JSP Interview Questions and Answers

What is JSP and why do we need it?

JSP stands for JavaServer Pages. JSP is java server side technology to create dynamic web pages. JSP is extension of Servlet technology to help developers create dynamic pages with HTML like syntax.

We can create user views in servlet also but the code will become very ugly and error prone. Also most of the elements in web page is static, so JSP page is more suitable for web pages. We should avoid business logic in JSP pages and try to use it only for view purpose. JSP scripting elements can be used for writing java code in JSP pages but it's best to avoid them and use JSP action elements, JSTL tags or custom tags to achieve the same functionalities.

One more benefit of JSP is that most of the containers support hot deployment of JSP pages. Just make the required changes in the JSP page and replace the old page with the updated jsp page in deployment directory and container will load the new JSP page. We don't need to compile our project code or restart server whereas if we make change in servlet code, we need to build the complete project again and deploy it. Although most of the containers now provide hot deployment support for applications but still it's more work that JSP pages.

What are the JSP lifecycle phases?

If you will look into JSP page code, it looks like HTML and doesn't look anything like java classes. Actually JSP container takes care of translating the JSP pages and create the servlet class that is used in web application. JSP lifecycle phases are:

Translation – JSP container checks the JSP page code and parse it to generate the servlet source code. For example in Tomcat you will find generated servlet class files at TOMCAT/work/Catalina/localhost/WEBAPP/org/apache/jsp directory. If the JSP page name is home.jsp, usually the generated servlet class name is home_jsp and file name is home_jsp.java

Compilation – JSP container compiles the jsp class source code and produce class file in this phase.

Class Loading – Container loads the class into memory in this phase.

Instantiation – Container invokes the no-args constructor of generated class to load it into memory and instantiate it.

Initialization – Container invokes the init method of JSP class object and initializes the servlet config with init params configured in deployment descriptor. After this phase, JSP is ready to handle client requests. Usually from translation to initialization of JSP happens when first request for JSP comes but we can configure it to be loaded and initialized at the time of deployment like servlets using load-on-startup element.

Request Processing – This is the longest lifecycle of JSP page and JSP page processes the client requests. The processing is multi-threaded and similar to servlets and for every request a new thread is spawned and ServletRequest and ServletResponse object is created and JSP service method is invoked.

Destroy – This is the last phase of JSP lifecycle where JSP class is unloaded from memory. Usually it happens when application is undeployed or the server is shut down.

What are JSP lifecycle methods?

JSP lifecycle methods are:

jspInit(): This method is declared in JspPage and it's implemented by JSP container implementations. This method is called once in the JSP lifecycle to initialize it with config params configured in deployment descriptor. We can override this method using JSP declaration scripting element to initialize any resources that we want to use in JSP page.

_jspService(): This is the JSP method that gets invoked by JSP container for each client request by passing request and response object. Notice that method name starts with underscore to distinguish it from other lifecycle methods because we can't override this method. All the JSP code goes inside this method and it's overridden by default. We should not try to override it using JSP declaration scripting element. This method is defined in HttpJspPage interface.

jspDestroy(): This method is called by container when JSP is unloaded from memory such as shutting down application or container. This method is called only once in JSP lifecycle and we should override this method to release any resources created in JSP init method.

Which JSP lifecycle methods can be overridden?

We can override jspInit() and jspDestroy() methods using JSP declaration scripting element. We should override jspInit() methods to create common resources that we would like to use in JSP service method and override jspDestroy() method to release the common resources.

How can we avoid direct access of JSP pages from client browser?

We know that anything inside WEB-INF directory can't be accessed directly in web application, so we can place our JSP pages in WEB-INF directory to avoid direct access to JSP page from client browser. But in this case, we will have to configure it in deployment descriptor just like Servlets. Sample configuration is given below code snippet of web.xml file.

```
<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>
```

What are different types of comments in JSP?

JSP pages provide two types of comments that we can use:

HTML Comments: Since JSP pages are like HTML, we can use HTML comments like <-- HTML Comment -->. These comments are sent to client also and we can see it in HTML source. So we should avoid any code level comments or debugging comments using HTML comments.
JSP Comments: JSP Comments are written using scriptlets like <%-- JSP Comment --%>. These comments are present in the generated servlet source code and doesn't sent to client. For any code level or debugging information comments we should use JSP comments.

What is Scriptlet, Expression and Declaration in JSP?

