**Servlets**

**Lesson2: Working with WildFly 8.x**

1. Discuss standalone.xml – (whatever is covered in syllabus)

*In standalone mode each wildfly 8 server instance is an independent process. The configuration files, deployment content and writable areas used by the single standalone server run from a WildFly installation)]*

*Datebase info and data source info- standalone.xml file*

1. WildFly server – App Server

*Two types of servers available [web server and application server], Jboss is an application server, Wildfly is flexible, lightweight, managed application runtime that help us build JEE applications, Some of the features supporeted by wildfly are as follows:*

* *Application Container,*
* *Java Messaging Service,*
* *Java Naming and Directory Interface (JNDI)*
* *ORM integrated for persistance programming*
* *Servlet 3.0 and JSP 2.1*
* *Latest JEE standards and technology*

*Some of the features are:*

* + *Users forket version of TOMCat server internally*
  + *Fully java complaint and works with javaSE 8*
  + *Multi OS Support,*
  + *Integration with hibernat*
  + *Caching and clustering*

**Lesson3: Introduction to Servlets API 3.0 and Ease of Development through Annotations**

1. Discuss @WebServlet with multiple url pattern,@MultipartConfig annotation

@WebServlet

* @*webServlet annotation affirms the annotede class is a Servlet*
* *@webservlet annotation is also used to hold meta information about the servlet*
* *Of all the attributes of webservlet the url pattern is an mandatory attribute*
* *Multiple url patterns can be allocated to the Servlet using the urlPatters attribute of the annotation*
* *Some of the other attributes of the annotation such as:*
  + displayName
  + initPatterns
  + loatOnStartup
  + name
  + urlPatterns
  + description
  + Value

1. Servlet life cycle(service ,doget and dopost method)

*The Servlet interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods and are called in the following sequence:*

* *init()*
* *service()*
* *destroy()*
* *init()*

*The init() method is guaranteed to be called only once during the Servlet’s lifecycle. The Servlet performs one-time setup configurations in this method. It stores the ServletConfig[\*] object so that it can be retrieved later by calling the Servlet’s getServletConfig() method (This is handled by GenericServlet). The ServletConfig object contains Servlet parameters and a reference to the*

*Servlet’s ServletContext.*

* *Service()*

*service() method gets called every time a new request comes in. The method is called concurrently (that is, multiple threads may call this method at the same time) so it should be implemented in a thread-safe manner.*

* *Destroy()*

*When servlet needs to be unloaded (for example: since a new version should be loaded or the server is shutting down), the destroy() method is called. This method too is guaranteed to be called only once during the Servlet’s lifecycle.All resources which were allocated in init() should be released in destroy().*

1. Discuss - Web Container

*Web Container is essentially a virtual Container that manages web components [interacts with servlets, manages lifes cycle calls…].*

*Some points to note of a web container:*

* *Provides communication support between Web Components and Web Server: Servlet as a web component does not communicate with Web Server directly. All the communication happens via Web container*
* *Provides Life cycle support for Web Components: Life cycle of Web components (Servlets and JSP) is managed by the container Provides Networking support: Web components do not need to open network and socket connections all is taken care by Web server*
* *Enabling Web Security: Access to Web components can be secured by granting privileges.*
* *Provides Multi Threading support: Web components are multi-thread, for every request there is one thread generated. Thus all threads are managed by container. We do not need to do explicit multi-threading.*

**Lesson 4:The Request object & Lesson 5:The Response object**

1. Discuss all the request object methods with their return types,

***Some of the Important Request Object Methods:***

* ***void ServletRequest.setAttribute(String arg, Object arg):*** *method is used to add attributes to the request object by providing key value pair of which key is the identifier for the data and value is the data itself. One Servlet can make data available to the next servlet by adding the data as attribute to its request*
* ***Object ServletRequest.getAttribute(String arg):*** *method is used to get access to the attribute available in the request object. In order to get access we have to pass the identifier which was used as an identifier to add the data to the request via setAttribute method, Since this method can return any object the return type is Object type.*
* ***void ServletRequest.removeAttribute(String arg):*** *this method is used to remove the attribute available in the request scope*
* ***Enumeration ServletRequest.getAttributeNames():***  *this method is used to get an Enumeration on the attributes names of all the attributes available in the request scope*
* ***Cookie[] HttpServletRequest.getCookies():*** *method is used to get all cookie objects that arrived from the client current request**as an Cookie array*
* ***String HttpServletRequest.getHeader(String arg):****method is used to get access to a particular request header value.*
* ***Enumeration HttpServletRequest.getHeaderNames():*** *method is used to get access to enumeration over all the header names available via the request object*
* ***String ServletRequest.getParameter(String arg):*** *method is used to fetch form request parameter based on parameter identifier provided, parameter is fetched as a String*
* ***String[] ServletRequest.****getParameterValues(): method is used to fetch multi values from form elements such as multi select drop down list or checkboxes, return is a string array*
* ***Enumeration ServletRequest****.****getParameterNames():*** *method is used to fetch all parameters names from the request using the numeration interface.*
* ***String ServletRequest****.getQueryString() method returns the raw query String of the request method*
* ***String HttpServletRequest****.****getPathInfo():*** *method is used to get extra path information.*
* ***String HttpServletRequest****.****getPathTranslated():*** *method is used to get the extra path information translated to a real file system path or null if there is no extra path information.*

