

## UNIT III- JAVA SERVER PAGE(JSP)

### What is JSP?

- ✓ Java Server Pages (JSP) is a technology which is used to develop web pages by inserting [Java](#) code into the HTML pages by making special JSP tags. The JSP tags which allow java code to be included into it are `<% ----java code----%>`.
- ✓ It can consist of either HTML or XML (combination of both is also possible) with JSP actions and commands.
- ✓ It can be used as HTML page, which can be used in forms and registration pages with the dynamic content into it.
- ✓ Dynamic content includes some fields like dropdown, checkboxes, etc. whose value will be fetched from the database.
- ✓ This can also be used to access JavaBeans objects.
- ✓ We can share information across pages using request and response objects.
- ✓ JSP can be used for separation of the view layer with the business logic in the web application.

### Why use JSP?

- In Java server pages JSP, the execution is much faster compared to other dynamic languages.
- It is much better than Common Gateway Interface (CGI).
- Java server pages JSP are always compiled before its processed by the server as it reduces the effort of the server to create process.
- Java server pages JSP are built over Java Servlets API. Hence, it has access to all Java APIs, even it has access to JNDI, JDBC EJB and other components of java.
- JSP are used in MVC architecture (which will be covered in MVC architecture topic) as view layer.
- The request is processed by a view layer which is JSP and then to servlet layer which is java servlet and then finally to a model layer class which interacts with the database.

### Advantages of JSP

- The advantage of JSP is that the programming language used is JAVA, which is a dynamic language and easily portable to other operating systems.
- It is very much convenient to modify the regular HTML. We can write the servlet code into the JSP.
- It is only intended for simple inclusions which can use form data and make connections.
- JSP can also include the database connections into it. It can contain all type of java objects.
- It is very easy to maintain
- Performance and scalability of JSP are very good because JSP allows embedding of dynamic elements in HTML pages.
- As it is built on Java technology, hence it is platform independent and not depending on any operating systems.
- Also, it includes the feature of multithreading of java into it.
- We can also make use of exception handling of java into JSP.
- It enables to separate presentation layer with the business logic layer in the web application.
- It is easy for developers to show as well as process the information.

## Life Cycle

JSP Life Cycle is defined as translation of JSP Page into servlet as a JSP Page needs to be converted into servlet first in order to process the service requests. The Life Cycle starts with the creation of JSP and ends with the disintegration of that.

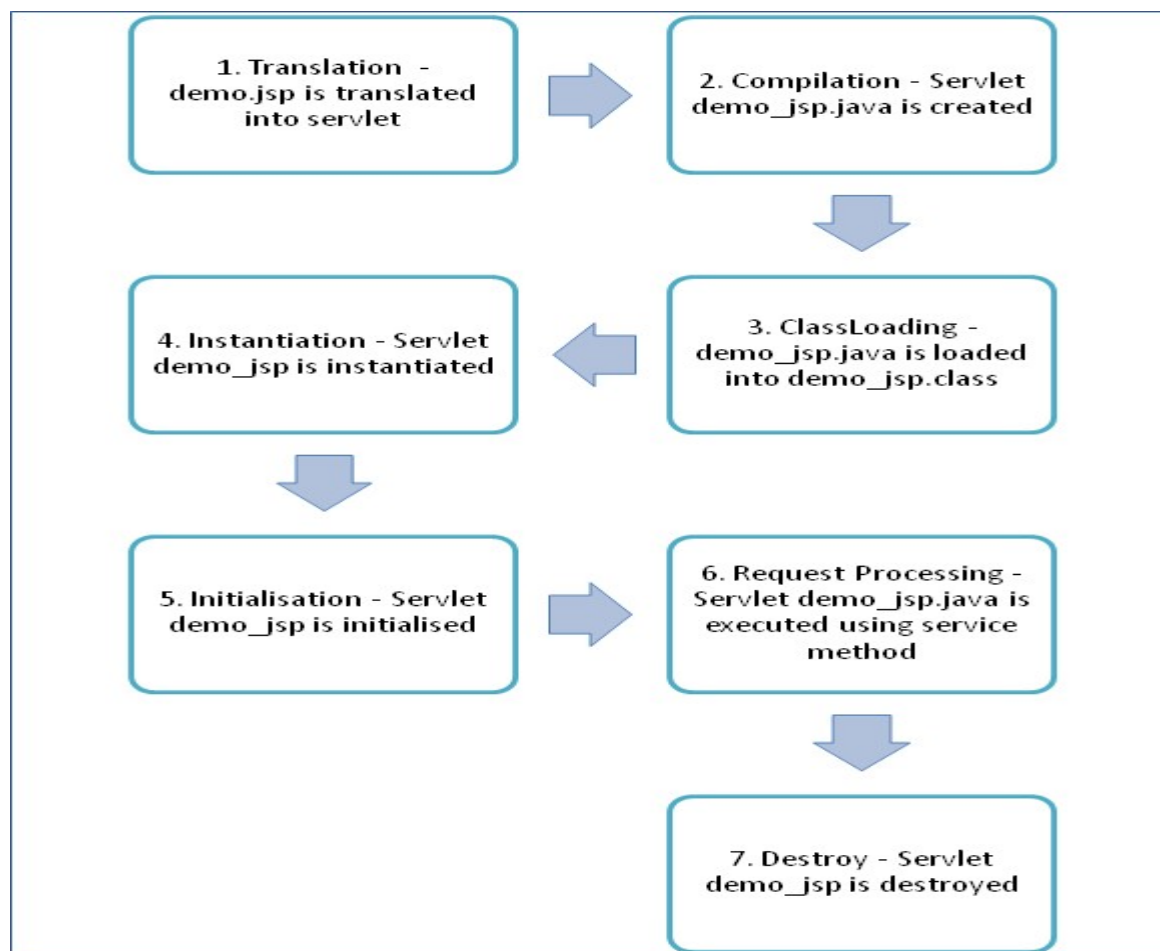
### Different phases of JSP Life Cycle

When the browser asks for a JSP, JSP engine first checks whether it needs to compile the page. If the JSP is last compiled or the recent modification is done in JSP, then the JSP engine compiles the page.

Compilation process of JSP page involves three steps:

- Parsing of JSP
- Turning JSP into servlet
- Compiling the servlet

JSP Lifecycle is depicted in the below diagram.



Following steps explain the JSP life cycle:

1. Translation of JSP page
2. Compilation of JSP page(Compilation of JSP page into \_jsp.java)
3. Classloading (\_jsp.java is converted to class file \_jsp.class)
4. Instantiation(Object of generated servlet is created)
5. Initialisation(\_jspinit() method is invoked by container)
6. Request Processing(\_jspervice() method is invoked by the container)
7. Destroy (\_jspDestroy() method invoked by the container)

Let us have more detailed summary on the above points:

### 1. Translation of the JSP Page:

A [Java](#) servlet file is generated from a JSP source file. This is the first step of JSP life cycle. In translation phase, container validates the syntactic correctness of JSP page and tag files.

