**JSP Interview Questions**

**Q1. Differentiate between Include Directive and Include Action.**

**Ans:** The Difference between both can be explained as follows:

|  |  |
| --- | --- |
| **Include Directive** | **Include Action** |
| Includes content at page translation time | Includes content at page request time |
| Preferred in Static Pages | Preferred in Dynamic Pages |
| Includes Original content of the page | Does not include Original content of the page |
| Cannot invoke include() method | Can invoke include() method |
| Page size can be increased in the run-time | Page size is fixed |

**Q2. What is JSP?**

**Ans:** JSP is an abbreviation for **Java Servlet Page**. It is a Server-Side Programming Language used to create dynamic web-pages in the form of HTML. The JSP page is implicitly converted into a servlet and it enables some additional features such as Expression Language, Custom Tags, and many more.

**Q3. How can you include the results of another page?**

**Ans:** The results of another page can be included by using the following methods:

* **Include Directive**
* **Include Action**

**Q4. Mention some important JSP Action Tags.**

**Ans:** The most frequently used **JSP Action** Tags are as follows:

|  |  |
| --- | --- |
| **JSP Action Tags** | **Description** |
| jsp:forward | Forward the request and response to another resource |
| jsp:include | To include another Resource |
| jsp:useBean | Locate/Create another bean object |
| jsp:setProperty | Sets the property value in a bean object |
| jsp:getPropety | Prints Property Value of the bean object |
| jsp:fallback | Used to print a message if the plugin is working |
| jsp:plugin | Used to embed another component such as an applet |
| jsp:param | Sets the Parameter value |

**Q5. How can I use JSP in the MVC model?**

**Ans:** JSP is used in MVC in the **presentation** tasks. It is used as a **view.** The controller calls the model and the business classes that get the data. This data is rendered to the JSP for rendering it to the **Client.**

**Q6. What do you mean by Context Initialization Parameters?**

**Ans:** Context Initialization Parameters are the **Initializing Parameters** for the whole application. They are not specific to any **Servlet** or a **JSP.** Context Initialization Parameters are specified using the following syntax in a **web.xml** file.

|  |  |
| --- | --- |
| 1  2  3  4 | <context-param>      <param-name>parametername</param-name>      <param-value>parametervalue</param-value>  </context-param> |

**Q7. Mention the scope values for <jsp.useBean> tag.**

**Ans:** There are mainly four scope values available for **<jsp.useBean>** tag.

* page
* request
* application
* session

**Q8. What are the Literals used in JSP?**

**Ans:** The **Literals** used in JSP are as follows:

* Null
* Boolean
* String
* Integer
* Float

**Q9. What is the major difference between ServletContext and PageContext?**

**Ans:**The major difference between ServletContect and PageContext is, the **ServletContext** is designed to provide information about the **Container** and on the other hand, the **PageContext** is designed to provide information about the **Request.**

**Q10. Why are the request.getRequestDispatcher() and context.getRequestDispatcher() used?**

**Ans:** The **RequestDispatcher()** and the **context.getRequestDispatcher()** are used for the following purposes.

* **request.getRequestDispatcher()** is used to create request. We need to give the relative path of the resource.
* **context.getRequestDispatcher()** is used to create context. We need to give the absolute path of the resource.

**Intermediate Level JSP Interview Questions**

**Q11. List down the major differences between the JSP Custom Tags and Java Beans.**

**Ans:** The Major Differences between **JSP Custom Tags** and **Java Beans** are as follows:

|  |  |
| --- | --- |
| **Custom Tags** | **Java Beans** |
| Custom Tags can manipulate JSP content | Java Beans cannot manipulate JSP content |
| Executing complex operations is simple | Executing complex operations is difficult |
| Custom Tags are hard to set up | Java Beans are simple to set up |
| Custom Tags are available only in JSP 1.1 | Java Beans are used in all JSP 1.x versions |

**Q12. How are Custom Tags in JSP created?**

**Ans:** Custom Tags in JSP are created using the following steps.

1. Creating the Tag Handler Class
2. Creating the TLD File
3. Creating the JSP File

**Creating the Tag Handler Class:**

To create a Tag Handler Class, we need to inherit the TagSupport Class and then override **doStartTag()** method. To write the data for JSP, we need to use the **JspWriter** class. The PageContext class provides **getOut()** method which returns the instance of the **JspWriter** class. Later, the **TagSupport** class provides an instance of **pageContext** by default.