Scriptlets, Expression and Declaration are scripting elements in JSP page using which we can add java code in the JSP pages.

A scriptlet tag starts with <% and ends with %>. Any code written inside the scriptlet tags go into the _jspService() method. For example;

```
<%
Date d = new Date();
System.out.println("Current Date="+d);
%>
```

Since most of the times we print dynamic data in JSP page using out.print() method, there is a shortcut to do this through JSP Expressions. JSP Expression starts with <%= and ends with %>.

<% out.print("Pankaj"); %> can be written using JSP Expression as <%= "Pankaj" %>

Notice that anything between <%= %> is sent as parameter to out.print() method. Also notice that scriptlets can contain multiple java statements and always ends with semicolon (;) but expression doesn't end with semicolon.

JSP Declarations are used to declare member methods and variables of servlet class. JSP Declarations starts with <%! and ends with %>.
For example we can create an int variable in JSP at class level as <%! public static int count=0; %>.

Can we use JSP implicit objects in a method defined in JSP Declaration?

No we can't because JSP implicit objects are local to service method and added by JSP Container while translating JSP page to servlet source code. JSP Declarations code goes outside the service method and used to create class level variables and methods and hence can't use JSP implicit objects.

Which implicit object is not available in normal JSP pages?

JSP exception implicit object is not available in normal JSP pages and it's used in JSP error pages only to catch the exception thrown by the JSP pages and provide useful message to the client

What are JSP implicit objects?

JSP implicit objects are created by container while translating JSP page to Servlet source to help developers. We can use these objects directly in scriptlets that goes in service method, however we can't use them in JSP Declaration because that code will go at class level.

We have 9 implicit objects that we can directly use in JSP page. Seven of them are declared as local variable at the start of _jspService() method whereas two of them are part of _jspService() method argument that we can use.

- out Object
- request Object
- response Object
- config Object
- application Object
- session Object
- pageContext Object
- page Object
- exception Object
- JSP Implicit Objects Example

Read in detail about each one of them at JSP Implicit Objects.

What are the benefits of PageContext implicit object?

JSP pageContext implicit object is instance of javax.servlet.jsp.PageContext abstract class implementation. We can use pageContext to get and set attributes with different scopes and to forward request to other resources. pageContext object also hold reference to other implicit object.

This is the only object that is common in both JSP implicit objects and in JSP EL implicit objects.

How do we configure init params for JSP?

We can configure init params for JSP similar to servlet in web.xml file, we need to configure JSP init params with servlet and servlet-mapping element. The only thing differs from servlet is jsp-file element where we need to provide the JSP page location.

Why use of scripting elements in JSP is discouraged?

JSP pages are mostly used for view purposes and all the business logic should be in the servlet or model classes. We should pass parameters to JSP page through attributes and then use them to create the HTML response in JSP page.

Most part of the JSP page contains HTML code and to help web designers to easily understand JSP page and develop them, JSP technology provides action elements, JSP EL, JSP Standard Tag Library and custom tags that we should use rather than scripting elements to bridge the gap between JSP HTML part and JSP java part.

Can we define a class in a JSP Page?

It’s not a good practice though, but we can define a class inside a JSP Page. Below is the sample code for this:

```
<%!
private static class NestedClass { //static is better because Servlet is multi-threaded
  private final int num = 0;
  public int getNum() {
    return num;
  }
}
%>

<%
    class Person {
        //this will go inside method body, so can't be public
    }
%>
```

How can we disable java code or scripting in JSP page?

We can disable scripting elements in JSP pages through deployment descriptor configuration like below.

