UNIT: 3 JSP, JAVA BEANS

CS-25 Advanced Java Programming (J2EE)

- INTRODUCTION TO JSP AND JSP BASICS
- JSP VS. SERVLET
- JSP ARCHITECTURE
- LIFE CYCLE OF JSP
- JSP ELEMENTS: DIRECTIVE ELEMENTS, SCRIPTING
- ELEMENTS, ACTION ELEMENTS
- DIRECTIVES ELEMENTS (PAGE, INCLUDE, TAGLIB)
- SCRIPTING ELEMENTS (DECLARATION, SCRIPTLET, EXPRESSION)
- ACTION ELEMENTS (JSP: PARAM, JSP: INCLUDE, JSP: FORWARD, JSP: PLUGIN)
- JSP IMPLICIT OBJECTS
- JSP SCOPE
- INCLUDING AND FORWARDING FROM JSP PAGES
- INCLUDE ACTION
- FORWARD ACTION
- WORKING WITH SESSION & COOKIE IN JSP
- ERROR HANDLING AND EXCEPTION HANDLING WITH JSP
- JDBC WITH JSP
- JAVABEAN PROPERTIES
- JAVABEAN METHODS
- COMMON JAVABEAN PACKAGING

: ASSIGNMENT 3:

1. Give the full form of JSP
2. ErrorPage attributes of page directive are used to
3. List out scope of JSP object
4. A bean consist of _____ and _____
5. Explain any two attributes of page directive
6. Discuss any three implicit object of JSP
7. Explain all scripting elements of JSP with example segment.
8. Write down syntax of <JSP:forword>
9. The object can be accessed from any pages has _____ scope.
10. The accessor and mutator methods are also known as _____
11. What is templet data in JSP ?
12. Differentiate JSP v/s servlet
13. Explain JSP life cycle in details with diagram.
14. ___directive is used to include the static page and dynamic pages with the other JSP pages.
15. Explain JSP implicit objects.
16. Explain JSP architecture.
17. Demonstrate JDBC with JSP to view below table data in JSP page.

Database Name	:	Varmora_tiles
Table Name	:	product_master
Fields	:	Prod_id
		Prod_name
		Size
		Description
		Price

Introduction to JSP

JSP is Java Server Pages (JSP) technology enables you to mix regular, static HTML with dynamically generated content. JSP are run in a server-side component known as JSP container which translates them into equivalent Java Servlet. You simply write the regular HTML in the normal manner, using familiar Web-page-building tools. You then enclose the code for the dynamic parts in special tags, most of which start with <% and end with %>. JSP typically comprised of:

- Static HTML/XML components
- Special JSP tags.
- Code written in the java language called" script less".

Benefits of JSP

- **Nobody can borrow the code**

The JSP code written and runs and remains on the server. So, issue of copy source code does not arise at all.

All of JSP's functionality is handled before the page is sent to browser.

- **Faster loading of pages.**

With JSP, decision can be made about what user want to see at web server prior to pages being dispatched. So only the content that the user is interested will be dispatched to the user. There is no extra code and extra content.

- **No browser compatibility Issues.**

JSP pages can run same way in browser. The developer ends up sending standard HTML to a user browser. This largely eliminates scripting issues and cross browser compatibility.

- **JSP support**

JSP is supported by number of web server like Apache, Microsoft IIS and PWS, Netscape's Fast Tracks and Enterprise web server and others. Built in support for JSP is available Java Server from Sun Microsystem.

- **Compilation**

JSP compiled before the web server processes it. This allows the server to handle JSP pages much faster, because in the older technologies such as CGI require the server to load an interpreter and the target script each time the page is requested.

- **JSP elements in HTML/XML pages**

JSP page look like HTML /XML page, it holds text marked with a collection of tags. While a regular JSP page is not a valid XML page, there is a variant JSP tag syntax that lets the developer use JSP tags within XML documents.

Disadvantages of JSP

Attractive Java Code

Putting java code within web page is really bad design, but JSP makes it tempting to do just that. Avoid this as far as possible. It is done using template using.

Java Code required

To relatively simple things in JSP can actually demand putting java code in a page.

Simple task are hard to code

Even including page headers and footers is a bit difficult with JSP.

Difficult looping in JSP

In regular JSP pages looping is difficult. In advance JSP we can use some custom tags for looping.