- The JSP container interprets the standard directives and actions, and the custom actions referencing tag libraries (they are all part of JSP page and will be discussed in the later section) used in this JSP page.
- In the above pictorial description, demo.jsp is translated to demo\_jsp.java in the first step
- Let's take an example of "demo.jsp" as shown below:

1. <html>
2. <head>
3. <title>Demo JSP</title>
4. </head>
5. <%
6. int demovar=0;%>
7. <body>
8. Count is:
9. <% Out.println(demovar++); %>
10. <body>
11. </html>

Demo JSP Page is converted into demo\_jsp servlet in the below code.

---

```

1  Public class demp_jsp extends HttpServlet{
2      Public void _jspervice(HttpServletRequest request, HttpServletResponse response)
3          Throws IOException, ServletException
4      {
5  PrintWriter out = response.getWriter();
6  response.setContentType("text/html");
7  out.write("<html><body>");
8  int demovar=0;
9  out.write("Count is:");
10 out.print(demovar++);
11 out.write("</body></html>");
12 }
13 }
14

```

### 2. Compilation of the JSP Page

- The generated java servlet file is compiled into java servlet class
- The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page.
- In the above pictorial description demo\_jsp.java is compiled to a class file demo\_jsp.class

### 3. Class loading

- Servlet class that has been loaded from JSP source is now loaded into the container

### 4. Instantiation

- In this step the object i.e. the instance of the class is generated.
- The container manages one or more instances of this class in the response to requests and other events. Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.
- A JSPPage interface which is provided by container provides init() and destroy () methods.
- There is an interface HttpJSPPage which serves HTTP requests, and it also contains the service method.

### 5. Initialization

```
public void jspInit()
{
    //initializing the code
}
```

- \_jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
- Once the instance gets created, init method will be invoked immediately after that
- It is only called once during a JSP life cycle, the method for initialization is declared as shown above

### 6. Request processing

```
void _jspservice(HttpServletRequest request HttpServletResponse response)
{
    //handling all request and responses
}
```

- \_jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle
- For this phase, it has to go through all the above phases and then only service method can be invoked.
- It passes request and response objects
- This method cannot be overridden
- The method is shown above: It is responsible for generating of all HTTP methods i.e GET, POST, etc.

## 7. Destroy

```
1. public void _jspdestroy()  
2. {  
3.     //all clean up code  
    }
```

- `_jspdestroy()` method is also invoked by the container
- This method is called when container decides it no longer needs the servlet instance to service requests.
- When the call to destroy method is made then, the servlet is ready for a garbage collection
- This is the end of the life cycle.
- We can override `jspdestroy()` method when we perform any cleanup such as releasing database connections or closing open files.

## JSP Elements

- JSP Declaration
- JSP Scriptlet
- JSP Expression
- JSP Comments
- Creating a simple JSP Page
- How to run simple JSP Page
- Directory Structure of JSP

## JSP Declaration

- A declaration tag is a piece of [Java](#) code for declaring variables, methods and classes. If we declare a variable or method inside declaration tag it means that the declaration is made inside the servlet class but outside the service method.
- We can declare a static member, an instance variable (can declare a number or string) and methods inside the declaration tag.

### Syntax of declaration tag:

```
<%! Dec var %>
```

Here Dec var is the method or a variable inside the declaration tag.

## JSP Scriptlet

- Scriptlet tag allows to write Java code into JSP file.
- JSP container moves statements in `_jspservice()` method while generating servlet from jsp.
- For each request of the client, service method of the JSP gets invoked hence the code inside the Scriptlet executes for every request.
- A Scriptlet contains java code that is executed every time JSP is invoked.

### Syntax of Scriptlet tag:

`<% java code %>`

Here `<%>` tags are scriptlets tag and within it, we can place java code.

### JSP Expression

- Expression tag evaluates the expression placed in it.
- It accesses the data stored in stored application.
- It allows create expressions like arithmetic and logical.
- It produces scriptless JSP page.

### Syntax:

`<%= expression %>`

Here the expression is the arithmetic or logical expression.

### Example:

In this example, we are using expression tag

1. `<%@ page language="java" contentType="text/html; charset=ISO-8859-1"`
2. `pageEncoding="ISO-8859-1"%>`
3. `<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"`  
`"http://www.w3.org/TR/html4/loose.dtd">`
4. `<html>`
5. `<head>`
6. `<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">`
7. `<title>Guru Expression</title>`
8. `</head>`
9. `<body>`
10. `<% out.println("The expression number is "); %>`
11. `<% int num1=10; int num2=10; int num3 = 20; %>`
12. `<%= num1*num2+num3 %>`
13. `</body>`
14. `</html>`

### JSP Comments

- Comments are the one when JSP container wants to ignore certain texts and statements.
- When we want to hide certain content, then we can add that to the comments section.

### Syntax:

`<% -- JSP Comments %>`

This tags are used to comment in JSP and ignored by the JSP container.

<!--comment -->

This is HTML comment which is ignored by browser

### Creating a simple JSP Page

- A JSP page has an HTML body incorporated with Java code into it
- We are creating a simple JSP page which includes declarations, scriptlets, expressions, comments tags in it.

#### Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4.     "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6.     <head>
7.         <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8.         <title>Guru JSP Example</title>
9.     </head>
10.     <body>
11.         <!-- This is a JSP example with scriptlets, comments , expressions --%>
12.         <% out.println("This is guru JSP Example"); %>
13.         <% out.println("The number is "); %>
14.         <%! int num12 = 12; int num32 = 12; %>
15.         <%= num12\*num32 %>
16.         Today's date: <%= (new java.util.Date()).toLocaleString()%>
17.     </body>
18. </html>

## JSP Directives

- JSP directives are the messages to JSP container. They provide global information about an entire JSP page.
- JSP directives are used to give special instruction to a container for translation of JSP to servlet code.
- In JSP life cycle phase, JSP has to be converted to a servlet which is the translation phase.
- They give instructions to the container on how to handle certain aspects of JSP processing
- Directives can have many attributes by comma separated as key-value pairs.
- In JSP, directive is described in `<%@ %>` tags.

### Syntax of Directive:

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

There are three types of directives:

1. Page directive
2. Include directive
3. Taglib directive

### JSP Page directive

#### Syntax of Page directive:

`<%@ page...%>`

- It provides attributes that get applied to entire JSP page.
- It defines page dependent attributes, such as scripting language, error page, and buffering requirements.
- It is used to provide instructions to a container that pertains to current JSP page.

Following are its list of attributes associated with page directive:

1. Language
2. Extends
3. Import
4. contentType
5. info
6. session
7. isThreadSafe
8. autoflush
9. buffer
10. isErrorPage
11. pageEncoding
12. errorPage
13. isELIgnored

More details about each attribute



1. **language:** It defines the programming language (underlying language) being used in the page.