1. Request Headers along with the method signature.

*When a browser send a request to the server along with the client request (which could be just a request for a resource or submission of form data) the browser send loads of information to the server (metadata). This metadata that arrives at the server is reffered as Headers. There are many headers availabe to fetch its value from. A developer than can read this header and do request processing using header information or choose to completely ingnore it.*

*Header Methods are*

* ***String HttpServletRequest.getHeader(String arg):****method is used to get access to a particular request header value.*
* ***Enumeration HttpServletRequest.getHeaderNames():*** *method is used to get access to enumeration over all the header names available via the request object*

*Some of the Headers are:*

* *Accept*
* *Cache-control*
* *Connection*
* *User-Agen*
* *Host*
* *Accept*

***Methods are:***

*public void forward(ServletRequest req, ServletResponse resp)*

*public void include(ServletRequest req, ServletResponse resp)*

*The forward() method of the RequestDispatcher interface may be called by thecalling servlet only when no output has been committed to the client. If noncommitted output data exists in the response buffer, the content must be clearedbefore the target servlet's service method is called. Else, an IllegalStateException will be thrown.*

*In the case of forward() , once another web resource is being invoked , we cannot access the previous web resource. include() includes the content of a resource in the response. The included servlet*

*cannot change the response status code or set headers; any attempt to make a change is ignored.*

*The target servlet of the include() method has access to all aspects of the request object, but its use of the response object is more limited. It can only write information to the ServletOutputStream or Writer of the response object and commit a response by writing content past the end of the response buffer, or by explicitly calling the flushBuffer() method of the ServletResponse interface. It cannot set*

*headers or call any method that affects the headers of the response. In the case of include() , once another web resource is being invoked , we can come back with the response embedded to access the previous web resource.*

*RequestDispatcher rd = request.getRequestDispatcher(String arg);*

*rd.forware(request, response);*

*rd.inclues(request, response);*

**Lesson 10: Session Tracking**

1. How will you get the HTTP method?

*To determine the current http request type you can call String request.getMethod() method.*

1. How will set the session timeout using xml and using code (discuss about the minutes /seconds)

***Using XML***

*<session-config>*

*<session-timeout>30</session-timeout>*

*</session-config>*

*Here time-out is set to 30 minutes*

***User Code***

*HttpSession session = request.getSession(boolean);*

*Session.setMaxInactiveInterval(int)*

*It specifies the maximum length of time, in seconds, that the servlet engine keeps this session if no user requests have been made of the session and takes an integer as a parameter specifying the number of seconds.*

1. How will you get the session object what is the difference between getSession(true) getSession(false)

*We can get session Object by calling getSession on Request Object. The getSession method is overloaded and comes in two flavours;*

*Request.getSession();*

*Request.getSession(boolean)*

*Request.getSession() acts just like request.getSession(true);*

***The request.getSession(true)*** *method creates a new session Object if already a session has not been asgined to the current session id however if already a session object exists assigned to the session id when the method returns that sessions object and does not creates a new session object*

***The request.getSession(false)*** *method does not create a new session Object if already a session has not been asgined to the current session id however if already a session object exists assigned to the session id when the method returns that sessions object and does not creates a new session object*

1. Session.invalidate() and illegalStateException discuss

***public void invalidate()***

*It invalidates this session and unbinds any objects bound to it. Eg.once you have clicked the logout link we would want to remove your session so invalidate method needs to be called.*

*The IllegalStateException would be thrown if you are perfoming any session related activity on Invalidated Session Object*

1. All points about cooikes,hidden form(works only for a sequence of dynamically generated forms),session api (how to retrieve the session object)

***Cookies:***

*Cookie is a bit of information stored in a small text file sent by a web server to a browser that can later be read back from the browser. When a browser receives a cookie, it saves the cookie and there-after sends the cookie back to the server each time it accesses a page on that server, subject to certain rules. Since cookie’s value can uniquely identify a client, cookies are often used for session tracking. The browser is expected to support 20 cookies for each Web server, 300 cookies total, and may limit cookie size to 4 KB each.*

***Working with Cookies***

*Create a cookie with the Cookie( ) constructor:*

*public Cookie( String name, String value )*

*This creates a new cookie with an initial name and value. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number. The name of the cookie must be an HTTP/1.1 token. Tokens are strings that contain*

*none of the special characters listed in. (Alphanumeric strings qualify as tokens.) The value of the cookie can be any string, though null values are not guaranteed to work the same way on all browsers. In addition, if a cookie is sent that complies with*