**Creating the TLD File:**

TLD stands for **Tag Library Descriptor** file. It contains the information related to the tag and Tag Hander classes. It must be held inside the **WEB-INF** directory.

**Creating the JSP File:**

We will be specifying the path of the TLD file directly. It is recommended to use the URI name instead of full a path of the TLD file. It uses **taglib** directive to use the tags defined in the TLD file.

**Q13. Mention the various Implicit Objects used in the Expression**

**Ans:** The various Implicit Objects used are as follows:

|  |  |
| --- | --- |
| **Implicit Objects** | **Description** |
| pageScope | Maps the attribute name with the value set in the page scope |
| requestScope | Maps the attribute name with the value set in the request scope |
| param | Maps the request parameter to a single value |
| sessionScope | Maps the attribute name with the value set in the session scope |
| applicationScope | Maps the attribute name with the value set in the application scope |
| paramValues | Maps the request parameter to an array of values |
| header | Maps the request header name to the single value |
| cookie | Maps the cookie name to the cookie value |
| pageContext | Provides access to Object requests, session and many more |
| initParam | Maps the Initialization Parameter |
| headerValues | Maps the request header name to the single values |

**Q14. Mention the three important tags used in the development of JSP Bean.**

**Ans:** The Three tags used in the **JSP Bean development** are as follows:

* jsp:useBean
* jsp:setProperty
* jsp:getProperty

**Q15. Can you disable the caching on the back button of a particular browser?**

**Ans: Yes,** The Caching process can be **disabled** on the **back button** of the **browser.** To do so, we can use the following code below.

|  |  |
| --- | --- |
| 1 | <%response.setHeader("Cache-Control","no-store");  response.setHeader("Pragma","no-cache");  response.setHeader ("Expires", "0"); %> |

**Q16. Mention the Implicit Objects in a JSP.**

**Ans:** The **Web Container** creates certain objects that include the information related to a particular *Request, Application*or a*Page*. These Objects are called as **Implicit Objects.** The Implicit Objects in the JSP are as follows:

1. **Request**
2. **Response**
3. **Application**
4. **Exception**
5. **Config**
6. **Page**
7. **Session**
8. **PageContext**
9. **Out**

**Q17. Can you stop Multiple Submits to a Web Page that are initiated by clicking to refresh button?**

**Ans:** Yes, This issue can be solved by using a **Post/Redirect/Get** or a **PRG** pattern.

* A form filed by the user gets submitted to the server using **POST/GET** method.
* The **state** in the database and business model are updated.
* A redirect response is used to reply by the **servlet** for a view page.
* A view is loaded by the browser using the **GET** command and no user data is sent.
* This is safe from multiple submits as it is a separate **JSP** page.

**Q18. How to include static files in a JSP?**

**Ans:** **Static pages** can be included in a JSP using the **include directive.** This way the inclusion is performed in the translation phase once. Note that a relative **URL** must be supplied for file attribute. Although static resources may be included, it is not preferred as each **request** requires inclusion.

**Q19. How can we stop errors on Display in a JSP Page?**

**Ans:** We can stop errors in display in a JSP Page by setting up an **“ErrorPage”**  attribute of the PAGE directory to the name of the error page in the JSP page, and then in the error JSP page set **“isErrorpage=”TRUE”.**

**Q20. Can a Constructor be used in place of init() method to initialize a servlet?**

**Ans:** Yes, We can use a constructor in place of **init()** method. But it is not preferred because init() was developed because earlier Java versions could not invoke constructors with arguments dynamically. So they could not assign a **servletConfig.** However, servlet containers still call an only no-arg constructor. So there is no access to **servletContext** or **servletConfig.**

**Q21. What are the different Life-Cycle methods?**

**Ans:** The different Life-Cycle Methods are as follows:

* **jspInit()**
* **\_jspService()**
* **jspDestroy**

**jspInit():**Container calls **jspInit()** method to initialize servlet instance. It is called **once** for the servlet instance and **preceded** every other method.

