContext` as obtained by `getServletContext()`. Servlet and jsp page can store persistent data in the `ServletContext` object. `ServletContext` has `setAttribute()` and `getAttribute()` to store data associated with specific keys. It is shared by all servlet and jsp pages on the web application.
- ✚ pageContext :- The pageContext implicit scripting variable is an instance of a `javax.servlet.jsp.PageContext` object. It represents the context of a single java server pages including all other implicit objects , methods for forwarding to and including web application resources and scope for binding objects to the page.
- ✚ config:- It is an instance of `javax.servlet.ServletConfig`. The object allows the jsp programmer to access the servlet or jsp initialization parameters such as path or file location provided in the web application deployment descriptor.
- ✚ page :- The page object is an actual reference to the instance of the page. It can be thought of an object that represents the entire jsp page. The page object is direct synonyms for the this object.
- ✚ Exception: - This is the type of `java.lang.Throwable` object. Exception object is a wrapper containing the exception thrown from previous page. It is typically used to generate an appropriate response to the error conditions.

## ❖ Directives

A jsp directive affects the overall structure of the servlet that results from jsp page.

The jsp specification defines three directives.

1. page directive
2. include directive
3. taglib directive

✚ page directive :- The tag provides page specific properties such as character encoding, the content type for the page response and whether the page should have implicit session object or not.

Example –

```
<% @ page import="pkgname.*" session="false/true" %>
```

✚ include directive :- it is used to include the content of an external file.

Example:-

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

```
<% @ include file="included.jsp" %>
```

✚ taglib directive:- It is used to import custom actions defined in tag libraries.

Example:-

```
<% @ taglib tagdir="/WEB-INF/tools/cool" prefix="cool" %>
```

## ❖ Page Directives Basics and Types

It provides the page specific information to the jsp container like type of content jsp produced, the default scripting language etc. Attributes of the page directive are import, contentType, pageEncoding, session, isELIgnored, errorPage, buffer, autoFlush, isThreadSafe, isErrorPage, language, extends and info.

- ✚ language –the language attribute defines the scripting language to be used by the scriptlet, expression and declaration occurring in the jsp.

`<%@ page language="java" %>`

- ✚ Extends: -the extends attributes designates the super class of the servlet that will be generated from the jsp page. The attribute is normally reserved for the developers of vendors that implement fundamental changes to the way in which the page operates.

`<%@ page extends="pkgname.class" %>`

- ✚ Import –it is used to specify the package and classes that should be imported to the servlet in to which the jsp page get translated. The default import list is `java.lang.* , javax.servlet.* , javax.servlet.http.* , javax.servlet.jsp.* ; .`

`<%@ page import="pkgname.class" %>`

- ✚ Session :- the session attribute controls whether the page participated in the HttpSession or not.

`<%@ page session="true"%>`

If true the implicit scripting variable session refers to the current /new session for the page. If false the page doesn't participated in the session.

- ✚ Buffer:-the buffer attribute specifies the size of the buffer used by out variable which is a jsp writer.

If the attribute value is none then there is no buffering and output is written directly through to the appropriate servlet's response PrintWriter.

If the buffer size is specified, then the output is buffered with the buffered size and not sent to the client until at least the buffered size. Depending upon the value of `autoFlush` , the content of the buffer is either automatically flushed or exception thrown when overflow occurs.

`<%@ page buffer="size in kb" %>`

`<%@ page buffer="none" %>`

- ✚ isThreadSafe :-The isThreadSafe attribute controls weather the servlet that results from the jsp page will allow concurrent access or will ignore that no servlet instance posses more than one request at a time.  
If the value is true the jsp container may choose to dispatch multiple client requests to the page simultaneously. The default value is true.  
`<% @ page isThreadSafe ="true" %>`
- ✚ isErrorPage:-The isErrorPage attribute indicate if the current jsp page is intended to be an Error page for other JSP. If true then the implicit scripting variable exception is defined. If false then the exception implicit object is unavailable.
- ✚ errorPage:-Define a relative URL to a resource in the web application to which any java programming language Throwable object thrown.
- ✚ contentType:- the contentType attribute defines the character encoding for the JSP page and the MIME type for the response of the jsp page. The default value of the contentType is “ text/html” with ISO-8859-1 character encoding for regular jsp.
- ✚ pageEncoding – pageEncoding defines the character encoding for the jsp page. The default of the pageEncoding attribute is ISO-8859-1 for regular jsp and UTF-8 for the jsp in XML.
- ✚ Info:-String returned by the getServletInfo() of the compiled servlet