*Netscape's original cookie specification, do not use whitespace or any of these characters:*

*[ ] ( ) = , " / ? @ : ;*

*A servlet can send a cookie to client by passing a Cookie object to the addCookie( ) method of HttpServletResponse:*

*public void HttpServletResponse.addCookie(Cookie cookie)*

*The method adds the specified cookie to the response. Additional cookies can be added with subsequent calls to addCookie( ).*

*Since cookies are sent using HTTP headers, they should be added to response before any content is sent.*

*The code to set a cookie looks like this:*

*Cookie cookie=new Cookie(“ID” , “123” );*

*res.addCookie(cookie);*

*A servlet retrieves cookies by calling the getCookies( ) method of HttpServletRequest.*

*public Cookies[ ] HttpServletRequest.getCookies( )*

*This method returns an array of Cookie objects that contains all the cookies sent by the browser as part of the request or null if no cookies were sent. Several cookies might have the same name but different path attributes. The code to fetch the cookies looks like this:*

*Cookie[] cookies = req.getCookies();*

*if (cookies != null) {*

*for (int i = 0; i < cookies.length; i++) {*

*if (cookies[i].getName().equals("sessionid")) {*

*String name =cookies[i].getName();*

*String value =cookies[i].getValue();*

*}*

*}*

*}*

***Hidden Form Fields:***

*In this technique the fields are added to an HTML form which are not displayed in the client’s request. The hidden form fields are sent back to the server when the form is submitted. In hidden form fields, the*

*HTML entry will be as shown below:*

*This means that when form is submitted ,the specified name and value will be get included in get or post method.*

*Execute the following servlets:*

[*http://localhost:9090/SessionManagement/login.html*](http://localhost:9090/SessionManagement/login.html)

*This would lead to servlet:*

[*http://localhost:9090/SessionManagement/HiddenFormServlet*](http://localhost:9090/SessionManagement/HiddenFormServlet)

*In HiddenFormServlet, username is added as Hidden Field in html form as shown*

*below in partial listing.*

*out.println("<form action='ShowServlet'>");*

*out.println("<input type='hidden' name='user' value='" + user + "'>");*

*out.println("<input type='submit' value='submit' >");*

*Then username is retrieved in ShowServlet.*

[*http://localhost:9090/SessionManagement/ShowServlet?user=John*](http://localhost:9090/SessionManagement/ShowServlet?user=John)*.*

*Query parameters get appended due to the GET method.*

***Session Tracking API:***

*One of the ways to track session is by using Session Tracking interfaces provided by Servlet API.*

*The API provides HttpSession interface to track and manage sessions. The servlet container uses this interface to create a session between an HTTP client and an HTTP server creating an unique identifier called Session ID. The server can maintain a session in many ways such as using cookies or rewriting URLs. This interface allows servlets to view and manipulate information about a session, such as*

*the session identifier, creation time, and last accessed time.*

*public HttpSession HttpServetRequest.getSession(boolean create) : retrieve the current HttpSession object. This method returns the current HttpSession object associated with the request or, if necessary, create a new session for the request.*

*Use true to create a new session if none exists. If create is false, and the request has no valid session then this method returns null else returns the existing session. To make sure the session is properly maintained, need to call this method at least once before writing any output to the response.*

*public void HttpSession.setAttribute(String name, Object value) :add information in form of an object to session object. This method binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced. The putValue() method is sometimes used instead of setAttribute() however it is deprecated now. public Object HttpSession.getAttribute(String name) : retrieve information object stored by setAttribute() method in form of an object from the session. Returns null if no object of that name exists. public java.lang.String getId() : This method returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet engine and is implementation dependent.*

*When a user first accesses the site, that user is assigned a new HttpSession object and a unique session ID. The Session ID identifies the user and is used to the user with HttpSession object in subsequent requests. Behind the scenes, the session ID is usually saved on the client in a cookie or sent as part of a rewritten URL. A server can discover session’s ID with the getID() method seen earlier. It returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet engine and is implementation dependent. This method throws an IllegalStateException if session is invalid.*

*Session Id is stored as part of URL to support session tracking via URL rewriting. Servlet rewrites every local URL before sending it to the client. The Servlet API’s HttpServletRequest interface provides two methods to perform this encoding: public java.lang.String encodeURL(java.lang.String url)*

*It encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged. The implementation of this method should include the logic to determine whether the session ID needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary.*

*All URLs emitted by a Servlet should be run through this method. Otherwise, URL rewriting cannot be used with browsers, which do not support cookies*

1. How will you create cookie, store cookie,add cookie and read cookie (which interfaces are used) discuss the getcookies method,maxium cookie size,methods used to keep the cookie alive

Refer to above question

1. Discuss – all the methods and points regarding cookies

For the descussion of points regarding cookies Refet to 5th point above.