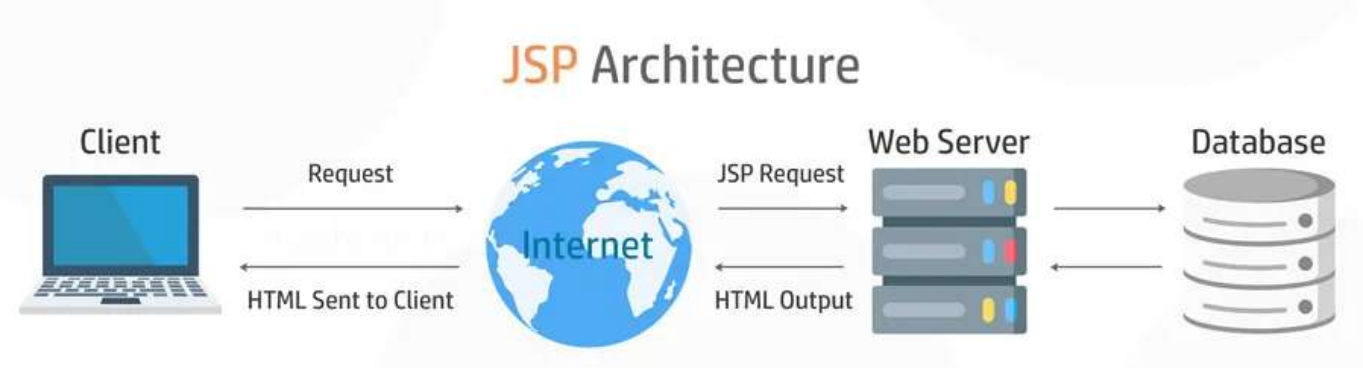
Occupies a lot of space.
JSP consumes extra hard drive and memory space.

Servlet v/s JSP

Servlet	JSP
Servlets run faster than JSP.	JSP runs slower than servlet as it takes time to compile the program and convert into servlets.
It is hard to write code in servlet.	It's easier to code in JSP compared to servlets.
In MVC architecture, servlet works as a controller.	In MVC architecture, JSP works as a view for displaying output.
It should be use when there is more data processing involved.	JSP is generally used when there is no involvement of much data processing.
There is no custom tag writing facility in servlets.	You can easily build custom tags that can directly call Java beans.
Servlet is a java code.	JSP is a HTML-based code.
It can accept all protocol requests, including HTTP.	It can only accept HTTP requests.
You can override the service() method.	In JSP, you can't override the service() method.
In Servlet, by default, session management is not enabled, user has to enable it explicitly.	In JSP, session management is automatically enabled.
In Servlet, you have to implement both business logic and presentation logic in the single file.	In JSP, business logic is split from presentation logic using JavaBeans.
Modification in Servlet file is a time consuming due to reloading, recompiling, and restarting the server.	JSP modification is fast, as you just need to click one refresh button.

JSP Architecture

The JSP architecture is a 3-tier architecture where each part has its own roles and functionalities. This chapter describes the JSP architecture and how the webserver processes JSP pages. The JSP-based web application needs a JSP engine, which is a Web container. A Web container can be defined as a web server component that helps in executing Web programs like servlet, ASP, etc. The JSP container intercepts the JSP page requests.



JSP Architecture Flow

1. When a user navigates to a page ending with a .JSP extension in their web browser, the web browser sends an HTTP request to the webserver.
2. The webserver checks if a compiled version of the JSP page already exists.
3. If the JSP page's compiled version does not exist, the file is sent to the JSP Servlet engine, it converts it into servlet content (with .java extension). This process is known as translation.
4. Then after translated .java file of the servlet is compiled into the Java servlet .class file. This process is known as compilation.
5. In the last step, the compiled .class file of the servlet is executed, and the result (HTML) is sent back to the client machine as an HTTP response.

JSP Life Cycle

A JSP page life cycle is defined as a process from its translation phase to the destruction phase. This lesson describes the various stages of a JSP page life cycle.

The life cycle of a JSP page can be divided into the following phase:

- **Translation Phase**
  - When a user navigates to a page ending with a .JSP extension in their web browser, the web browser sends an HTTP request to the webserver.
  - The webserver checks if a compiled version of the JSP page already exists.
  - If the JSP page's compiled version does not exist, the file is sent to the JSP Servlet engine, which converts it into servlet content (with .java extension). This process is known as translation.
  - The web container automatically translates the JSP page into a servlet. So as a developer, you don't have to worry about converting it manually.
- **Compilation Phase**
  - In case the JSP page was not compiled previously or at any point in time, then the JSP page is compiled by the JSP engine.
  - In this compilation phase, the Translated .java file of the servlet is compiled into the Java servlet .class file.
- **Initialization Phase**
  - Load the equivalent servlet class.