**Syntax of language:**

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

Here value is the programming language (underlying language)

**Example:**

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

2. **Extends:** This attribute is used to extend (inherit) the class like JAVA does

**Syntax of extends:**

```
<%@ page extends="value" %>
```

Here the value represents class from which it has to be inherited.

**Example:**

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

```
<%@ page extends="demotest.DemoClass" %>
```

3. **Import:** This attribute is most used attribute in page directive attributes. It is used to tell the container to import other java classes, interfaces, enums, etc. while generating servlet code. It is similar to import statements in java classes, interfaces.

**Syntax of import:**

```
<%@ page import="value" %>
```

Here value indicates the classes which have to be imported.

**Example:**

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

4. **contentType:**

- It defines the character encoding scheme i.e. it is used to set the content type and the character set of the response
- The default type of contentType is "text/html; charset=ISO-8859-1".

### **Syntax of the contentType:**

```
<%@ page contentType="value" %>
```

### **Example:**

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

## **5. info**

- It defines a string which can be accessed by `getServletInfo()` method.
- This attribute is used to set the servlet description.

### **Syntax of info:**

```
<%@ page info="value" %>
```

Here, the value represents the servlet information.

### **Example:**

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

## **6. Session**

- JSP page creates session by default.
- Sometimes we don't need a session to be created in JSP, and hence, we can set this attribute to false in that case. The default value of the session attribute is true, and the session is created.

When it is set to false, then we can indicate the compiler to not create the session by default.

### **Syntax of session:**

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

Here in this case session attribute can be set to true or false

### **Example:**

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

## **7. isThreadSafe:**

- It defines the threading model for the generated servlet.
- It indicates the level of thread safety implemented in the page.

- Its default value is true so simultaneous
- We can use this attribute to implement SingleThreadModel interface in generated servlet.
- If we set it to false, then it will implement SingleThreadModel and can access any shared objects and can yield inconsistency.

#### **Syntax of isThreadSafe:**

```
<% @ page isThreadSafe="true/false" %>
```

Here true or false represents if synchronization is there then set as true and set it as false.

#### **Example:**

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

### **8. AutoFlush:**

- This attribute specifies that the buffered output should be flushed automatically or not and default value of that attribute is true.
- If the value is set to false the buffer will not be flushed automatically and if its full, we will get an exception.
- When the buffer is none then the false is illegitimate, and there is no buffering, so it will be flushed automatically.

#### **Syntax of autoFlush:**

```
<% @ page autoFlush="true/false" %>
```

Here true/false represents whether buffering has to be done or not

#### **Example:**

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

### **9. Buffer:**

- Using this attribute the output response object may be buffered.
- We can define the size of buffering to be done using this attribute and default size is 8KB.
- It directs the servlet to write the buffer before writing to the response object.

#### **Syntax of buffer:**

```
<%@ page buffer="value" %>
```

Here the value represents the size of the buffer which has to be defined. If there is no buffer, then we can write as none, and if we don't mention any value then the default is 8KB

**Example:**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    buffer="16KB"%>
```

**10. isErrorPage:**

- It indicates that JSP Page that has an errorPage will be checked in another JSP page
- Any JSP file declared with "isErrorPage" attribute is then capable to receive exceptions from other JSP pages which have error pages.
- Exceptions are available to these pages only.
- The default value is false.

**Syntax of isErrorPage:**

```
<%@ page isErrorPage="true/false"%>
```

**Example:**

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

**11. PageEncoding:**

- The "pageEncoding" attribute defines the character encoding for JSP page.
- The default is specified as "ISO-8859-1" if any other is not specified.

**Syntax of pageEncoding:**

```
<%@ page pageEncoding="value" %>
```

Here value specifies the charset value for JSP

**Example:**

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

**12. errorPage:**

This attribute is used to set the error page for the JSP page if JSP throws an exception and then it redirects to the exception page.

**Syntax of errorPage:**

```
<%@ page errorPage="value" %>
```

Here value represents the error JSP page value

**Example:**

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

**13. isELIgnored:**

- IsELIgnored is a flag attribute where we have to decide whether to ignore EL tags or not.
- Its datatype is java enum, and the default value is false hence EL is enabled by default.

**Syntax of isELIgnored:**

```
<%@ page isELIgnored="true/false" %>
```

Here, true/false represents the value of EL whether it should be ignored or not.

**Example:**

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

**JSP Include directive**

- JSP "include directive"( codeline 8 ) is used to include one file to the another file
- This included file can be HTML, JSP, text files, etc.
- It is also useful in creating templates with the user views and break the pages into header&footer and sidebar actions.
- It includes file during translation phase

**Syntax of include directive:**

```
<%@ include....%>
```

**Example:**

Directive\_jsp2.jsp (Main file)

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3.     <%@ include file="directive\_header\_jsp3.jsp" %>
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5.     "http://www.w3.org/TR/html4/loose.dtd">
6. <html>
7. <head>
8.     <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
9.     <title>Guru Directive JSP2</title>
10. </head>
11. <body>
12. <a>This is the main file</a>

12. </body>
13. </html>

Directive\_header\_jsp3.jsp (which is included in the main file)

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4.     "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7.     <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8.     </head>
9. <body>
10. <a>Header file : </a>
11. <%int count =1; count++;
12.     out.println(count);%> :
13. </body>
14. </html>

## JSP Taglib Directive

- JSP taglib directive is used to define the tag library with "taglib" as the prefix, which we can use in JSP.
- More detail will be covered in JSP Custom Tags section
- JSP taglib directive is used in the JSP pages using the JSP standard tag libraries
- It uses a set of custom tags, identifies the location of the library and provides means of identifying custom tags in JSP page.

### Syntax of taglib directive:

```
<%@ taglib uri="uri" prefix="value"%>
```

Here "uri" attribute is a unique identifier in tag library descriptor and "prefix" attribute is a tag name.

### Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3.     <%@ taglib prefix="gurutag" uri="http://java.sun.com/jsp/jstl/core" %>
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5.     "http://www.w3.org/TR/html4/loose.dtd">
6. <html>
7. <head>
8.     <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
9.     <title>Guru Directive JSP</title>
10.     <gurutag:hello/>

10. </head>
11. <body>
12. </body>
13. </html>

## JSP Implicit Objects

- JSP implicit objects are created during the translation phase of JSP to the servlet.
- These objects can be directly used in scriptlets that goes in the service method.
- They are created by the container automatically, and they can be accessed using objects.

There are 9 types of implicit objects available in the container:

1. out
2. request
3. response
4. config
5. application
6. session
7. pageContext
8. page
9. exception

### out

- Out is one of the implicit objects to write the data to the buffer and send output to the client in response
- Out object allows us to access the servlet's output stream
- Out is object of javax.servlet.jsp.jspWriter class
- While working with servlet, we need printwriter object

### Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
- "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Implicit Guru JSP1</title>
8. </head>
9. <body>
10. <% int num1=10;int num2=20;
11. out.println("num1 is " +num1);
12. out.println("num2 is "+num2);
13. %>

14. </body>
15. </html>

## Request

- The request object is an instance of `java.servlet.http.HttpServletRequest` and it is one of the argument of service method
- It will be created by container for every request.
- It will be used to request the information like parameter, header information , server name, etc.
- It uses `getParameter()` to access the request parameter.

## Example:

Implicit\_jsp2.jsp(form from which request is sent to guru.jsp)

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
- "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Implicit Guru form JSP2</title>
8. </head>
9. <body>
10. <form action="guru.jsp">
11. <input type="text" name="username">
12. <input type="submit" value="submit">
13. </form>
14. </body>
15. </html>

## Response

- "Response" is an instance of class which implements `HttpServletResponse` interface
- Container generates this object and passes to `_jspService()` method as parameter
- "Response object" will be created by the container for each request.
- It represents the response that can be given to the client
- The response implicit object is used to content type, add cookie and redirect to response page

## Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
- "http://www.w3.org/TR/html4/loose.dtd">
4. <html>



5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Implicit Guru JSP4</title>
8. </head>
9. <body>
10. <%response.setContentType("text/html"); %>
11. </body>
12. </html>

## Application

- Application object (code line 10) is an instance of javax.servlet.ServletContext and it is used to get the context information and attributes in JSP.
- Application object is created by container one per application, when the application gets deployed.
- Servletcontext object contains a set of methods which are used to interact with the servlet container. We can find information about the servlet container

## Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4.     "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8. <title>Guru Implicit JSP6</title>
9. </head>
10. <body>
11. <% application.getContextPath(); %>
12. </body>
13. </html>

## Session

- The session is holding "httpsession" object(code line 10).
- Session object is used to get, set and remove attributes to session scope and also used to get session information

## Example:

Implicit\_jsp7(attribute is set)

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4.     "http://www.w3.org/TR/html4/loose.dtd">
5. <html>

5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Implicit JSP</title>
8. </head>
9. <body>
10. <% session.setAttribute("user","GuruJSP"); %>
11. <a href="implicit\_jsp8.jsp">Click here to get user name</a>
12. </body>
13. </html>

#### Implicit\_jsp8.jsp (getAttribute)

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4.     "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8. <title>implicit Guru JSP</title>
9. </head>
10. <body>
11. <% String name = (String)session.getAttribute("user");
12.     out.println("User Name is " +name);
13. %>
14. </body>
15. </html>

#### pageContext:

- This object is of the type of pagecontext.
- It is used to get, set and remove the attributes from a particular scope

Scopes are of 4 types:

- Page
- Request
- Session
- Application

#### Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4.     "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8. <title>Example</title>
9. </head>
10. <body>
11. <% String name = (String)session.getAttribute("user");
12.     out.println("User Name is " +name);
13. %>
14. </body>
15. </html>

```

7. <title>Implicit Guru JSP9</title>
8. </head>
9. <body>
10. <% pageContext.setAttribute("student","gurustudent",pageContext.PAGE_SCOPE);
11. String name = (String)pageContext.getAttribute("student");
12. out.println("student name is " +name);
13. %>
14. </body>
15. </html>

```

## Page

- Page implicit variable holds the currently executed servlet object for the corresponding jsp.
- Acts as this object for current jsp page.

## Example:

In this example, we are using page object to get the page name using toString method

```

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Implicit Guru JSP10</title>
8. </head>
9. <body>
10. <% String pageName = page.toString();
11. out.println("Page Name is " +pageName);%>
12. </body>
13. </html>

```

## Exception

- Exception is the implicit object of the throwable class.
- It is used for exception handling in JSP.
- The exception object can be only used in error pages.

## Example:

```

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1" isErrorPage="true"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

```

7. <title>Implicit Guru JSP 11</title>
8. </head>
9. <body>
10. <%int[] num1={1,2,3,4};
11. out.println(num1[5]);%>
12. <%= exception %>
13. </body>
14. </html>

## JSP Expression Language (EL)

Expression Language (EL) is mechanism that simplifies the accessibility of the data stored in [Java](#) bean component and other object like request, session and application, etc. There are many operators in JSP that are used in EL like arithmetic and logical operators to perform an expression.

- JSP Syntax of Expression Language (EL)
- JSP If-else
- JSP Switch
- JSP For loop
- JSP While loop
- JSP Operators

## JSP Syntax of Expression Language (EL)

**Syntax of EL :**\$(expression)

- In JSP, whatever present in the braces gets evaluated at runtime sent to the output stream.
- The expression is a valid EL expression and it can be mixed with a static text and can be combined with other expression to form larger expression.

To get a better idea, on how expression works in JSP, we will see below example.

In this example, we will see how EL is used as an operator to add two numbers (1+2) and get the output respectively.

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2. pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4. "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8. <title>Guru JSP1</title>
9. </head>
10. <body>
11. <a>Expression is:</a>
12. \${1+2};
13. </body>

13. </html>

### **Flow Control Statements:**

JSP provides the power of Java to be embedded in the application. We can use all the APIs and building blocks of Java in JSP programming including control flow statements which include decision making and the loop statements.

There are two types of flow control statements described below;

1. Decision-Making statements
2. Loop Statements

### **Decision-Making Statements:**

Decision-making statement in JSP is based on whether the condition set is true or false. The statement will behave accordingly.

There are two types of decision-making statements described below:

1. If – else
2. switch

### **JSP If-else**

"If else" statement is basic of all control flow statements, and it tells the program to execute the certain section of code only if the particular test evaluates to true.

This condition is used to test for more than one condition whether they are true or false.

- If the first condition is true then "if block" is executed and
- if it is false then "else block" is executed

### **Syntax for if – else statement:**

```
If(test condition)
{
    //Block of statements
}
else
{
    //Block of statements
}
```

In this example,

We are going to test "if else" condition by taking variable and checking the value if the variable matches with what it is initialized:

```
1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru JSP2</title>
8. </head>
9. <body>
10. <%! int month=5; %>
11. <% if(month==2){ %>
12. <a>Its February</a>
13. <% }else{ %>
14. <a>Any month other than February</a>
15. <%} %>
16. </body>
17. </html>
```

## JSP Switch

The body of the switch statement is called as a "switch block".

- The switch case is used to check the number of possible execution paths.
- A switch can be used with all data types
- The switch statements contain more than one cases and one default case
- It evaluates the expression then executes all the statements following the matching case

### Syntax for switch statement:

```
switch (operator)
{
    Case 1:
        Block of statements
    break;
    Case 2:
        Block of statements
    break;
```