### Page Directives Example

```
<% @
    page language="java"
    buffer="10kb"
    autoflush="true"
    errorPage="/error.jsp"
    import="java.util.*, javax.sql.RowSet"
%>
```

## ❖ EL(Expression Language)

- ✚ One of the features of JSP 2.0 is a jsp specific expression language commonly called as JSP EL.
- ✚ EL provides a simple and elegant solution to embedding expression in jsp and provides a method to avoiding a traditional JSP expression `<%= %>`
- ✚ JSP EL is simple , powerful and alternative to scripting elements.
- ✚ Features of JSP EL such that it can be used anywhere in JSP.
- ✚ By default JSP EL is disabled for web application that uses web.xml file defined by servlet 2.3 or the followings.
- ✚ Application that uses servlet 2.4 defined web.xml , the JSP EL is automatically enabled.
- ✚ The JSP EL handles both expression and literals.
- ✚ Expression are always enclosed with `${ }` character.

## ❖ Expression Language supports following capabilities:-

- ✚ Consise access to stored objects- To output a scoped variable i.e object stored with `setAttribute` in the `PageContext`, `HttpServletRequest`, `HttpSession` or `ServletContext`.
- ✚ Shorthand notation for bean properties:- To output a company name property of a scoped variable Company we can use- `${Company.companyName }`
- ✚ Simple access to collection elements – To access an element of an `Array`, `List`, `Map` we use `${variable[index/key] }`
- ✚ A small but useful set of operators :- we can use several set of arithmetic , relational , logical or empty testing operators.
- ✚ Conditional output :- `${ test ? operation 1 : operation 2 }`
- ✚ Automatic type conversion :- Remove the need of most type casts.
- ✚ It shows empty value instead of error message.
- ✚ Access the standard types of request data , we can use one of the several predefined implicit objects.

➤ Deactivation the EL in entire web application

The jsp 2.0 expression language is automatically deactivated in the web application whose deployment descriptor refers to servlet specification version 2.3 or earlier.

➤ Deactivating the EL in multiple JSP Pages:-

In web application whose deployment descriptor specifies servlet 2.4 (JSP 2.0) we can use el-ignored sub element of jsp-property . web.xml element to designate the pages in which expression language should be ignored.

Putting <el-ignored> in Deployment Descriptor

```
<web-app>

    <jsp-config>

        <jsp-property-group>

            </jsp-property-group>

        </jsp-config>

</web-app>
```

➤ Deactivating the EL in individual JSP Pages:-

To disable EL evaluation in an individual page, supply false as the value of isEnabled attribute of page directive.

```
<%@ page isELEnabled="false" %>
```

## ❖ Using Custom Tags

A custom tag is a user defined JSP language element. When a jsp page containing a custom tag is translated into servlet, the tag is converted to operations on an object is called a tag handler.

JSP tag extensions let you create new tags that can be inserted directly into a java server page like built-in tags.

Jsp 2.0 specification introduced simple tag handlers for writing these custom tags.

To write a custom tag we can simply extend SimpleTagSupport class and override doTag() method.

Custom tags provide a method to cleanly separate logic from content.

Custom tags are easy to use.

Custom tags are portable.

Example:-

HelloTag.java

```
package com.example;
```

```
import javax.servlet.jsp.tagext.*;
```

```
import javax.servlet.jsp.*;
```

```
import java.io.*;
```

```
public class HelloTag extends SimpleTagSupport{
```

```
    public void doTag() throws JspException,IOException{
```

```
        JspWriter out=getJspContext().getOut();
```

```
        out.println("Hello Custom Tag!");
```

```
    }
```

```
}
```

Create a tld file



### Custom.tld

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD</short-name>

  <tag>
    <name>Hello</name>
    <tag-class>com.example.HelloTag</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

### Hello.jsp

```
<% @ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<% @ taglib prefix="c" uri="WEB-INF/custom.tld" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
  <c:Hello/>
</body>
</html>
```

## ❖ JSP-STANDARD TAG LIBRARY (JSTL)

## ❖ Exception Handling in JSP

If an exception is to be thrown from either servlet or jsp, tomcat automatically generates a simple error page.