  - Create instance.
  - Call the JSPInit() method for initializing the servlet instance.
  - This initialization is done only once with the servlet's init method where database connection, opening files, and creating lookup tables are done.
- **Execution Phase**
  - This Execution phase is responsible for all JSP interactions and logic executions for all requests until the JSP gets destroyed. As the requested JSP page is loaded and initiated, the JSP engine has to invoke the _JSPService() method. This method is invoked as per the requests, responsible for generating responses for the associated requests, and responsible for all HTTP methods.
- **Destruction (Clean-up) Phase**
  - This destruction phase is invoked when the JSP has to be cleaned up from use by the container. The JSPDestroy() method is invoked. You can incorporate and write clean-up codes inside this method for releasing database connections or closing any file.

- **JSPInit()**

When JSP servlet instane is created, JSPInit() called .You can use servlteContext object or ServletConfig object to get initial parameters.It is similar to init() of servlet.

Syntax of JSPInit() is as follows:
public void JSPInit()

- **_JSPService()**

This is similar to service() of servlet .when JSP servlet instance is called ._JSPService() is called where request and response object are sent.

Syntax of JSPService() is as follows:

public void _JSPService(HttpServletRequest req,HttpServletResponse res)throws Servlet Exception,IOException

- **JSPDestroy()**

This is similar to destroy() of servlet .This method is called when the JSP servlet instance is destroyed from the web container.
Syntax of JSPInit() is as follows:
pubic void destroy()

Request-response cycle of JSP

1. Client sends an HTTP request for a JSP page to the server, either using a GET /POST method.
2. the browser identifies the JSP files and compiles and loads the corresponding servlet class if the servlet class is not already present in the JVM. The server calls the service() which often passes the request and response to the doGet() or doPost(). The servlet processes the input form the request and prepares an HTML page as a response.
3. The response is sent back to the client browser.

JSP Elements

JSP elements are instruction to JSP container about what code to generate and how it should operate. JSP elements have a special identity to JSP compiler because it starts and ends with special kind of tags. Template data (HTML) code is not compiled by the JSP compiler and also not recognized by the JSP container.it is also known as Component of the JSP pages. There are basically three type of JSP elements are as given follow.

1. **Directive Elements**
2. **Scripting Elements**
3. **Action Elements**

Directive Elements

The role of directive is to pass information to the JSP container.
Syntax: <%@ directive {attribute name="value"} %>
There are three type of directive elements are as given below.

- **Page Directive**
- **Include Directive**
- **Taglib Directive**

Page Directive

Page Directive is uses to specify attributes for the whole JSP page.
syntax is as follows:
<%@ page [attribute1="value1" attribute2="value2"……attributeN= "valueN"] %>

Where the attribute can be as follows:

	if it is true jsp page handles the all the request simultaneously from multiple threads if false then generates servlet declares that it implements the SingleThreadModel.	
Info	It returns string message information related to jsp page using getServletInfo()	-
isErrorPage	It defines the Boolean indicating if it is true jsp page considered as error page and if false then it is normal page and implements it with other jsp page to handle exception.	False
errorPage	It defines the url of the error page we want to implement the error page with the other jsp pages we can use this directive.	
contentType	It specifies the MIME type and character encoding which used with generated servlet.	<%@ page contentType="text/html"%>
pageEncoding	It defines the character encoding of the jsp page	ISO 8859 -1 (Latin script)for JSP style and UTF-8 for xml style tags.

Attribute Name	Use(description)	Default value
Language	It is used to define the language which is used with the scriptlet elements.Most probably its valid value is java only.	Java
extends	It defines fully qualified name of the super class of the jsp page.	
Import	It defines list of packages which are imported with the JSP page with its fully qualified name.	Javax.servlet.*, Javax.servlet.http.*, Javax.servlet.jsp.*,java.lang.*
Session	It defines the Boolean value indicating if jsp page require HTTP session then its value Is true else it becomes false.	True
Buffer	It specifies the size of buffer.	8 kb
autoFlush	It defines the Boolean indicating if it is true then it automatically flushed the buffer ad if false then it throws an exception buffer overflow.	True
isThreadSafe	It defines the Boolean indicating	True

Syntax for all JSP directives:

```
<%@page [language=”java”]
[extends=”package.class”]
[import=”{package.class|package.*},…..”
[session=”true/false”]
[buffer=”none|8kb|sizekb”]
[autoFlush=”true|false”]
[isThreadSafe=”true/false”]
[info=”text”]
[errorpage=”relativeURL”]
[contentType=”MIME[;charset=charset]” | ”text/html”;
Charset=ISO 8859-1”]
[isErrorPage=”true|false”]%>
```