*We can create a cookie by using the below code*

*Cookie cookie = new Cookie(String key, String value);*

***Methods of Cookie are:***

* *Public String getName()*
* *Public Void setValue(String ar1)*
* *Public String getValue()*
* *Public int getVersion()*
* *Public void setVersion(int)*

**JSP**

**Lesson 5 : Writing Java Server Page**

1. Discuss -Jsp life cycle & processing scenarios

*JSP Life Cycle:*

*JSP is stored as text file (.jsp) in the web application. When the client requests for the page for the first time, the server (web container) translates the JSP into Java source code (.java) and if there are no translation errors then it is compiled into a servlet class file. The servlet class is then loaded*

*by the class loader, and then instantiated.*

*For the first time request, the jspInit( ) method (if present) is called to initialize resources.*

*If the request for the JSP page is not the first time request, then the translation to Initialization steps are skipped and the request is directly processed by the \_jspService( ) method.*

*The web container invokes the jspDestroy method (if present), to cleanup resources, when the server shuts down or if the JSP page is replaced with a modified one.*

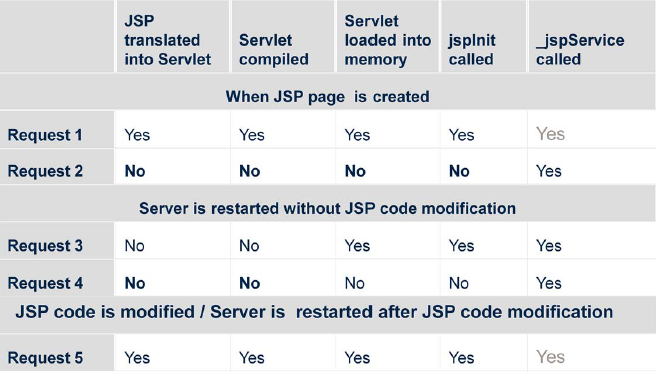
*The lifecycle methods of the JSP page translated into the servlet have the following form:*

* *public void jspInit ( )*
* *public void jspDestroy( )*
* *public void \_jspService (HttpServletRequest request,HttpServletResponse response) throws ServletException, IOException*

*The .java and .class file generated from the JSP page have the same name as the jsp file with a postfix “\_jsp”.*

*For example: If the jsp file name is helloworld.jsp, then the generated files have the names as helloworld\_jsp.java and helloworld\_jsp.class.*

*Some scenarios are displayed below:*



1. Which life cycle methods cannot be overridden

*\_JspService() method cannot be overriden.*

1. Jsp comments vs HTML comments-discuss

*HTML Comment are reperseted as follows:*

*<!-- -- >*

*Any Content between the opening and closing of the tags are ignored by the browser.*

*JSP Comment:*

*In JSP there are many comments available for us to use.*

* *Scripting Comment: <%-- -->*
* *Single line comment in the scriptlet*

*<%*

*//System.out.println(“Hello World”);*

*%>*

* *Block Comment in scriplet*

*<%*

*/\**

*\* line one*

*\* line two*

*\*/*

*%>*

* *HTML Comment in JSP*

*<!-- -->*

**Lesson 6 : JSP Scripting Elements**

1. Discuss all the JSP scripting elements i.e expression, declaration and scriplets along with syntax and when it is evaluated and also check the sample code snippets

*Scripting elements add the java code into the generated Servlet*

***JSP Scripting Elements:***

*With JSP scripting elements Java code into the servlet that will be generated from the current JSP page.*

*There are three forms of scripting elements:*

* *Expressions of the form <%= expression %> that are evaluated and inserted into the output.*
* *Scriptlets of the form <% code %> that are inserted into the servlet's service method.*
* *Declarations of the form <%! code %> that are inserted into the body of the servlet class, outside of any existing methods.*

***JSP Expression:***

*The Syntax for JSP Expression:*

*<%= java expression %>*

*The JSP expression is evaluated at run-time (when the page is requested), and thus has full access to information about the request. The JSP expression example shown on the above slide will retrieve the remote host information from the pre-defined request object and insert it in the JSP page. The result is in fact stored in the out object and inserted where the expression appears in the JSP page.*

*To simplify these expressions, there are a number of predefined variables that can be used*

*When using Java as the scripting language, remember that:*

* *A semicolon should not be used to end an expression.*
* *The expression tag can contain any expression that is valid according to the Java Language Specification.*
* *Expressions are evaluated in left-to-right order as they app ear in the tag.*

***JSP Scriptlets:***

*Scriptlets have the following form: <% Java Code %>. Scriptlets are executed when the JSP engine processes the client request. If the scriptlet produces output, then the output is stored in the out object, from which it can be displayed.*

*Note that code inside a scriptlet gets inserted exactly as written. Any static HTML (template text) before or after a scriptlet gets converted to print statements. This means that scriptlets need not contain complete Java statements, and blocks left open can affect the static HTML situated outside of the scriptlets.*