```
case n:
    Block of statements
break;
default:
    Block of statements
break;
```

}

- Switch block begins with one parameter, which is the operator that needs to be passed and
- Then there are different cases which provides condition and whichever matches with the operator that case is executed.

In below example,

We have defined a variable week, and it is matched with the case whichever is true. In this case, week is 2 hence 2<sup>nd</sup> case is matched, and the output is Tuesday:

```
1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru JSP3</title>
8. </head>
9. <body>
10. <%! int week=2; %>
11. <% switch(week){
12. case 0:
13.   out.println("Sunday");
14.   break;
15. case 1:
16.   out.println("Monday");
17.   break;
18. case 2:
19.   out.println("Tuesday");
20.   break;
21. case 3:
22.   out.println("wednesday");
23.   break;
24. case 4:
25.   out.println("Thursday");
26.   break;
27. case 5:
28.   out.println("Friday");
29.   break;
30. default:
31.   out.println("Saturday");
32. }
33. %>
34. </body>
35. </html>
```

## Loop Statements

### JSP For loop

It is used for iterating the elements for a certain condition, and it has three parameters.

- Variable counter is initialized
- Condition till the loop has to be executed
- Counter has to be incremented

#### For loop Syntax:

```
For(int i=0;i<n;i++)  
{  
    //block of statements  
}
```

In this Example,

We have for loop which iterates till counter is less than the given number:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
- "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru JSP4</title>
8. </head>
9. <body>
10. <%! int num=5; %>
11. <% out.println("Numbers are:");
12. for(int i=0;i<num;i++){
13.     out.println(i);
14. } %>
15. </body>
16. </html>

### JSP While loop

It is used to iterate the elements wherein it has one parameter of the condition.

#### Syntax:

```
While(i<n)  
{  
    //Block of statements  
}
```



In this example,

We have a while loop which will iterate till day is greater than equal to the counter:

```
1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru JSP5</title>
8. </head>
9. <body>
10. <%! int day=2; int i=1; %>
11. <% while(day>=i){
12.   if(day==i){
13.     out.println("Its Monday");
14.     break;}
15.   i++;}
16. %>
17.
18. </body>
19. </html>
```

## JSP Operators

JSP Operators supports most of its arithmetic and logical operators which are supported by java within expression language (EL) tags.

Frequently used operators are mentioned below:

### Following are the operators:

.	Access a bean property or Map entry
[]	Access an array or List element
( )	Group a subexpression to change the evaluation order
+	Addition
-	Subtraction or negation of a value
*	Multiplication
/ or div	Division
% or mod	Modulo (remainder)
== or eq	Test for equality
!= or ne	Test for inequality
< or lt	Test for less than
> or gt	Test for greater than

<= or le    Test for less than or equal  
 >= or ge    Test for greater than or equal  
 && or and   Test for logical AND  
 || or or     Test for logical OR  
 ! or not     Unary Boolean complement  
 Empty       Test for empty variable values

In this example,

- We are declaring two variables num1 and num2 and then take a variable num3, where we use JSP operator + by to add num1 and num2 and get num3.
- Then we check a condition whether num3 is not equal to 0 by using JSP operators (!= , >) and
- Then take another variable num4 by multiplying two num1 and num2 we get num4.

These all numbers should be printed out as our output:

```

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru JSP6</title>
8. </head>
9. <body>
10. <% int num1=10; int num2 = 50;
11.   int num3 = num1+num2;
12.   if(num3 != 0 || num3 > 0){
13.     int num4= num1*num2;
14.     out.println("Number 4 is " +num4);
15.     out.println("Number 3 is " +num3);
16.   }%>
17. </body>
18. </html>

```

### Try Catch: Exception Handling in JSP

- Exceptions occur when there is an error in the code either by the developer or internal error from the system.
- Exception handling in JSP is same as in java where we manage exceptions using try catch blocks.

Exceptions are of three types:

1. Checked Exception

2. RuntimeException
3. ErrorsException

## **Checked Exceptions**

It is normally a user error or problems which are not seen by the developer are termed as checked exceptions.

Some of the examples are:

1. FileNotFoundException: This is a checked exception (where it tries to find a file when the file is not found on the disk).
2. IO Exception: This is also checked exception if there is any exception occurred during reading or writing of a file then the IO exception is raised.
3. SQLException: This is also a checked exception when the file is connected with [SQL](#) database, and there is issue with the connectivity of the SQL database then SQLException is raised

## **Runtime Exceptions**

Runtime exceptions are the one which could have avoided by the programmer. They are ignored at the time of compilation.

Some of the examples are:

1. ArrayIndexOutOfBoundsException : This is a runtime exception when array size exceeds the elements.
2. ArithmeticException: This is also a runtime exception when there are any mathematical operations, which are not permitted under normal conditions, for example, dividing a number by 0 will give an exception.
3. NullPointerException: This is also a runtime exception which is raised when a variable or an object is null when we try to access the same. This is a very common exception.

## **Errors:**

The problem arises due to the control of the user or programmer. If stack overflows, then error can occur.

Some examples of the error are listed below:

1. Error: This error is a subclass of throwable which indicates serious problems that an application cannot catch.
2. Instantiation error: This error occurs when we try to instantiate an object, and it fails to do that.
3. Internal Error: This error occurs when there is an error occurred from JVM i.e. Java Virtual Machine.

## **Error Exception**

It is an instance of the throwable class, and it is used in error pages.

Some methods of throwable class are:

- Public String getMessage() – returns the message of the exception.
- Public Throwable getCause() – returns cause of the exception
- Public printStackTrace() – returns the stacktrace of the exception.

## JSTL

JSTL stands for [Java](#) server pages standard tag library, and it is a collection of custom JSP tag libraries that provide common web development functionality.

### Advantages of JSTL

1. **Standard Tag:** It provides a rich layer of the portable functionality of JSP pages. It's easy for a developer to understand the code.
2. **Code Neat and Clean:** As scriptlets confuse developer, the usage of JSTL makes the code neat and clean.
3. **Automatic JavaBeans Interospection Support:** It has an advantage of JSTL over JSP scriptlets. JSTL Expression language handles JavaBean code very easily. We don't need to downcast the objects, which has been retrieved as scoped attributes. Using JSP scriptlets code will be complicated, and JSTL has simplified that purpose.
4. **Easier for humans to read:** JSTL is based on XML, which is very similar to HTML. Hence, it is easy for the developers to understand.
5. **Easier for computers to understand:** Tools such as Dreamweaver and front page are generating more and more HTML code. HTML tools do a great job of formatting HTML code. The HTML code is mixed with the scriptlet code. As JSTL is expressed as XML compliant tags, it is easy for HTML generation to parse the JSTL code within the document.

## JSTL Core

The core tags are most frequently used tags in JSP. They provide support for

- Iteration
- Conditional logic
- Catch exception
- url forward
- Redirect, etc.

To use core tags we need to define tag library first and below is the syntax to include a tag library.

### Syntax :

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

Here,

- prefix can be used to define all the core tags and
- uri is the library of taglib from which it is imported

Let see some of the core tags in detail,

### 1. Out:

- Result of expression is displayed in the out tag
- It can directly escape the XML tags. Hence, they are not evaluated as actual tags

#### Syntax:

```
<c:out value="" default="" escapeXML="">
```

- Here value represents information to the output, and it is mandatory
- Default is failure to output information, and it is not mandatory
- escapeXML – It is true if it escapes XML characters.

#### Example:

Coretag\_jsp1.jsp

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
- 3.
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5.     "http://www.w3.org/TR/html4/loose.dtd">
6. <html>
7.     <head>
8.         <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
9.         <title>Core Tag JSP1</title>
10.     </head>
11.     <body>
- 12.
13.     </body>
14. </html>

### 2. Catch

- It catches any throwable exception which occurs in the body and shows as output.
- It is used for handling the errors and to catch them.

#### Syntax:

```
<c:catchvar="">
```

Here var represents the name of the variable, which will hold throwable exception.

### Example:

```
1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1"%>
3.   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4.   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
5.   <html>
6.   <head>
7.     <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8.     <title>Core Tag JSP2</title>
9.   </head>
10.  <body>
11.    <c:catch var="guruException">
12.      <% int num = 10/0; %>
13.    </c:catch>
14.    The Exception is : ${guruException}
15.  </body>
16. </html>
```

### 3. Import

- We can import another file contents into a JSP page like we did in JSP include action.
- Here we can also include URL and contents will be displayed on that page.

#### Syntax:

```
<c:importvar="" uri="">
```

Here var is a variable name which is an identifier, which will hold the filename/uri.

uri is relative filename or uriname.

coretag\_jsp31.jsp

```
1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.   pageEncoding="ISO-8859-1"%>
3.   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4.   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
5.   <html>
6.   <head>
7.     <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8.     <title>Core Tag JSP 31</title>
9.   </head>
10.  <body>
11.    <c:import var="displayfile" url="coretag_jsp32.jsp">
12.  </c:import>
13.  <c:out value="${displayfile}"/>
```

14. </body>
15. </html>

#### Coretag\_jsp32.jsp

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
- "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Insert title here</title>
8. </head>
9. <body>
10. <a>The file is displayed after importing</a>
11. </body>
12. </html>

#### 4. forEach

- It is used to iterate the number of elements in series of statements.
- It is same as a Java forloop.

#### Syntax:

<c:forEach var="" begin="" end="">

- Here var represents variable name which will hold counter name
- Begin represents counter begin value
- End will represent its end value

#### Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3.     <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
- "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8. <title>Core Tag JSP4</title>
9. </head>
10. <body>
11. <c:forEach var="gurucount" begin="5" end="10">
12.     <c:out value="{gurucount}"/>
13. </c:forEach>
14. </body>

15. </html>

## 5. If

- It is used for [Testing](#) conditions.
- If the tag is used to test a condition whether it is true or not based on this, the block of code would be executed.

### Syntax:

```
<c:if test="${condition}"></c:if>
```

Here if the condition is true then series of statements are executed.

Example:

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2.     pageEncoding="ISO-8859-1"%>
3.     <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
- "http://www.w3.org/TR/html4/loose.dtd">
5. <html>
6. <head>
7.     <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8.     <title>Core Tag JSP5</title>
9. </head>
10. <body>
11.     <c:set var="count" value="100"/>
12.     <c:if test="\${count == 100}">
13.         <c:out value="The count is 100"/>
14.     </c:if>
15. </body>
16. </html>

## 6. redirect:

- It is used for redirecting the current page to another URL by providing the relative URL of this tag.
- It supports context relative URLs

### Syntax:

```
<c:redirect url="" context=""/>
```

Here url is relative url to which it has to be redirected and context name of the local web application.

**Example:**



1. `<%@ page language="java" contentType="text/html; charset=ISO-8859-1"`
2. `pageEncoding="ISO-8859-1"%>`
3. `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
4. `<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"`  
`"http://www.w3.org/TR/html4/loose.dtd">`
5. `<html>`
6. `<head>`
7. `<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">`
8. `<title>Core Tag JSP6</title>`
9. `</head>`
10. `<body>`
11. `<c:redirect url="/" />`
12. `</body>`
13. `</html>`

## JSP Custom Tags

- It is a user-defined JSP language element.
- When JSP is translated into a servlet, custom tag is converted into a class which takes action on an object and is called as a tag handler.
- Those actions when the servlet is executed are invoked by the web container.
- To create the user-defined custom tag, we need to create the tag handler which will be extending the SimpleTagSupport and have to override doTag() method.
- We need to create TLD where we need to map the class file in TLD.

## Advantages of custom tags in JSP:

- **Portable** - An action described in a tag library must be usable in any JSP container.
- **Simple** - Unsophisticated users must be able to understand and use this mechanism. Vendors of JSP functionality must find it easy to make it available to users as actions.
- **Expressive** - The mechanism must support a wide range of actions, including nested actions, scripting elements inside action bodies, creation, use and updating of scripting variables.
- **Usable from different scripting languages** - Although the JSP specification currently only defines the semantics for scripts in the Java programming language, we want to leave open the possibility of other scripting languages.
- **Built upon existing concepts and machinery** - We do not want to reinvent what exists elsewhere. Also, we want to avoid future conflicts whenever we can predict them.

## Syntax:

Consider we are creating testGuru tag and we can use tag handler testTag class, which will override doTag() method.