**\_jspService():**Container calls **\_jspService()** method for each **request** and passes it on to the **objects.**

**jspDestroy():**Container calls the **jspDestroy()**just before destruction of the instance.

**Q22. What are the attributes on page directives?**

**Ans:** The different **attributes** of the Page Directives are as follows;

* **Session:** It is designed to show if any **session** data is available to the page or not.
* **Import:** It is dedicated show **packages** that are imported.
* **isELIgnored:** It shows whether **EL expressions** are ignored when JSP transforms into a servlet.
* **contentType:** It allows the user to specify the **content-type** of the page.

**Q23. Explain Client-Side and Server-Side Validation.**

**Ans:**The **Client-Side** validation is done using **JavaScript.** The validation takes place within the browser. Javascript is used to submit the data in the form when the validation is **successful.** Validation errors do not require any extra network trip because the form cannot be submitted if there are any errors.

Similar kind of data validation is carried out in the **Server-Side** after submission of the form. In if the validation **fails,** then, the extra network trip is required to **resend** the form to the client to refill the form with the correct data.

**Q24. Explain Translation Phase.**

**Ans:** During the Translation Phase, the **JSP engine** translates and compiles a JSP file into a **servlet.** This servlet moves to the **execution** **phase** where all the requests and responses are handled. They are compiled for the **first time.** They are not accessed unless they are manually compiled. The **manual/explicit compilation** is useful for long and convoluted programs.

**Q25. What is Object Cloning?**

**Ans:** The **object cloning** is a process of creating an exact copy of the existing object. The **clone()** method of Object class is used to create the clone an existing object. The class, whose object the user tries to clone is expected to implement the **java.lang.Cloneable interface.** If it does not implement the **Cloneable interface,** then the clone() method generates the **CloneNotSupportedException.**

|  |  |
| --- | --- |
| 1 | **protected** Object clone() **throws** CloneNotSupportedException |

**Q26. Write a simple example for the Clone() Method.**

**Ans:** The code is a simple example for the **Clone()** Method.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | **class** Student18 **implements** Cloneable{  **int** rollno;       String name;       Student18(**int** rollno,String name){  **this**.rollno=rollno;  **this**.name=name;       }  **public** Object clone()**throws** CloneNotSupportedException{  **return** **super**.clone();       }  **public** **static** **void** main(String args[]){  **try**{                 Student18 s1=**new** Student18(101102,"Arjun");                 Student18 s2=(Student18)s1.clone();                 System.out.println(s1.rollno+" "+s1.name);                 System.out.println(s2.rollno+" "+s2.name);            }  **catch**(CloneNotSupportedException c){            }       }  } |

**//Output:**

101102 Arjun  
101102 Arjun

**Q27. Define JSP Declaration.**

**Ans**: The **JSP declaration tag** is used to declare fields and methods. The code written inside the JSP declaration is enclosed in <%!%> tag. It is placed outside the **service()** method of the auto-generated servlet.

**Syntax:**

|  |  |
| --- | --- |
| 1 | <%! field or method declaration %> |

**Example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <html>       <body>            <%! **int** data=50; %>            <%= "Value of the variable is:"+data %>       </body>  </html> |

**Q28. Differentiate between JSP Scriptlet tag and Declaration tag.**

**Ans:** The difference in both is discussed as follows:

|  |  |
| --- | --- |
| **JSP Scriptlet Tag** | **Declaration Tag** |
| JSP Scriptlet Tag can only declare Variables | Declaration Tag declares Methods and Variables |
| Scriptlet Tag is placed within \_jspService() | Declaration Tag is placed outside \_jspService() |

**Q29. How are JSP(Java Server Pages) better than ASP(Active Server Pages)?**

**Ans:** The advantages of **JSP** over **ASP** are as follows:

* The dynamic part of the code is written in **Java,** not in **Visual Basic** or the **Microsoft-specific** language. Hence, it is **powerful** and **easier** to use.
* It is **portable** to other operating systems and **Non-Microsoft** Web servers.

**30. Mention the advantages of JSP over Pure Servlets?**

**Ans:**Some of the Major Advantages of JSP over Pure Servlets are as discussed below:

* It is more convenient to write and modify normal **HTML** than to have plenty of **println** statements that generate the **HTML.**
* **Embedding** of Java code in HTML pages.
* **Platform** independence.
* Creation of **database-driven** Web applications.
* **Server-side** programming capabilities.