Example:

```
<%@page import="java.util.*;" %>
<html>
 <head>

 <title> Page Directive Example </title>
 </head>
 <body>
 <%
 Date d1=new Date();
 out.println("curent date is=");
 out.println(d1);
 %>
 </body>
</html>
```

Include Directive:

Include directive is used to include the static page and dynamic pages with the other JSP pages. For example, if we want to Set same header and footer for all the pages, we can create header.html and can include it with necessary JSP pages. In short it is used to insert a part of the code that is common to multiple pages.

Syntax:<%@include file=” relative path” %>

The file attribute is used to specify the name of the file to be included.

First.html

```
<html>
 <head><title>Include example</title></head>
 <body>
Username: <input type="text" name="txtnm" value="">
Password: <input type="text" name="txtps" value="">
 <input type="submit" name="submit" value="submit">
 </body>
</html>
```

IncludeDemo.JSP

```
<%@include file="First.html" %>
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>JSP Page<./title>
 </head>
 <body>
 <h1>welcome to include directive example</h1>
 </body>
</html>
```

Example for include one JSP page to another JSP page:

IncrementDemo.JSP

```
<html>
 <head>
 <title>JSP Page</title>
 </head>
 <body>
 <% for (int sctr=0; ctr<=5; ctr ++)
 {%>
 The value of ctr is <%= ctr% ><br/>
 <%}%>
 </body>
</html>
```

Taglib Directive:

It declares that the JSP file uses custom tags, names the tag library that defines them, and specifies their tag prefix.

Syntax:

```
<%@ Taglib uri="URItoTagLibrary" prefix="tagPrefix" %>

<%@ taglib uri="/tlds/Fancy/TableGenerator.tld" prefix="fancytable" %>
< fancytable: table>
……..
</fancytable:table>
```

Scripting Elements

Scripting Elements are used to write java code with the JSP file. As you know in JSP java code is embedded within HTML code. You need to write some java language statements or use java features within the JSP page.

There are 3 types of scripting elements which are as follows:

- Scriptlet
- Declaration
- Expression

Scriptlet:
Scriptlets are block of java code with the JSP page. Scriptlets starts with <% tag and end with %>closing tag. JSP converted into servlet code and then JSP engine adds all processing statements of JSP processes them under _JSPService().

Syntax:
```
<%
 Statement 1;
 Statement 2;
 ……..
 Statement n;
 %>
```
We can write any valid java statements with the scriptlets declaring variable, writing processing and output part.

Declaration
Declaration tags are used to declare the variables, methods and instance of the classes with the JSP page scriptlets code become part of the _JSPService() whereas declaration code is incorporated into generate d source file outside the _JSPService().Declaration starts with <%! tag and end with %>closing tag.

```
<%!
Statement 1;
 Statement 2;
 ……..
 Statement n;
 %>
```

Expression:
Expression element is used to print value of any variable or any valid expression when the JSP page is requested. All the expressions are printed automatically by converting values into string values. If the result cannot be converted into a string, an error will be raised at translation time. An expression starts with <% = and ends with %>

Syntax:
<%= expressions %>
Example:
Current Time :< %= java.util.Calendar.getInstance().getTime() %>

This expression will be translated into the following statements in the _JSPService() of the generated servlet.

Action Elements:
Action elements are high level JSP elements which are used to create, modify and use other objects.Syntax of action element's tags just like XML syntax.

- **<JSP:param>**
- **<JSP:include>**
- **<JSP:forward>**
- **<JSP:plugin>**

1. **<JSP:param>**

This element Is used to provide the tag/value pair of information, by including these as sub attribute of the <JSP:include>,<JSP:forward> and the <JSP:plugin> actions.

Syntax:
<JSP:param name=”pname” value=”pvalue”/>
OR
<JSP:param name=”pname” value=”pvalue”>
</JSP: param>

2. **<JSP: include>**

This element is used to include static and dynamic resource of the current JSP page.
This object is just used to include resource on the current JSP page.It can not be used to send response.

Syntax:
<JSP: include page=”JSP page” flush=”true/false”/>
<JSP: param name=”pname” value=”pvalue”>
</JSP: include>

Example:
<JSP:include page=”login.JSP” flush=”true”/>
<JSP: param name=”username” value=”java”>
</JSP:include>

Here <JSP: include> has two attribute which are:
Page: Specifies the resource (JSP/HTML) file which will be included.
Flush: An optional parameter used to flush buffer. Specified true/false.