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <scripting-invalid>true</scripting-invalid>
    </jsp-property-group>
</jsp-config>
```

Above url-pattern will disable scripting for all the JSP pages but if you want to disable it only for specific page, you can give the JSP file name itself.

Explain JSP Action Elements or Action Tags?

JSP action elements or action tags are HTML like tags that provide useful functionalities such as working with Java Bean, including a resource, forwarding the request and to generate dynamic XML elements. JSP action elements always starts with jsp: and we can use them in JSP page directly without the need to import any tag libraries or any other configuration changes. Some of the important action elements are jsp:useBean, jsp:getProperty, jsp:setProperty, jsp:include and jsp:forward.

Read more in details about these at JSP Action Elements.

What is difference between include directive and jsp:include action?

The difference between JSP include directive and include action is that in include directive the content to other resource is added to the generated servlet code at the time of translation whereas with include action it happens at runtime.

Another difference is that in JSP include action, we can pass params to be used in the included resource with jsp:param action element but in JSP include directive we can't pass any params.

When included resource is static such as header, footer, image files then we should use include directive for faster performance but if the included resource is dynamic and requires some parameters for processing then we should use include action tag.

What is JSP Expression Language and what are it's benefits?

Most of the times we use JSP for view purposes and all the business logic is present in servlet code or model classes. When we receive client request in servlet, we process it and then add attributes in request/session/context scope to be retrieved in JSP code. We also use request params, headers, cookies and init params in JSP to create response views.

We can use scriptlets and JSP expressions to retrieve attributes and parameters in JSP with java code and use it for view purpose. But for web designers, java code is hard to understand and that's why JSP Specs 2.0 introduced Expression Language (EL) through which we can get attributes and parameters easily using HTML like tags.

Expression language syntax is ${name} and we can use EL implicit objects and EL operators to retrieve the attributes from different scopes and use them in JSP page.

Read more about JSP EL with example program at JSP EL Tutorial.

How to use JSP EL to get HTTP method name?

We can use pageContext JSP EL implicit object to get the request object reference and use dot operator to get the HTTP method name in JSP page. The JSP EL code for this will be ${pageContext.request.method}.

What is JSP Standard Tag Library, provide some example usage?

JSP Standard Tag Library or JSTL is more versatile than JSP EL or Action elements because we can loop through a collection or escape HTML tags to show them like text in response.
JSTL is part of the Java EE API and included in most servlet containers. But to use JSTL in our JSP pages, we need to download the JSTL jars for your servlet container. Most of the times, you can find them in the example projects and you can use them. You need to include these libraries in the project WEB-INF/lib directory. These jars are container specific, for example in Tomcat, we need to include jstl.jar and standard.jar jar files in project build path.

Read more about JSTL tags with example program at JSTL Tutorial.

What are JSP EL implicit objects and how it's different from JSP implicit Objects?

JSP Expression Language provides many implicit objects that we can use to get attributes from different scopes and parameter values. Note that these are different from JSP implicit objects and contains only the attributes in given scope. The only common implicit object in JSP EL and JSP page is pageContext object.

Below table provides list of implicit object in JSP EL.

JSP EL Implicit Objects	Type	Description
pageScope	Map	A map that contains the attributes set with page scope.
requestScope	Map	Used to get the attribute value with request scope.
sessionScope	Map	Used to get the attribute value with session scope.
applicationScope	Map	Used to get the attributes value from application scope.
param	Map	Used to get the request parameter value, returns a single value
paramValues	Map	Used to get the request param values in an array, useful when request parameter contain multiple values.
header	Map	Used to get request header information.
headerValues	Map	Used to get header values in an array.
cookie	Map	Used to get the cookie value in the JSP
initParam	Map	Used to get the context init params, we can't use it for servlet init params
pageContext	pageContext	Same as JSP implicit pageContext object, used to get the request, session references etc. example usage is getting request HTTP Method name.

What are the types of JSTL tags?

Based on the JSTL functions, they are categorized into five types.

Core Tags – Core tags provide support for iteration, conditional logic, catch exception, url, forward or redirect response etc.

Formatting and Localization Tags – These tags are provided for formatting of Numbers, Dates and i18n support through locales and resource bundles.

SQL Tags – JSTL SQL Tags provide support for interaction with relational databases such as Oracle, MySql etc.

XML Tags – XML tags are used to work with XML documents such as parsing XML, transforming XML data and XPath expressions evaluation.

JSTL Functions Tags – JSTL tags provide a number of functions that we can use to perform common operation, most of them are for String manipulation such as String Concatenation, Split String etc.

What is JSP Custom Tag and what are it's components?

Sometimes JSP EL, Action Tags and JSTL tags are not enough and we might get tempted to write java code to perform some operations in JSP page. Fortunately JSP is extendable and we can create our own custom tags to perform certain operations.

We can create JSP Custom Tags with following components:

JSP Custom Tag Handler
Creating Tag Library Descriptor (TLD) File
Deployment Descriptor Configuration for TLD

We can add custom tag library in JSP page using taglib directive and then use it.

Give an example where you need JSP Custom Tag?

Let's say we want to show a number with formatting with commas and spaces. This can be very useful for user when the number is really long. So we want some custom tags like below:

<mytags:formatNumber number="123456.789" format="#,###.00"/>

Based on the number and format passed, it should write the formatted number in JSP page, for above example it should print 123,456.79

We know that JSTL doesn't provide any inbuilt tags to achieve this, so we will create our own custom tag implementation and use it in the JSP page.

Read above example implementation at JSP Custom Tag.

How do we catch exception and process it using JSTL?

We can use JSTL Core tags c:catch and c:if to catch exception inside the JSP service method and process it. c:catch tag catches the exception and wraps it into the exception variable and we can use c:if condition tag to process it. Below code snippet provide sample usage.