**Advanced Level JSP Interview Questions**

**Q31. What is Auto-Flush Attribute?**

**Ans:** The **autoFlush** attribute is used to specify if a buffered output should be flushed automatically when the buffer is filled, or whether an exception should be raised to indicate buffer overflow. A value of **true** by default indicates automatic buffer flushing and a value of **false** throws an exception.

**Q32. What do you mean by isScriptingEnabled Attribute?**

**Ans:** **isScriptingEnabled** attribute determines if scripting elements are allowed for use or not. The default value is **true** and it enables scriptlets, expressions, and declarations. If the attribute’s value is set to false, a translation-time error will be raised if the JSP uses any scriptlets, **expressions/declarations.**

**Q33. What are the steps involved in reading data from a form using JSP?**

**Ans:** The data parsing is a JSP is **Automatic.** It is done through the following steps depending on the situation.

1. **getParameter():** request.getParameter() method is called to get the value of the **form parameter.**
2. **getParameterValues():** This method is called if the parameter appears more than once and **returns multiple values.**
3. **getParameterNames():**This method is called if the user wants a complete list of all parameters in the current **request.**
4. **getInputStream():** This method is used to **read binary data** stream coming from the client.

**Q34. How are cookies set in JSP?**

**Ans:** Setting cookies with JSP involves the following steps:

1. **Creating a Cookie object:**Cookie constructor is called with a cookie **name** and a cookie **value,** both are strings.
2. **Setting the maximum age:** **setMaxAge** is used to specify the length of the cookie(in seconds) should be valid.
3. **Sending the cookie into the HTTP response headers:** **response.addCookie** is used to add cookies in the HTTP response header.

**Q35. How do you delete the Session Data?**

**Ans:** Deleting the Session Data involves the following steps.

1. **Remove a particular attribute:** *public void removeAttribute(String name)* method is called to delete the value associated with the particular key.
2. **Delete the whole session:** *public void invalidate()* method is called to discard an entire session.
3. **Setting the Session timeout:** *public void setMaxInactiveInterval(int interval)* method is called to set the timeout for a session individually.
4. **Log the user out:**The **logout** is called to log the client out of the Web server and invalidate all sessions belonging to all the users.
5. **web.xml Configuration:**In **Tomcat,** using the above-mentioned methods, one can configure session time out in **web.xml** file as follows.

**Q36. How to delete a Cookie in JSP?**

**Ans:** The following code snippet is followed to **delete** a cookie in JSP.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Cookie mycookie = **new** Cookie("name","value");  response.addCookie(mycookie);  Cookie killcookie = **new** Cookie("mycookie","value");  killcookie . set MaxAge ( 0 );  killcookie . set Path (" / ");  killcookie . addCookie ( killcookie 1 ); |

**Q37. Explain the difference between forward and sendRedirect?**

**Ans:** When a **forward** request is called, the request is sent to a different resource on the server, without the client being informed that a different resource is going to process the request. This process occurs completely with in the web container.

When a **sendRedirtect** method is invoked, it causes the web container to return to the browser indicating that a new **URL** should be requested. Since the browser issues a completely new request any objects that are stored as request attributes before the redirect occurs will be lost. This extra round trip a redirect is **slower than forward.**