Note: Once the buffer is flushed, the value can not be recalled as it is flushed from the memory directly.

3. <JSP:forward>:

This control is used to transfer control from current JSP page to another source which may be any valid source of application. It is obvious to understand that whenever this action elements is called, execution of current JSP page is stopped and control is transferred to another specified URL into <JSP:forward>:

Note:<JSP:forward> action element is same as forward() of request dispatcher of servlet programming.

Syntax:
<JSP:forward page=”destinationpage”/>
Example:
<JSP:forward page=”abc.JSP”/>
Above example will transfer control from current page to abc.JSP

4. **<JSP:plugin>**

This element is used to embed an applet and java beans with the JSP page.the tag automatically detects the bowser type and inserts the appropriate HTML tag either<embed> or <object> in the output.

Syntax:
<JSP:plugin type=”plugintype” code=”classfilename” codebase=”url”>
</JSP:plugin>
Example:
<JSP:plugin type=”applet ” code=”AppletDemo.class” >
</JSP:plugin>

Comments and template data

We can use two type of comment with JSP page one is for HTML and another for JSP.
HTML comment :<! --this comment will appear in the client’s browser -->
JSP comment: <% -- this comment will not appear in the client’s browser --%>

JSP comment will not appear in the page output to the client.

Template Data:
In JSP page everything that is not a directive, declaration, scriptlet, expression, action elements or JSP comments is termed as Template Data. Usually, all the HTML and text in the page. In other words, template data is ignored by the JSP translator. This data Is output to the client as if it had appeared within a static web page.

Scope of the JSP objects
Objects that are created as part of the JSP page have a certain lifetime and may or may not be accessible to other components or objects in the web application. The lifetime and accessibility of an object is known as scope. There are four valid scope.

Page Scope
- Request Scope
- Session Scope
- Application Scope
- Object are accessible from page that belongs that
- That belongs to same page
- Least restrictive & most visible
- Object are accessible from page that belongs that That belongs to same session
- Object are accessible from page processing the most restrictive & least visible request where they were created.
- Object are accessible only within the page that where they were created.

Page scope:
This scope is most restrictive. With page scope, the object is accessible only within the current JSP page in which it is defined. JavaBeans created with page scope and objects created by Scriptlet are thread safe. JSP implicit objects out, exception, response, config, page Context and page have page context.

Request scope:
JSP object created using the ’request’ scope can be accessed from any pages that serve that request. This means that the object is available within the page in which it is created and within pages to which

the request is forwarded or included Object with request scope are thread safe. Only the execution thread for a particular request are can access these objects. Implicit object request has 'request scope'

session scope:
Objects with session scope are available to all application component that participate in the client's session. These objects are not thread safe. If multiple requests could use same session object at the same time, you must synchronize access to that object. JSP object that is created using the session scope is bound to the session object. Implicit object session has the 'session' scope.

Application Scope:
Objects with application scope are available to the entire application for the life of the application. These objects are not thread safe. And access to them must be synchronized if there is a chance that multiple requests will attempt to change the object at the same time's object is bound to the application object. Implicit object application has the 'application scope'.

Implicit object of JSP
Implicit objects mean the object which is created already created by JSP itself. Implicit objects are automatically created in JSP pages and can be used without declaring their object. The reason behind that implicit object is, JSP page is also converted into servlet at last, so some of objects should be provided by JSP also for programming similar to servlet. Implicit object are used within scriptlet and expression elements.

- Request
- Response
- Out
- Session
- Config
- Exception
- Application

Request:
This is the most important object which is used in JSP programming. The entire request coming from the client can be obtained from this object. Similar to servlet. programming, request object in JSP also belongs to javax.servlet.httpServletRequest class .You can see all the parameters sent by client through the request object. The request object has request scope. That means that the implicit request object can is in scope until the response to the client is complete.

Example:
```
<%
String u=request.getParameter("txt_username");
String p=request.getParameter("txt_password");
Out.println("welcome,"+u);
%>
```

The above example is used to retrieve username and password from request object and then print it.

Response:
This is another most important object which is used in JSP programming. If you want to generate any output for client, the you can generate it using response object.As similar to servlet programming

,response object in JSP is belongs to javax.servlet.HttpServletResponse class. This object allows us to set content type using setContentType (), set headser using addHeader (String name, String value), to set cookie using addCookie (Cookie cookie) for the client and send a redirect response to the client using sendRedirect () of response.