***JSP Declarations:***

* *A JSP declaration lets helps to define methods or fields that are inserted in the servlet class (outside of the service method processing the request).*
* *It has the following form: <%! Java Code %>*
* *The example, shown on the slide prints out the number of times the current page has been requested since the server booted (or the servlet class was changed and reloaded). The accessCount variable*
* *is now an instance variable that is defined only once within the lifecycle of JSP. If, however, accessCount was part of a scriptlet, then it would have been a local variable whose scope would be the service() method!*
* *The JSP usually runs as multiple “threads” of one single instance. Different threads interfere with variable access, because it will be the same variable for all of them. If variables has to be used in JSP, it*
* *should be used with “synchronized access”, but that hurts the performance. In general, any data should go either in the “session object” or the “request object” if passing data between different JSP pages.*
* *JSP declarations can be used to override jspInit( ) and jspDestroy( ) methods.*

1. *Discuss all the JSP Implicit objects*

***Predefined Variables:***

*To simplify code in JSP expressions and scriptlets, JSPs are supplied with eight automatically defined variables, sometimes are called implicit objects.*

*The available variables are as follows:*

* ***request****: This is the HttpServletRequest associated with the request, and lets to look at the request parameters (via getParameter), the request type (GET, POST, HEAD, and so on), and the incoming HTTP headers (cookies, Referrer, and so on).*
* ***response****: This is the HttpServletResponse associated with the response to the client. Note that, since the output stream is buffered, it is legal to set HTTP status codes and response headers, even though this is not permitted in regular Servlets once any output has been sent to the client.*
* ***out****: This is the PrintWriter used to send output to the client. However, in order to make the response object useful, this is a buffered version of*
* ***PrintWriter*** *called JspWriter. Note that we can adjust the buffer size, or even turn buffering off, through use of the buffer attribute of the page directive. In scriptlets, need to refer to out explicitly in case we need to display anything.*
* ***session:*** *This is the HttpSession object associated with the session. As sessions are created automatically, this variable is bound even if there is no incoming session reference. The one exception is that if the session attribute of the page directive is used to turn sessions off. In this case, attempts to reference the session variable cause errors at the time the JSP page is translated into a servlet.*
* ***Application****: This is the ServletContext as obtained via the getServletConfig().getContext().*
* ***config****: This is the ServletConfig object for this page.*
* ***pageContext****: JSP introduced a new class called PageContext to encapsulate use of server-specific features like higher performance*
* ***JspWriters****. The advantage is that these can be accessed through this class rather than directly, code will still run on “regular” servlet/JSP engines.*
* ***page****: This is simply a synonym for this, and is not very useful in Java. It was created as a placeholder for the time when the scripting language could be something other than Java.*
* ***exception****: This implicit object applies only to JSP error pages – these are pages to which processing is forwarded when an exception is thrown from another JSP page. They must have the page directive isErrorPage attribute set to true. The implicit exception object is a java.lang.Exception instance that represents the uncaught exception that was thrown from another JSP page and that resulted in the current error page being invoked. The exception object is accessible only from the JSP error page instance to which processing was forwarded when the exception was encountered*

1. Discuss all implicit objects . when will the exception implicit object be enabled?

Refer to point number 2 above.

1. How to disable the scriplets

*You can disable scirpting for a group of JPS pages in an application inone of two ways:*

* *Select the Ignore Java Script check box in the JSP Property Group node of the NetBeans IDE web.xml editor.*
* *Set the scripting-invalid element of the applications deployent descriptor to true. The Scirpting-invalid elementn is the child of jsp-property-group element that defined properties for a set of JSP pages.*

1. How will you declare a instance variable and local variable

Ans<%!  Int k ;%>  and <% int k ; %>

1. How will you declare a method in jsp?(declarative tag)

*We can define methods in JSP using the declaration scripting element as given below:*

*<%!*

*public void print1(){*

*System.out.println(“Hello Im a instance method”)  
 }*

*public static void print2(){*

*System.out.println(“Hello Im a Static method”)  
 }*

*%>*

1. How will you disable the session object in JSP

*You can disable session object is a particular JSP by setting the attribute session = false of the page directive.*

1. Is exception object enabled by default

*No. Inored to enable the exception object the attribute isErrorPage should be set to true of the page directive*

1. Which attribute of page directive can be used multiple times

*Of all the attributes of page directive the import attribute can be used many time. It makes sense as we might need to import many classes or interfaces however other settings values need to be given only once*.