Ex:-

```
    Throwable.jsp
    <%
    If(true)
    Throw new Exception("Exception occurred");
    %>
```

There are 2 types of exception 1. Translation time error 2. Runtime exception  
Translation time error occurs during page compilation, this results in an internal server error -500

An exception on the other hand occurs when the page is compiled and the servlet is running.

Exception in JSP can be handled in 3 ways:-

1. Java Exception handling mechanism
2. Dealing with exception with page directive
3. Dealing with exception with deployment descriptor.

### **Java Exception Handling mechanism**

By using try-catch block.

```
    <%
    try{
    }catch(Exception e){
    }
    %>
```

### **Dealing with Exception by page directive:-**

Two attributes of the page directive `errorPage` and `isErrorPage` are used to deal with exception.

### **In Deployment Descriptor:-**

Error page can be defined on a per-web-application basis by web application Deployment Descriptor i.e `web.xml`.

The `error-page` element is used to define error handling based on the type of exception thrown.

The functionality can be configured in `web.xml` by using `error-page` element with `exception-type` and `location` sub element.

```
<error-page>
    <error-type>Throwable object</error-type>
    <location>/Relative URL </location>
</error-page>
```

When even a web component throws an exception , the web container call the respective error page present in the location tag.

Http Status Code:

Example-

```
<error-page>
    <error-code>404</error-code>
    <location>/FileNotFound.jsp</location>
</error-page>
```

## ❖ Session Management

✚ Session in JSP:- Session handling is mandatory when a request of data need to be sustained for future use.

✚ The following are some of the methods to handle session.

- Whenever a request carries the server generated unique session id which is stored in client machine.
- Cookies stores information in client's browser.
- In URL rewriting the session information is appended to the end of the URL.
- Storing some information in hidden fields.

✚ Session creation and identification in JSP pages:

The jsp pages that need session management must use the following page directive.

```
<%@ page session="true" %>
```

By using the above directive in jsp page we can refer to the session which is associated by means of implicit variable session.

✚ Methods of HTTP Session interface:

- public String getId() – it returns a unique identifier value.
- public long getCreationTime() – it returns the time stamp i.e when the session was created.
- public void invalidated() – it invalidated the existing session.

Example-

Form1.jsp

```
<% @ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

    <form method="post" action="session1.jsp">

        <input type="text" value="" name="username" />

        <input type="submit" value="enter" />

    </form>

</body>

</html>
```

## Session1.jsp

```
<% @ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

    <%

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

        if(name!=null)

        {

            session.setAttribute("username",name);

        }

    %>

    <a href="DisplaySession.jsp">next page</a>

</body>

</html>
```

## DisplaySession.jsp

```
<% @ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

    Welcome to session continue page

    <%=session.getAttribute("username") %>

</body>

</html>
```

❖ JSP With JAVA Beans

## ❖ JSP with Database:

Connect to JSP to access database:

The jsp program has to do several things.

- Identifying the source file for java to handle SQL.
- Load driver programs that lets java connect to database
- Execute the driver to establish connection

JSP systax to do this as follows:

```
<% @ page import="java.sql.*" %>
<%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=null;
con =DriverManager.getConnection("jdbc:odbc:cit"," "," ");
out.println("Database Connected");
%>
```

Explanation :-

The page directive at the top of the page allows the program to use methods and classes from java.sql\* package that knows how to handle SQL queries.

The Class.forName methods loads the driver. The

DriverManager.getConnection() methods connect the program to the database identified by the data source "cit" allowing the program to make queries insert, select etc.

The DriverManager.getConnection() creates a connection object which will be used latter when we make SQL queries.

In order to make queries we will need a Statement Object.

Inorder to make a Statement object we need a connection object.

In order to make a Connection object we need to Connect to the Database.



In order to connect to the database we need to load the Driver that can make the connection.

#### Inserting Records in database using JSP

In order to insert or to do any kind of SQL queries we have to

1. Create a statement object- which has the methods for handling SQL
2. Define SQL queries – such as insert queries
3. Execute the queries

```
Statement st=conn.createStatement();
```

```
String s="insert into Student values(1,'Bswajit');"
```

```
st.executeUpdate(s);
```

the conn object that has previously created has the method that allows to create a statement object by calling createStatement() method.