Session:

This object is used for session tracking. This belongs to javax.servlet.http.HttpSession class.session object Is for requesting the client is created under sessionimplicit object.we can use session method like getAttribute(), setAttribute(), getValue(),putValue(), removeAttribute(), removeValue(), isNew(), getId(), getCreationTime().

Application

This object is used for set values and attributes at application level.Application object of JSP is similar to ServletContext object of servlet programming getServletConfig (), getServletContext () can be used to retrieve application object. As it is similar servletContext object of application object.you cann use all methods available under ServletContext object. getAttribute(), setAttribute(), removeAttribute(), getServletInfo(), getInitParameter(), getInitParameterNames().

Out:

This object is used to write output to the output stream of the client.The scope of out is current page.out obect is created using javax.sevlet.JSP.JSPWriter class.methods of out object is print() , println(), clear(), clearBuffer(), close(), flush().

Config:

This object is similiar to ServeltConfig object of servlet class.config object is created using javax.Servlet.ServletConfig class.this is mainly used to read some initial parameters which are passed to particular page.Method available inside config object are getInitParameter(), getInitParameterNames(), getServletContext(),getServletName().

Page:

As you know JSP page is converted into servlet class at last. Then the servlet instance is created of that perticuar sevlet class. Page is the instance of the instance of the JSP servlet class created by web container for the current request. As page is the instance of the current JSP sevlet class; it contains method of object class along with all the methods created inside JSP page.

Implicit objects	Super class	Description
Request	HttpServletRequest	Provides HTTP request information
Response	HttpServletResponse	Send back data to the client
Out	JspWriter	Write data to the response stream
Session	HttpSession	Track information about a user from one request to another.
Application	ServletContext	Data shared by all JSP and servlets in the application.
pageContext	Pagecontext	Contains data associated with the whole page.
Config	ServletConfig	Provides servlet configuration data.
Page	Objects	Similar with “this” object in java
Exception	Throwable	Exception not caught by appliacation code

Working with Session & Cookie in JSP

- Cookies are the text files which are stored on the client machine.
- They are used to track the information for various purposes.
- It supports HTTP cookies using servlet technology
- The cookies are set in the HTTP Header.
- If the browser is configured to store cookies, it will keep information until expiry date.

JSP cookies methods

Public void setDomain(String domain): This JSP set cookie is used to set the domain to which the cookie applies

public String getDomain(): This JSP get cookie is used to get the domain to which cookie applies

public void setMaxAge(int expiry): It sets the maximum time which should apply till the cookie expires

public intgetMaxAge(): It returns the maximum age of cookie in JSP

public String getName(): It returns the name of the cookie

public void setValue(String value): Sets the value associated with the cookie

public String getValue(): Get the value associated with the cookie

public void setPath(String path): This set cookie in JSP sets the path to which cookie applies

public String getPath(): It gets the path to which the cookie applies

public void setSecure(Boolean flag): It should be sent over encrypted connections or not.

public void setComment(String cmt): It describes the cookie purpose

public String getComment(): It the returns the cookie comments which has been described.

How to Handle Cookies in JSP

1. Creating the cookie object
2. Setting the maximum age
3. Sending the cookie in HTTP response headers

Example:

Action_cookie.JSP.

```
<%@ page language="java" contentType="text/html %>
<html>
<head>
<title>Cookie</title>
</head>
<body>
<form action="action_cookie_main.JSP" method="GET">
Username: <input type="text" name="username">
<br />
Email: <input type="text" name="email" />
```

```
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Action_cookie_main.JSP

```
<%@ page language="java" contentType="text/html; %>

<%
   Cookie username = new Cookie("username", request.getParameter("username"));
   Cookie email = new Cookie("email", request.getParameter("email"));
   username.setMaxAge(60*60*10);
   email.setMaxAge(60*60*10);

   // Add both the cookies in the response header.
   response.addCookie( username );
   response.addCookie( email );
%>

<html>
<head>
<title> Cookie JSP</title>
</head>
<body>

<b>Username:</b>
   <%= request.getParameter("username")%>
<b>Email:</b>
   <%= request.getParameter("email")%>