```
<ex:testGuru/>
```

```
Class testTag extends SimpleTagSupport{ public void doTag()}
```

## JSP Form Processing Using `getParameter()`

### JSP Form Processing

Forms are the common method in web processing. We need to send information to the web server and that information.

There are two commonly used methods to send and get back information to the web server.

1. GET Method:

- This is the default method to pass information from browser to web server.
- It sends the encoded information separated by `?` character appended to URL page.
- It also has a size limitation, and we can only send 1024 characters in the request.
- We should avoid sending password and sensitive information through GET method.

2. POST Method:

- Post method is a most reliable method of sending information to the server.
- It sends information as separate message.
- It sends as text string after `?` in the URL.
- It is commonly used to send information which are sensitive.

JSP handles form data processing by using following methods:

1. `getParameter()`:

It is used to get the value of the form parameter.

2. `getParameterValues()`:

It is used to return the multiple values of the parameters.

3. `getParameterNames()`

It is used to get the names of parameters.

4. `getInputStream()`

It is used to read the binary data sent by the client.

### Example:

In this example, we have taken a form with two fields."username" and "password" with a submit button

Action\_form.jsp

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
- 2.
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru Form</title>
8. </head>
9. <body>
10. <form action="action\_form\_process.jsp" method="GET">
11. UserName: <input type="text" name="username">
12. <br />
13. Password: <input type="text" name="password" />
14. <input type="submit" value="Submit" />
15. </form>
16. </body>
17. </html>

#### Action\_form\_process.jsp

18. <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
19. <pageEncoding="ISO-8859-1"%>
20. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
21. <html>
22. <head>
23. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
24. <title>Insert title here</title>
25. </head>
26. <body>
- 27.
28. <h1>Form Processing</h1>
- 29.
30. <p><b>Welcome User:</b>
31. <%= request.getParameter("username")%>
32. </p>
- 33.
34. </body>
35. </html>

## JSP Current Date and Time

### JSP Date Handling

1. **Date()** – It gives us current date and time
2. **Date(long mililsec)** – This takes parameter of milliseconds which has been elapsed on January 1 1970
3. **Boolean after(Date date)** – It gives after date of the given date parameter
4. **Boolean before(Date date)** – It gives the before date of the given date parameter
5. **Object clone()** – It creates a copy of the date object
6. **IntcompareTo(Date date)** – it compares the date class object with another one
7. **IntcompareTo(Object date)** – it compares with the object class object with another one
8. **Boolean equals(Object date)** – it checks whether two date objects are equal
9. **Long getTime()** – it fetches the time
10. **InthashCode()** – it fetches the hashcode of the given date
11. **Void setTime(long Time)** – It sets the time of given date object
12. **String toString()** – It converts into string object of date object.

### Example:

In this example, we are fetching the current date and time using date object

1. `<%@ page language="java" contentType="text/html; charset=ISO-8859-1"`
2. `pageEncoding="ISO-8859-1"%>`
3. `<%@ page import="java.util.*" %>`
4. `<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"`  
`"http://www.w3.org/TR/html4/loose.dtd">`
5. `<html>`
6. `<head>`
7. `<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">`
8. `<title>Guru current Date</title>`
9. `</head>`
10. `<body>`
11. Today's date: `<%= (new java.util.Date()).toLocaleString()%>`
12. `</body>`
13. `</html>`

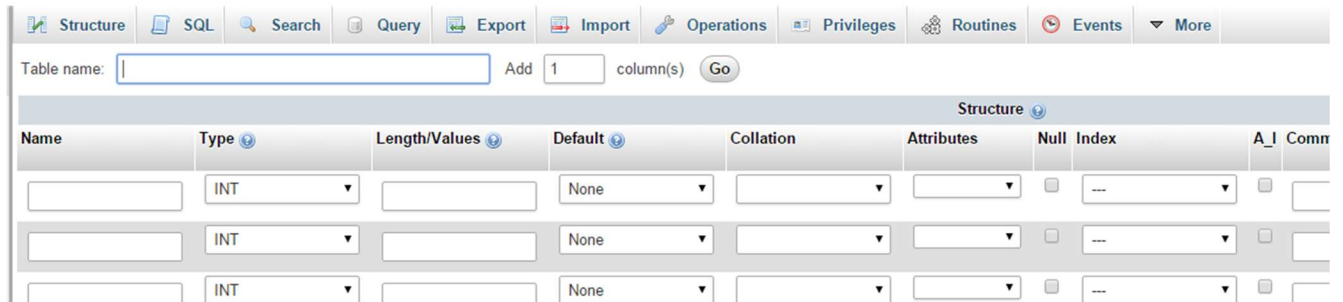
### JSP Database Connection: Select, Insert, Update & Delete Example

- The database is used for storing various types of data which are huge and has storing capacity in gigabytes. JSP can connect with such databases to create and manage the records.
- [Create Table](#)
- [Create Records](#)
- [JSP Operations: Insert, Update, Delete, Select](#)

## Create Table

In MYSQL database, we can create a table in the database with any MYSQL client.

Here we are using PHPMysqladminclient, and there we have an option "new" to create a new table using below screenshot.

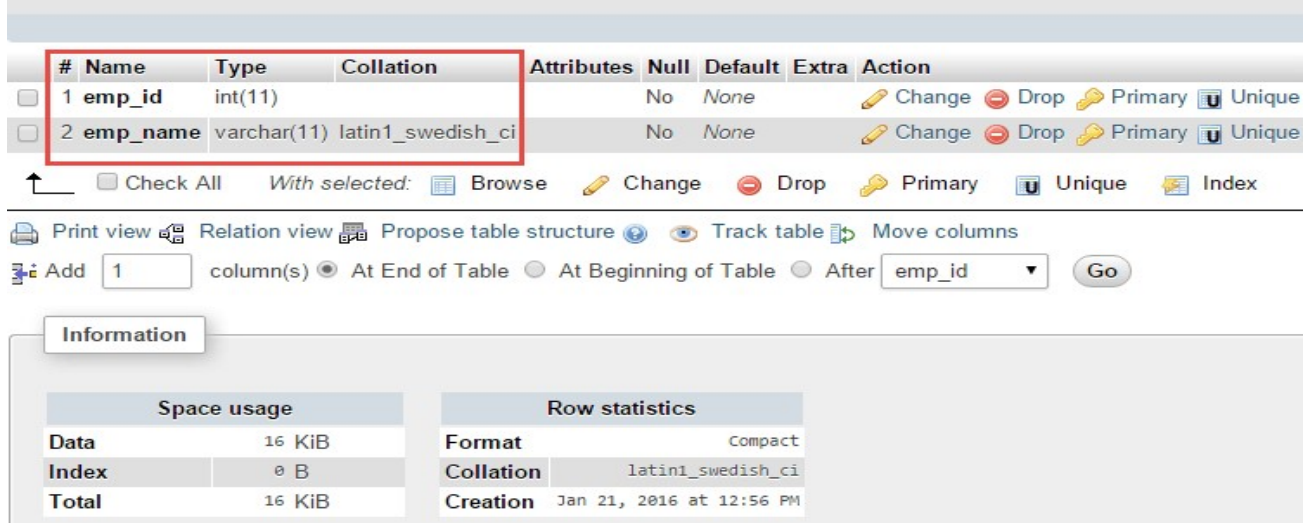


The screenshot shows the MySQL Structure tool interface. At the top, there is a 'Table name' field with a placeholder 'Add 1 column(s) Go'. Below this is a table with the following columns: Name, Type, Length/Values, Default, Collation, Attributes, Null, Index, A\_I, and Comm. The table is currently empty, with only the headers visible.

In this, we have to provide table name as guru\_test, and we will create two fields emp\_id and emp\_name.

Emp\_id is having datatype as int

Emp\_name is having datatype as varchar



The screenshot shows the MySQL Structure tool interface for a table named 'guru\_test'. The table structure is displayed in a table with the following columns: #, Name, Type, Collation, Attributes, Null, Default, Extra, and Action. The table has two columns: emp\_id (int(11)) and emp\_name (varchar(11) latin1\_swedish\_ci). The table is currently empty, with only the headers visible.

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	emp_id	int(11)		No	None			Change Drop Primary Unique
2	emp_name	varchar(11)	latin1_swedish_ci	No	None			Change Drop Primary Unique

Below the table structure, there are options to 'Check All', 'With selected', 'Browse', 'Change', 'Drop', 'Primary', 'Unique', and 'Index'. There are also links for 'Print view', 'Relation view', 'Propose table structure', 'Track table', and 'Move columns'. At the bottom, there is an 'Add' button with a dropdown menu showing '1 column(s)' and 'At End of Table', 'At Beginning of Table', and 'After emp\_id'. There is also a 'Go' button.

**Information**

Space usage		Row statistics	
Data	16 KiB	Format	Compact
Index	0 B	Collation	latin1_swedish_ci
Total	16 KiB	Creation	Jan 21, 2016 at 12:56 PM

Another option is by using command prompt and changes to MYSQL directory:

C:\>

C:\>cd Program Files\MY SQL\bin

C:\>Program Files\MySql\bin>

We can login to database as follows:

**C:\Program Files\MYSQL\bin>mysql -u gururoot -p**

**Enter Password: \*\*\*\*\***

**Mysql>**

Create table guru\_test in the database named as GuruTestas the following on MYSQL prompt:

```
Mysql> use GuruTest;
MySql> create table guru_test(
Emp_id int NOT NULL,
Emp_name varchar(11),
);
```

Once you execute this you get the following:

Query OK, 0 rows affected(0.10 sec)

```
MySQL> select * from guru_test;
```

Query OK, 0 rows affected(0.10 sec)

First the records are inserted using **INSERT** query and then we can use **SELECT** query to check whether the table is created or not.

### **Create Records**

After creating a table we need to create records into the guru\_test table using insert query, which is shown below:

The records entered here are:

- 1 and guru emp1
- 2 and guru emp2

```
MySQL>INSERT INTO `couch_tomato_db`.`guru_test` (`emp_id`, `emp_name`) VALUES ('1', 'guru emp1');
```

Query OK, 1 row affected (0.05 sec)

```
MySQL>INSERT INTO `couch_tomato_db`.`guru_test` (`emp_id`, `emp_name`) VALUES ('2', 'guru emp2');
```

Query OK, 1 row affected (0.05 sec)

### **JSP Operations: Insert, Update, Delete, Select**

Using JSP, we can do multiple operations into the database. We can insert the records, and also, we can delete the records which are not required. If any record needs to be edited, then we can do using an update. The Select operation will help to fetch the records which are required.

#### **Select**

The Select operation is used to select the records from the table.

### Example:

In this example, we are going to learn about the select operation of fetching records from guru\_test table which was created in the above section.

```
1. <%@ page import="java.io.*,java.util.*,java.sql.*"%>
2. <%@ page import="javax.servlet.http.*,javax.servlet.*" %>
3. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
4. <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
5. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
6.     pageEncoding="ISO-8859-1"%>
7. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
8. <html>
9. <head>
10. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
11. <title>Guru Database JSP1</title>
12. </head>
13. <body>
14.
15. <sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
16.     url="jdbc:mysql://localhost/GuruTest"
17.     user="gururoot" password="guru"/>
18.
19. <sql:query dataSource="${snapshot}" var="result">
20. SELECT * from guru_test;
21. </sql:query>
22.
23. <table>
24. <tr>
25.     <th>Guru ID</th>
26.     <th>Name</th>
27.
28. </tr>
29. <c:forEach var="row" items="${result.rows}">
30. <tr>
31.     <td><c:out value="${row.emp_id}"/></td>
32.     <td><c:out value="${row.emp_name}"/></td>
33.
34. </tr>
35. </c:forEach>
36. </table>
37.
38. </body>
39. </html>
```

### Insert

Insert operator is used to insert the records into the database.

### Example:

In this example, we are going to learn about inserting the records in the table guru\_test

```
1. <%@ page import="java.io.*,java.util.*,java.sql.*"%>
2. <%@ page import="javax.servlet.http.*,javax.servlet.*" %>
3. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="gurucore"%>
4. <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="gurusql"%>
5. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
6.   pageEncoding="ISO-8859-1"%>
7. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
8. <html>
9. <head>
10. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
11. <title>Guru Database JSP1</title>
12. </head>
13. <body>
14.
15. <gurusql:setDataSource var="guru" driver="com.mysql.jdbc.Driver"
16.   url="jdbc:mysql://localhost/GuruTest"
17.   user="gururoot" password="guru"/>
18.
19. <gurusql:update dataSource="{guru}" var="guruvar">
20. INSERT INTO guru_test VALUES (3, 'emp emp3');
21. </gurusql:update>
22.
23.
24.
25. </body>
26. </html>
```

### Delete

This is delete operation where we delete the records from the table guru\_test.

### Example:

Here we will delete query to delete the record from the table guru\_test. The record which has to be deleted has to be set in variable "guruid", and the corresponding record is deleted from the database.

```
1. <%@ page import="java.io.*,java.util.*,java.sql.*"%>
2. <%@ page import="javax.servlet.http.*,javax.servlet.*" %>
3. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="gurucore"%>
4. <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="gurusql"%>
5. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
6.   pageEncoding="ISO-8859-1"%>
```



7. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
8. <html>
9. <head>
10. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
11. <title>Guru Database JSP1</title>
12. </head>
13. <body>
- 14.
15. <gurusql:setDataSource var="guru" driver="com.mysql.jdbc.Driver"
16. url="jdbc:mysql://localhost/GuruTest"
17. user="gururoot" password="guru"/>
18. <gurucore:set var="guruid" value="3"/>
19. <gurusql:update dataSource="{guru}" var="guruvar">
20. DELETE FROM guru\_test WHERE emp\_id = ?
21. <gurusql:param value="{guruid}" />
22. </gurusql:update>
- 23.
- 24.
- 25.
26. </body>
27. </html>

## Update

The update is used to edit the records in the table.

### Example:

1. <%@ page import="java.io.\*,java.util.\*,java.sql.\*"%>
2. <%@ page import="javax.servlet.http.\*,javax.servlet.\*" %>
3. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="gurucore"%>
4. <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="gurusql"%>
5. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
6. pageEncoding="ISO-8859-1"%>
7. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
8. <html>
9. <head>
10. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
11. <title>Guru Database JSP1</title>
12. </head>
13. <body>
- 14.
15. <gurusql:setDataSource var="guru" driver="com.mysql.jdbc.Driver"
16. url="jdbc:mysql://localhost/GuruTest"
17. user="gururoot" password="guru"/>
18. <gurucore:set var="guruid" value="2"/>
19. <gurusql:update dataSource="{guru}" var="guruvar">

20. UPDATE guru\_test SET emp\_name='emp guru99'  
21. <gurusql:param value="{guruid}" />  
22. </gurusql:update>  
23.  
24.  
25.  
26. </body>  
27. </html>  
28.