**Lesson 7 : JSP Directives**

1. Discuss Page directive and all it’s attributes

***JSP Page Directive:***

* *The <%@ page %> directive applies to an entire JSP file and any of its static include files, which together are called a translation unit. A static include file is a file whose content becomes part of the calling JSP file. The <%@ page %> directive does not apply to any dynamic include files.*
* *The <%@ page %> directive can be used more than once in a translation unit. However, each attribute, except import, can be used once. Since in Java programming language, the import attribute is similar to the import statement, a <%@ page %> directive with import more than once can be used in a JSP file or translation unit.*
* *No matter the position of <%@ page %> directive in a JSP file or included files, it applies to the entire translation unit. However, it is often good programming style to place it at the top of the JSP file.*

***Attributes of JSP Page directive:***

* ***language="java"*** *: It indicates the scripting language used in scriptlets, declarations, and expressions in the JSP file and any included files. In this release, the only allowed value is java.*
* ***extends="package.class"*** *: It indicates the fully qualified name of the superclass of the Java class file this JSP file will be compiled to. Use this attribute cautiously, as it can limit the JSP container’s ability to provide a specialized superclass that improves the quality of the compiled file.*
* ***import="{package.class | package.\* },*** *..." : It indicates a comma-separated list of Java packages that the JSP file should import. The packages (and their classes) are available to scriptlets, expressions, and declarations within the JSP file. If more than one package to be imported, then specify a comma-separated list after import or use import more than once in a JSP file. The following packages are implicitly imported, so it does not need to specify them with the import attribute: java.lang.\* , javax.servlet.\*, javax.servlet.jsp.\*, javax.servlet.http.\* Place the import attribute before the element that calls the imported class. The import attribute is the only one that is allowed to appear multiple times*
* ***autoFlush*** *= true|false : indicates to the server to flush the buffer to the outputstream. If set to false flush will not happen and an exception will get raise on buffer overflow.\*
* ***buffer=”8kb”*** *= can be used to set the buffer size*
* ***contentType=”MIME”*** *= is used to set the current response MIME type*
* ***isELIgnored = “true|false”****is used to disable Expression Language for the current JPS page. All the expression language will be treated as text by the browser*
* ***isErrorPage = “true|false”*** *is used to tell the ServletContainer that the current JSP is a centralized error handling JSP and hence make the implicit exception object available of the current JPS page*
* ***errorPage = “somejsp.jsp”*** *= tells the ServletContainer the if any uncaught exception gets raised in the current page invoke somejsp.jsp and let the exception object be made available in the next jsp*
* ***isTreadSage = “true|false”****= is used to tell the ServletContainer that the current jsp and hence the translated servlet is not thread safe. On this the server uses single thread model on the generated servlet enabling only one thread to execute the \_jspservice at a given time*
* ***session=”true|false” =*** *if is true marks the jsp is going to participate in the current active session tracking. If false marks otherwise. If true the implicit session object is available to the jsp if false its not available. By default its true*

1. ,include directive with it’s attributes(types of files that can be included,include process-static, and in which phase it is included) a

*The JSP Include Directive:*

* *The <%@ include %> directive inserts a file of text or code in a JSP file at translation time, when the JSP file is compiled. When the <%@ include %> directive is used, the include process is static.*
* *A static include means that the text of the included file is added to the JSP file. The included file can be a JSP file, HTML file, or text file. If the included file is a JSP file, then its JSP elements are parsed and their results are included (along with any other text) in the JSP file.*
* *We can only use include to include static files.*
* *This means that the parsed result of the included file is added to the JSP file where the <%@ include %> directive is placed.*
* *Once the included file is parsed and included, processing resumes with the next line of the calling JSP file.*
* *A JSP container can include a mechanism for being notified if an included file changes, so the container can recompile the JSP page. However, the JSP 2.1 specification does not have a way of directing the JSP container that included files have changed.*
* *The included file must not contain <html>, </html>, <body>, or </body> tags. Since the entire content of the included file is added at that location in the JSP file, these tags would conflict with the same tags in the calling JSP file, causing an error.*

*<%@ include file="relativeURL" %> file="relativeURL"*

*The pathname to the included file is always a relative URL. A relative URL is just the path segment of an URL, without a protocol, port, or domain name. An example is shown below*

*"error.jsp" , "/templates/onlinestore.html" , "/beans/calendar.jsp"*

* *If the relative URL starts with /, then the path is relative to the JSP application’s context, which is a javax.servlet.ServletContext object that is in turn stored in the application object. If the relative URL starts with a directory or file name, then the path is relative to the JSP file.*
* *The example on the slide shows how we can add header and footer to our include.jsp page by making use of the include directive to include header.html and footer.html.*
* *Note that the include directive inserts the files at the time the page is translated. Hence, if the header and footer changes, then it needs to re-translate all the JSP pages that refer to it. If, however, the included files changed more often, then the jsp:include action should be used instead. This action includes the file at the time the JSP page is requested, and is discussed in the next lesson.*

1. Discuss isErrorPage and errorPage attribute of page directive

* ***isErrorPage = “true|false”*** *is used to tell the ServletContainer that the current JSP is a centralized error handling JSP and hence make the implicit exception object available of the current JPS page*
* ***errorPage = “somejsp.jsp”*** *= tells the ServletContainer the if any uncaught exception gets raised in the current page invoke somejsp.jsp and let the exception object be made available in the next jsp*