</body>
</html>
```

Handling and Errors and Exception with JSP Page

Exception referred to the error occurred at run time. In java exception handled through the exception object. In JSP exception is implicit object with page scope.It is an instance of java.lang.throwable as you know that Throwable class is the super class of all the exception and error classes in the java .Here with the JSP we can handle exception with Scriptlets, by creating an errorpage and with deployment descriptor.

Exception handling using Scriptlet:
Exception can be handled in a scriptlet in the same way as in java, by using the try and catch block.

Exception handling using the page Directive:
This is the second way to handle exception by creating an error page. To create an error page with JSP we can use isErrorpage=true attribute of page directive. After creating an error page we can handle the

exception with any JSP page by invoking this error page with it using errorPage=”URL of the error page” attribute of the page directive.

Exception handling using Deployment Descriptor
This is the third way to by which we can handle an exception with JSP page. Deployment descriptor is a web.xml file which defines the classes, resources and configuration of the application and how the web server uses them to serve web requests. It resides in the application under the WEB-INF/directory. If an error page is defined for handling an exception, the request is directed to the error page’s URL. The web application deployment descriptor uses the <error–page> tag. To define web components that handle errors. We can set deployment descriptor for error handling in two ways, either using exception type or using error code.

<error-page>
<exception-type>java.lang.throwable</exception-type>
<location>/errorpage.JSP</location>
<error-page>
<error-page>
<exception-type>500</exception-type>
<location>/erorpage.JSP</location>
<error-page>

Including and forwarding from JSP

JSP pages have the ability to include other JSP pages or servlet in the output that is sent to the client, or to forward the request to other JSP page or servlet for servicing. This thing is possible through the standard actions, <JSP: include> and <JSP: forward>.

Include Action
Including a JSP page or servlet through a standard differs from the include directive.
<%@ include file=”/WEB-INF/footer.JSP”>
<JSP:directive.include file=”/WEB-INF/footer.JSP”>
When the JSP container translates the page, this directive causes the indicated file to be included in that place in the page and become part of the java source file that is compiled into the JSP page implementation class; that is, it is included at translation time. Using the include directive, the included file does not need to be a complete and valid JSP page.

With the “include” standard action, the JSP file stops processing the current request and passing the request to the included file. The included file passes its output to the response. After that control of the response to the calling JSP, which completes further process of the response? The output of the included page or servlet is included at request time. Components that are included via the include action must be valid JSP pages or servlet.

The included file is neither allowed to modify the headers of the response, not to set cookies in the response.

Syntax:
<JSP:include page=”URL” flush=”true/false”>
<JSP:param name=”paramname” value=”paramvalue”/>
</JSP:include>

For the include element, the page attribute is required, and its value is the URL of the page whose output is included in the response.this URL is relative to the JSP page. The flush attribute is optional, and it indicates whether the output buffer should be flushed before the included file is called.the default value is false.If the JSP needs to pass parameters to the included file, it does so with the <JSP:param>element.one element is used for each parameter. This element is optional. If it is included, both the name and value attributes are required. The included JSP page can access the parameters using the getParameter() and getParameterValues() methods of the request object

Forward action

With the forward action, the current page stops processing the request and forward the request to another web component. These other components complete the response. Execution never returns to the calling page. unlike the include action, which can occur at any time during a response, the forward action must occur prior to writing any output-to-Output Stream. In other words, the forward action must occur prior to any HTML template data in the JSP, and prior to any Scriptlet or expressions that write data to the Output Stream. If any output has occurred in calling JSP, an exception will be thrown when the forward action is encountered.

Syntax:

<JSP:forward page=”URL”>
<JSP:param name=”paramname” value=”paramvalue”/>
</JSP:forward>

The meaning and use of the attributes and of the <JSP:param>element are the same as those for the include action.

JDBC with JSP

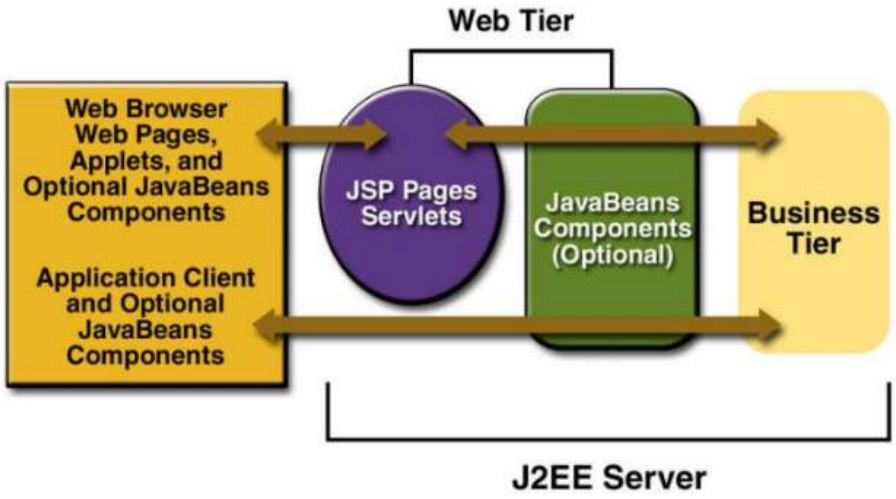
```
<%@ page import="java.sql.*" %>