**Q38. Mention the JSTL core tags.**

**Ans: JSTL core tags**are as follows.

* **<c:out> tag:** It is used for displaying the content on client after escaping **XML** and **HTML** markup tags. Main attributes are default and **escapeXML.**
* **<c:set> tag:** This tag is useful for setting up a **variable value** in a specified scope. It basically evaluates an expression and sets the result in the given variable.
* **<c:remove> tag:** It is used for **removing** an **attribute** from a specified scope or from all scopes (page, request, session and application. By default removes from all.
* **<c: if> tag:** This **JSTL** core tag is used for testing conditions. There are two other optional attributes for this tag which are var and scope, the test is mandatory.
* **<c:choose> tag:** It’s like **switch** statement in Java.
* **<c:when> tag:** It’s like **case** statement in Java.
* **<c:otherwise> tag:** It works like **default** attribute in **switch-case** statements.
* **<c:catch>tag:** This tag is used in **exception handling.** In this post, we have discussed exception handling using **<c:catch>** core tag.
* **<c:import> tag:** This **JSTL** core tag is used for importing the content from another **file/page** to the current JSP page. Attributes – var, URL and scope.
* **<c:forEach> tag:** This tag in **JSTL** is used for executing the same set of statements for a **finite** number of times.
* **<c:forTokens> tag:** It is used for **iteration** but it only works with the delimiter.
* **<c:param> tag:** This JSTL tag is mostly used with **<c:url>** and **<c:redirect>** tags. It adds parameter and their values to the output of these tags.
* **<c:url> tag:** It is used for URL formatting or **URL encoding.** It converts a relative URL into an application context’s URL. Optional attributes var, context and scope.
* **<c:redirect> tag:** It is used for redirecting the current page to another **URL,** provide the relative address in the URL attribute of this tag and the page will be **redirected** to the URL.

**Q39. Why are JSP pages preferred for creating web-based client program?**

**Ans:**JSP is preferred for creating web-based client program. Because no **plug-ins/security** policy files are needed on the client systems whereas applet does. Also, JSP pages enable cleaner and more module application design because they provide a way to separate applications programming from web page design. This means personnel involved in web page design do not need to understand Java programming language syntax to do their jobs.

**Q40. How can you make the Finally Clause not to fail to execute?**

**Ans:** It is possible to make the **Finally Clause** to not to fail by using **System.exit(1);** in the try block.

**Q41. How can we retrieve Warnings?**

Ans: **SQLWarning** objects are a subclass of **SQLException** that deal with database access warnings. Warnings do not stop the execution of an application, as exceptions do; they simply alert the user that something did not happen as planned. A warning can be reported on a Connection object, a Statement object including PreparedStatement and CallableStatement objects, or a ResultSet object. Each of these classes has a **getWarnings** method, which you must invoke in order to see the first warning reported on the calling object.

The following code snippet can be used to retrieve Warnings.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | SQLWarning warning = stmt.getWarnings();  **if** (warning != **null**){  **while** (warning != **null**){              System.out.println(\"Message: \" + warning.getMessage());              System.out.println(\"SQLState: \" + warning.getSQLState());              System.out.print(\"Vendor error code: \");              System.out.println(warning.getErrorCode());              warning = warning.getNextWarning();       }  } |

**Q42. Why Does Jcomponent have Add() And Remove() methods but the component does not?**

**Ans:** It is because, the **JComponent** is a subclass of **Container,** and it can contain other components and **JComponents.**You can make your JSPs **thread-safe** by having them implement the SingleThreadModel interface. This is done by adding the directive **<%@ page isThreadSafe=”false” % >** within your JSP page

**Q43. Explain some JSP Life-Cycle methods that can be overridden.**

**Ans:**You can override the **jspInit()** and **jspDestroy()** methods within a JSP page. It is good programming practice to free any allocated resources within **jspDestroy().** The **jspInit()** and **jspDestroy()** methods are each executed just once during the lifecycle of a JSP page and are typically declared as JSP declarations:

**Q44. How can I declare methods within my JSP page?**

**Ans:** Methods can be declared for use within a **JSP** page. The methods are invoked within any other methods you declare, or within **JSP scriptlets** and **expressions.**

**NOTE:**Do note that you do not have **direct access** to any of the **JSP implicit objects** like request, response, session and so forth from within JSP methods. However, you should be able to pass any of the implicit JSP variables as parameters to the methods you declare.

**Q45. How does a servlet communicate with a JSP page?**

**Ans:**The following code snippet shows how a servlet **instantiates** a bean and initializes it with **FORM** data posted by a browser. The **bean** is then placed into the request, and the call is then forwarded to the JSP page, **Bean1.jsp,** by means of a **request dispatcher** for downstream processing.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **public** **void** doPost (HttpServletRequest request, HttpServletResponse response){  **try** {             govi.FormBean f = **new** govi.FormBean();             String id = request.getParameter("id");             f.setName(request.getParameter("name"));             f.setAddr(request.getParameter("addr"));             f.setAge(request.getParameter("age"));             f.setPersonalizationInfo(info);             request.setAttribute("fBean",f);             getServletConfig().getServletContext().getRequestDispatcher             ("/jsp/Bean1.jsp").forward(request, response);       }  **catch** (Exception ex) {       }  } |