1. Which attributes of page directive takes Boolean value?

*Refer to Point no 1.*

1. What is the use of contentType attribute of page directive

*Refer to point no 1.*

1. Which attribute of page directive can be repeated(import)

*Refer to point no 1.*

1. What will happen if session attribute of page is set to false?

*Refer to point no 1.*

1. What is the use of contentType attribute of page directive

*Refer to point no 1.*

**Lesson 8 : JSP Action**

1. [%@include file="header.jsp" %](mailto:%25@include%20file=%22header.jsp%22%20%25)- recap of syntax
2. <jsp:forward page="shopcart.jsp">
   1. <jsp:param value='<%= request.getParameter("category") %>' name="mcategory"/>
   2. </jsp:forward>
3. <jsp:include page="names.jsp" >

<jsp:param name="fruit" value="apple" />

</jsp:include>

* + 1. ---
    2. <%= request.getParameter("fruit") %>

1. Diff between include directive and include action

*The JSP Include Action versus JSP Include Directive:*

* *Unlike the include directive, which inserts the file at the time the JSP page is “translated” into a servlet, the include action inserts the file at the time the page is “requested”. This pays a small penalty in efficiency, and precludes the included page from containing general JSP code (for example, it cannot set HTTP headers), but it gains significantly in flexibility.*
* *A JSP container can include a mechanism for being notified if an included file changes (in case of include directive), so the container can recompile the JSP page. However, the JSP 2.2 specification does not have a way of directing the JSP container that included files that have changed.*
* *In the recent versions of the Server containers (such as JBOSS, Tomcat, Websphere Application server), this mechanism has been implemented. Hence there is no change in the behavior of include directive and include action. Both include the files dynamically, hence change in the included file is reflected in the original page in both the cases. However, it is recommended that we should not rely on the container’s behavior and always use include action whenever a dynamic include is required.*

***The jsp:forward Action:***

* *The<jsp:forward> element forwards the request object containing the client request information from one JSP file to another file. The target file can be an HTML file, another JSP file, or a servlet, as long as it is in the same application context as the forwarding JSP file. The lines in the source JSP file after the*
* *<jsp:forward> element are not processed.*
* *We can pass parameter names and values to the target file by using a <jsp:param> clause. An example of this would be passing the parameter name username (with name="username") and the value scott (with value="scott") to a jsp login file as part of the request. If we use <jsp:param>, then the target file should be a dynamic file that can handle the parameters.*
* *Be careful while using <jsp:forward> with unbuffered output. If the <%@ page %> directive has used with buffer=none to specify that the output of the JSP file should not be buffered, and if the JSP file has any data in the out object, using <jsp:forward> will cause an IllegalStateException.*

***The jsp:forward Action:***

* *Like jsp:include, the value for the page attribute in case of jsp:forward is a relative URL or an expression representing the file to which we are forwarding the request. The file can be another JSP file, a servlet, or any other dynamic file that can handle a request object.*
* *The jsp:param element is similar to the one that we saw for jsp:include action.*

**Lesson 9 : JSP Standard Tag Library (JSTL) and EL**

1. Jstl- set and out tags, foreach  discuss with example (discuss all attributes)

*The set tag sets the value of an EL variable or the property of an EL variable in any of the JSP scopes (page, request, session, or application). If the variable does not already exist, it is created.*

*These tags exist as a good alternative to embedding a Java for, while, or do-while loop via a scriptlet. The <c:forEach> tag is a commonly used tag because it iterates over a collection of objects. The <c:forTokens> tag is used to break a string into tokens and iterate through each of the tokens.*

|  |  |  |  |
| --- | --- | --- | --- |
| ***Attribute*** | ***Description*** | ***Required*** | ***Default*** |
| *Items* | *Information to loop over* | *No* | *None* |
| *Begin* | *Element to start with (0 = first item, 1 = second item, ...)* | *No* | *0* |
| *End* | *Element to end with (0 = first item, 1 = second item, ...)* | *No* | *Last element* |
| *Step* | *Process every step items* | *No* | *1* |
| *Var* | *Name of the variable to expose the current item* | *No* | *None* |
| *varStatus* | *Name of the variable to expose the loop status* | *No* | *None* |

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

*<html>*

*<head>*

*<title><c:forEach> Tag Example</title>*

*</head>*

*<body>*

*<c:forEach var = "i" begin = "1" end = "5">*

*Item <c:out value = "${i}"/><p>*

*</c:forEach>*

*</body>*

*</html>*

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

*<html>*

*<head>*

*<title><c:forTokens> Tag Example</title>*

*</head>*

*<body>*

*<c:forTokens items = "Eric,Sam,Peter" delims = "," var = "name">*

*<c:out value = "${name}"/><p>*

*</c:forTokens>*

*</body>*

*</html>*

*The <c:out> tag displays the result of an expression. This is almost similar to the way <%= %> works. The difference here is that <c:out> tag lets you use the simpler "." notation to access properties. For example, to access customer.address.street, use the tag <c:out value = "customer.address.street"/>.*