#### Inserting data from an HTML Form in database using JSP

In a three-tier application, the JSP acquires the data to be inserted into a database from an HTML Form.

Step involves

1. An HTML Form with named input fields.
2. JSP statement that access the data sent to the server by the Form
3. Construction of an insert query based on this data by the JSP Program.
4. Execute the query by the JSP program.

#### Retrieving the data from database using JSP ResultSet

A ResultSet object is essentially a table of returned results as done for any SQL select statements. The executeQuery() method of statement object is used in this case. The steps involved in a select retrieval are:-

1. Construct the desired Select Query as a java String.
2. Execute the executeQuery() method and saving the result in a ResultSet object.
3. Process the ResultSet using two of its methods next() and getString()
  - a. Let r be the ResultSet object holds set of results  
r.next() moves the pointer to the next row of the result set
  - b. r.getString("attribute-name") extracts the given attribute value from the current row pointed by r .

Example:-

```
String s="select * from Students";
ResultSet r= stmt.executeQuery(s);
while(r.next())
{
    out.println("<br>Sl.No"+r.getString("id"));
    out.println("<br>Name"+r.getString("name"));
}
```

Note: if the attribute is not a String for example if it is a numeric value then we have to use     int N= r.getInt("attribute-name"). Similarly  
double d=r.getFloat("attribute-name");

## ❖ MVC Architecture:

One of the most common design patterns is Model View Controller.

Model:

Model does all computational work. All communication with model is done via methods. Model encapsulates all the core data and functionality.

View:-

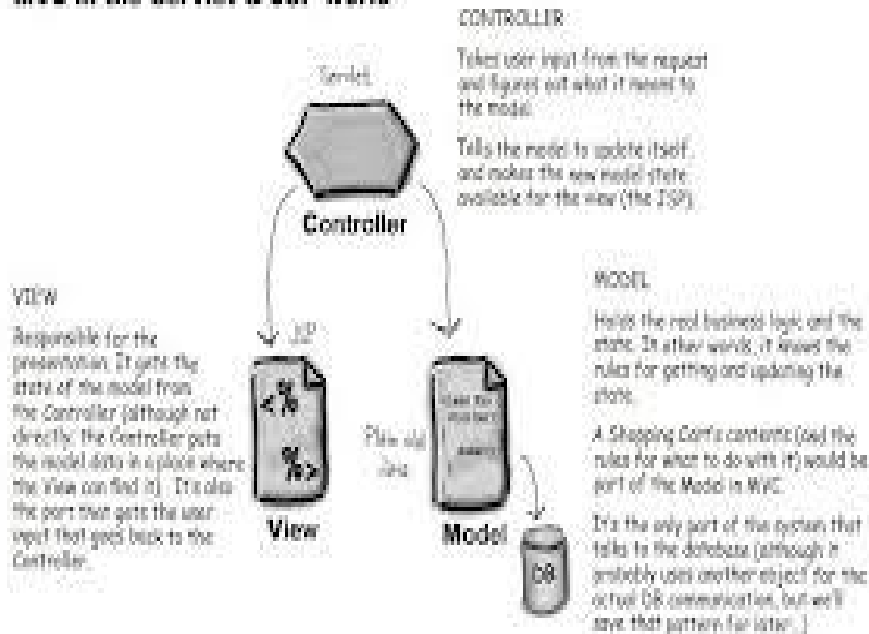
View encapsulates the presentation of the data. View presents the data in some form to a user, in the context of some application function. View gets result from the controller. View can also get result directly from the model.

It is the HTML pages that is return to the user, is frequently created by JSP.

Controller:

Controllers tell the model what to do. The Servlet class acts as the controller. The Servlet gives any relevant information from the user request to the model. The Servlet takes the result and passes them to the view.