The JSP page **Bean1.jsp** can then process **fBean,** a fter first extracting it from the default request scope via the useBean action.

|  |  |
| --- | --- |
| 1  2  3  4 | jsp:useBean id="fBean" **class**="govi.FormBean" scope="request"/ jsp:getProperty name="fBean"  property="name" / jsp:getProperty name="fBean"  property="addr" / jsp:getProperty name="fBean" property="age" / jsp:getProperty name="fBean"  property="personalizationInfo" / |

**Q46. What is a Hidden Comment?**

**Ans:** A comment that documents the JSP page but is not sent to the client is known as a **Hidden comment.** The JSP engine ignores a hidden comment and does not process any code within hidden comment tags. A hidden comment is not sent to the client, either in the displayed JSP page or the HTML page source. The hidden comment is useful when you want to hide or “comment out” part of your JSP page.

You can use any characters in the body of the comment except the closing –%> combination. If you need to use –%> in your comment, you can escape it by typing –%>.

**JSP Syntax:**

|  |  |
| --- | --- |
| 1 | <%-- comment --%> |

**Examples**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <%@ page language="java" %>  <html>      <head><title>A Hidden Comment </title></head>          <body>               <%-- This comment will not be visible to the client in the page source --%>          </body>  </html> |

**Q47. Can you disable JSP Scripting?**

**Ans:** Yes, **Scripting is disabled** by setting the scripting-invalid element of the deployment descriptor to true. It is a **sub-element** of **JSP-property-group.** Its valid values are true and false.

The **syntax** for disabling scripting is as follows:

|  |  |
| --- | --- |
| 1  2  3  4 | <jsp-property-group>  <url-pattern>\*.jsp</url-pattern>  <scripting-invalid>**true**</scripting-invalid>  </jsp-property-group> |

**Q48. How to deactivate EL on JSP?**

**Ans:**There are two ways to ignore the execution of an **(EL) Expression Language** on a **JSP** page.

* Use the directive  **<% @ page isELIgnored = “true”%>.**
* Configure web.xml (**best suited to disable EL on multiple pages**)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <jsp-config>     <jsp-property-group>         <url-pattern>\*.jsp</url-pattern>         <el-ignored>**true**</el-ignored>     </jsp-property-group>  </jsp-config> |

**Q49. When does a container initialize multiple JSP objects?**

**Ans:** In the case, where there are multiple **servlets** and **servlet-mapping** elements in the deployment descriptor for one **servlet** or **JSP** page, then the **container** initializes an object for each element and each of these objects has its own **ServletConfig** object and initialization parameters.

The following code snippet uses one **JSP** page in **web.xml** as shown below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <servlet>  <servlet-name>Test</servlet-name>       <jsp-file>/WEB-INF/test.jsp</jsp-file>           <init-param>               <param-name>test</param-name>               <param-value>Test Value</param-value>           </init-param>  </servlet>    <servlet-mapping>        <servlet-name>Test</servlet-name>        <url-pattern>/Test.**do**</url-pattern>  </servlet-mapping>    <servlet>        <servlet-name>Test1</servlet-name>        <jsp-file>/WEB-INF/test.jsp</jsp-file>  </servlet>    <servlet-mapping>        <servlet-name>Test1</servlet-name>        <url-pattern>/Test1.**do**</url-pattern>  </servlet-mapping> |

**Q50. Give a sample JSP configuration in the deployment descriptor.**

**Ans:**The **JSP-config** element is used to configure various parameters of JSP pages.

* Management of **scriptlet** elements on the page,
* Controlling the **execution** of expressions in a language
* URL pattern definition for **encoding,**
* Determining the **size of the** **buffer** that is used for objects on the page
* Identification of resource groups corresponding to a **URL pattern** to be processed as an XML document.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | jsp-config>         <taglib>             <taglib-uri>https://www.edureka.co/jsp/tlds/mytags</taglib-uri>             <taglib-location>/WEB-INF/numberformatter.tld</taglib-location>         </taglib>  </jsp-config> |