<html>
<body>
<table border="1">
 <th>Roll</th><th>Name</th>
 <%!
   Connection con = null;
   Statement stmt = null;
   ResultSet rs = null;
 %>
 <%
   try {
     Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/stud_db", "root", "");
     if(con!=null) {
       out.println("Successfully connected to " + "MySQL server using TCP/IP..." + "<br>");
       stmt = con.createStatement();
       rs = stmt.executeQuery("select * from students");
     }
     while (rs.next()) {
 %>
```

```
        <tr>
          <td><%=rs.getString(1)%></td>
          <td><%=rs.getString(2)%></td>
        </tr>
  <%
      }
    }
    catch (Exception e) {
        out.println("Exception: " + e.getMessage());
    }
    finally {
       try {
         if (con != null) {
            con.close();
         }
       }
       catch (SQLException e) { }
    }
  %>
</table>
</body>
</html>
```

JavaBeans Introduction

- The server and client tiers might also include components based on the Java-Beans component architecture (JavaBeans components) to manage the data flow between an application client or applet and components running on the J2EE server, or between server components and a database.
- JavaBeans components are not considered J2EE components by the J2EE specification. JavaBeans components have properties and have get and set methods for accessing the properties.
- JavaBeans components used in this way are typically simple in design and implementation but should conform to the naming and design conventions outlined in the JavaBeans component architecture.



As shown in above Figure, the web tier, like the client tier, might include a Java-Beans component to manage the user input and send that input to enterprise beans running in the business tier for processing.

Java Bean Properties

To define a property in a bean class, supply public getter and setter methods. A builder tool like NetBeans recognizes the method names and shows the mouth Width property in its list of properties. It also recognizes the type, int, and provides an appropriate editor so the property can be manipulated at design time. Various specializations of basic properties are available and described in the following sections.

Indexed Properties

An indexed property is an array instead of a single value. In this case, the bean class provides a method for getting and setting the entire array.

Bound Properties

A bound property notifies listeners when its value changes. This has two implications:

1. The bean class includes addPropertyChangeListener() and removePropertyChangeListener() methods for managing the bean's listeners.
2. When a bound property is changed, the bean sends a PropertyChangeEvent to its registered listeners.

PropertyChangeEvent and PropertyChangeListener live in the java.beans package. The java.beans package also includes a class, PropertyChangeSupport, that takes care of most of the work of bound properties. This handy class keeps track of property listeners and includes a convenience method that fires property change events to all registered listeners.

Bound properties can be tied directly to other bean properties using a builder tool like NetBeans. NetBeans allows you to do this without writing any code.

Constrained Properties

A constrained property is a special kind of bound property. For a constrained property, the bean keeps track of a set of veto listeners. When a constrained property is about to change, the listeners are consulted about the change. Any one of the listeners has a chance to veto the change, in which case the property remains unchanged. The veto listeners are separate from the property change listeners. Fortunately, the java.beans package includes a Vetoable Change Support class that greatly simplifies constrained properties.

JavaBean Methods

A bean's methods are the things it can do. Any public method that is not part of a property definition is a bean method.

When you use a bean in the context of a builder tool like NetBeans, you can use a bean's methods as part of your application. For example, you could wire a button press to call one of your bean's methods.

Package java.beans Description

- Contains classes related to developing beans -- components based on the JavaBeansTM architecture.
- A few of the classes are used by beans while they run in an application. For example, the event classes are used by beans that fire property and vetoable change events (see PropertyChangeEvent). However, most of the classes in this package are meant to be used by a

bean editor (that is, a development environment for customizing and putting together beans to create an application).

- In particular, these classes help the bean editor create a user interface that the user can use to customize the bean. For example, a bean may contain a property of a special type that a bean editor may not know how to handle. By using the PropertyEditor interface, a bean developer can provide an editor for this special type.
- To minimize the resources used by a bean, the classes used by bean editors are loaded only when the bean is being edited. They are not needed while the bean is running in an application and therefore not loaded. This information is kept in what's called a bean-info (see BeanInfo).
- Unless explicitly stated, null values or empty Strings are not valid parameters for the methods in this package. You may expect to see exceptions if these parameters are used.