### MVC in the Servlet & JSP world



## ❖ Spring Framework

Spring is an open source layered Java/J2EE application framework

The Spring Framework is licensed under the terms of the Apache License, Version 2.0 and can be downloaded at:

<http://www.springframework.org/download>

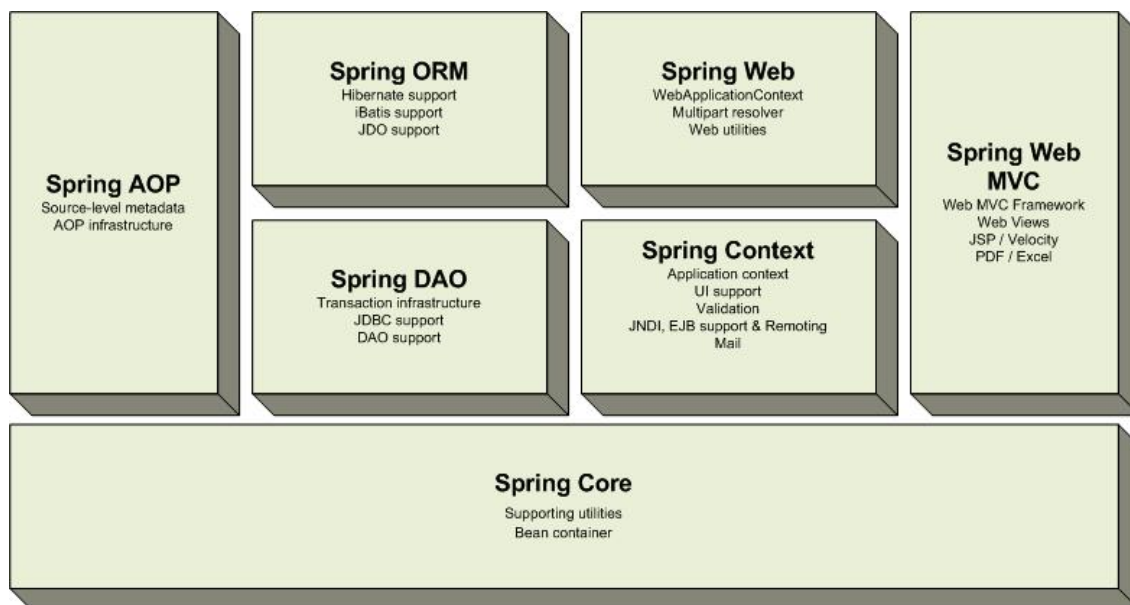
### ❖ Mission:

- J2EE should be easier to use
- It's best to program to interfaces, rather than classes. Spring reduces the complexity cost of using interfaces to zero.
- JavaBeans offer a great way of configuring applications
- OO design is more important than any implementation technology, such as J2EE
- Checked exceptions are overused in Java. A framework shouldn't force you to catch exceptions you're unlikely to be able to recover from.
- Testability is essential, and a framework such as Spring should help make your code easier to test
- Spring should be a pleasure to use
- Your application code should not depend on Spring APIs
- Spring should not compete with good existing solutions, but should foster integration. (For example, JDO and Hibernate are great O/R mapping solutions. Don't need to develop another one).

## Modules of the Spring Framework:

The Spring Framework can be considered as a collection of frameworks-in-the-framework:

- Core - Inversion of Control (IoC) and Dependency Injection
- AOP - Aspect-oriented programming
- DAO - Data Access Object support, transaction management, JDBC-abstraction
- ORM - Object Relational Mapping data access, integration layers for JPA, JDO, Hibernate, and iBatis
- MVC - Model-View-Controller implementation for web-applications
- Remote Access, Authentication and Authorization, Remote Management, Messaging Framework, Web Services, Email, Testing, ...



It is Very loosely coupled, components widely reusable and separately packaged.

## **Advantages of Spring Architecture:-**

- Enable you to write powerful, scalable applications using POJOs
- Lifecycle – responsible for managing all your application components, particularly those in the middle tier container sees components through well-defined lifecycle: init(), destroy()
- Dependencies - Spring handles injecting dependent components without a component knowing where they came from (IoC)
- Configuration information - Spring provides one consistent way of configuring everything, separate configuration from application logic, varying configuration
- In J2EE (e.g. EJB) it is easy to become dependent on container and deployment environment, proliferation of pointless classes (locators/delegates); Spring eliminates them
- Cross-cutting behavior (resource management is cross-cutting concern, easy to copy-and-paste everywhere)
- Portable (can use server-side in web/ejb app, client-side in swing app, business logic is completely portable)