*The <c:out> tag can automatically escape XML tags so they aren't evaluated as actual tags.*

*The <c:out> tag has the following attributes –*

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Required** | **Default** |
| Value | Information to output | Yes | None |
| Default | Fallback information to output | No | body |
| escapeXml | True if the tag should escape special XML characters | No | true |

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

*<html>*

*<head>*

*<title> <c:out> Tag Example</title>*

*</head>*

*<body>*

*<c:out value = "${'<tag> , &'}"/>*

*</body>*

*</html>*

*The****<c:set>****tag is JSTL-friendly version of the****setProperty****action. The tag is helpful because it evaluates an expression and uses the results to set a value of a JavaBean or a****java.util.Map object****.*

*Attribute*

*The****<c:set>****tag has the following attributes −*

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **Required** | **Default** |
| Value | Information to save | No | body |
| Target | Name of the variable whose property should be modified | No | None |
| property | Property to modify | No | None |
| Var | Name of the variable to store information | No | None |
| Scope | Scope of variable to store information | No | Page |

*If target is specified, property must also be specified.*

*Example*

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

*<html>*

*<head>*

*<title><c:set> Tag Example</title>*

*</head>*

*<body>*

*<c:set var = "salary" scope = "session" value = "${2000\*2}"/>*

*<c:out value = "${salary}"/>*

*</body>*

*</html>*

*• The JSP EL variable or property can be set from the attribute value as shown*

*below:*

1. Discuss EL expressions

*JSP Expression Language (EL) makes it possible to easily access application data stored in JavaBeans components. JSP EL allows you to create expressions both****(a)****arithmetic and****(b)****logical. Within a JSP EL expression, you can use****integers, floating point numbers, strings, the built-in constants true and false****for boolean values, and null.*

*Simple Syntax*

*Typically, when you specify an attribute value in a JSP tag, you simply use a string. For example −*

*<jsp:setProperty name = "box" property = "perimeter" value = "100"/>*

*JSP EL allows you to specify an expression for any of these attribute values. A simple syntax for JSP EL is as follows −*

*${expr}*

*Here****expr****specifies the expression itself. The most common operators in JSP EL are****.****and****[]****. These two operators allow you to access various attributes of Java Beans and built-in JSP objects.*

*For example, the above syntax****<jsp:setProperty>****tag can be written with an expression like −*

*<jsp:setProperty name = "box" property = "perimeter"*

*value = "${2\*box.width+2\*box.height}"/>*

*When the JSP compiler sees the****${}****form in an attribute, it generates code to evaluate the expression and substitues the value of expresson.*

*You can also use the JSP EL expressions within template text for a tag. For example, the****<jsp:text>****tag simply inserts its content within the body of a JSP. The following****<jsp:text>****declaration inserts****<h1>Hello JSP!</h1>****into the JSP output −*

*<jsp:text>*

*<h1>Hello JSP!</h1>*

*</jsp:text>*

*You can now include a JSP EL expression in the body of a****<jsp:text>****tag (or any other tag) with the same****${}****syntax you use for attributes. For example −*

*<jsp:text>*

*Box Perimeter is: ${2\*box.width + 2\*box.height}*

*</jsp:text>*

*EL expressions can use parentheses to group subexpressions. For example,****${(1 + 2) \* 3} equals 9, but ${1 + (2 \* 3)} equals 7****.*

*To deactivate the evaluation of EL expressions, we specify the****isELIgnored****attribute of the page directive as below −*

*<%@ page isELIgnored = "true|false" %>*

*The valid values of this attribute are true and false. If it is true, EL expressions are ignored when they appear in static text or tag attributes. If it is false, EL expressions are evaluated by the container.*

*Expression Language:*

* *Expression Language is introduced with JSP 2.0, and many more capabilities are added with JSP 2.1.*
* *Expression Language is used with JSTL to simplify the presentation layer.*
* *Expression Language replaces the action tags <jsp:useBean> <jsp:getProperty> used to access Java beans properties with short and readable expressions.*
* *Express Language uses the form ${expr} to access and specify an expression.*

*Expression Language Capabilities:*

* *EL provides concise access to stored objects: to output a “scoped variable”named saleItem, use ${saleItem}*
* *It facilitates shorthand notation for bean properties: To output the companyName property of a scoped variable named company, use ${company.companyName}.*
* *It facilitates simple access to collection elements: To access an element of an array, List or Map, use ${variable[indexorKey]}*
* *It provides access to request parameters, cookies, and other request data: To access the standard types of request data, we can use one of the implicit objects mentioned in first lesson.*
* *It provides a small but useful set of simple operators.*