```
<c:catch var ="exception">
   <% int x = 5/0;%>
</c:catch>

<c:if test = "${exception ne null}">
   <p>Exception is : ${exception} <br />
   Exception Message: ${exception.message}</p>
</c:if>
```

Notice the use of JSP EL in the c:if condition.

Why don’t we need to configure JSP standard tags in web.xml?

We don’t need to configure JSP standard tags in web.xml because the TLD files are inside the META-INF directory of the JSTL jar files. When container loads the web application and find TLD files inside the META-INF directory of JAR file, it automatically configures them to be used directly in the application JSP pages. All we need to do it to include it in the JSP page using taglib directive.

How can we handle exceptions thrown by JSP service method?

To handle exceptions thrown by the JSP page, all we need is an error page and define the error page in JSP using page directive.

To create a JSP error page, we need to set page directive attribute isErrorPage value to true, then we can access exception implicit object in the JSP and use it to send customized error message to the client.

We need to define exception and error handler JSP pages in the deployment descriptor like below.

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

<error-page>
    <exception-type>java.lang.Throwable</exception-type>
    <location>/error.jsp</location>
</error-page>
```

Read more with example program at JSP Exception Handling.

How to ignore the EL expression evaluation in a JSP?

We can ignore EL evaluation in JSP page by two ways.

Using page directive as <%@ page isELIgnored="true" %>
Configuring in web.xml – better approach when you want to disable EL evaluation for many JSP pages.

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <el-ignored>true</el-ignored>
    </jsp-property-group>
</jsp-config>
```

**How do we print “
 creates a new line in HTML” in JSP?**

We can use c:out escapeXml attribute to escape the HTML elements so that it get’s shown as text in the browser, for this scenario we will write code like below.

```
<c:out value="<br> creates a new line in HTML" escapeXml="true"></c:out>
```

What is jsp-config in deployment descriptor?

jsp-config element is used to configure different parameters for JSP pages. Some of it’s usage are:

Configuring tag libraries for the web application like below.

```
<jsp-config>
    <taglib>
        <taglib-uri>http://journaldev.com/jsp/tlds/mytags</taglib-uri>
        <taglib-location>/WEB-INF/numberformatter.tld</taglib-location>
    </taglib>
</jsp-config>
```

We can control scripting elements in JSP pages.
We can control JSP Expression Language (EL) evaluation in JSP pages.
We can define the page encoding for URL pattern.
To define the buffer size to be used in JSP page out object.
To denote that the group of resources that match the URL pattern are JSP documents, and thus must be interpreted as XML documents.

Can we use JavaScript with JSP Pages?

Yes why not, I have seen some developers getting confused with this. Even though JSP is a server side technology, it’s used to generate client side response and we can add javascript or CSS code like any other HTML page.

How can we prevent implicit session creation in JSP?

By default JSP page creates a session but sometimes we don’t need session in JSP page. We can use JSP page directive session attribute to indicate compiler to not create session by default. It’s default value is true and session is created. To disable the session creation, we can use it like below.

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

What is difference between JspWriter and Servlet PrintWriter?

PrintWriter is the actual object responsible for writing the content in response. JspWriter uses the PrintWriter object behind the scene and provide buffer support. When the buffer is full or flushed, JspWriter uses the PrintWriter object to write the content into response.

When will Container initialize multiple JSP/Servlet Objects?

If we have multiple servlet and servlet-mapping elements in deployment descriptor for a single servlet or JSP page, then container will initialize an object for each of the element and all of these instances will have their own ServletConfig object and init params.

For example, if we configure a single JSP page in web.xml like below.

```
<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>
```

Then if we can access same JSP page with both the URI pattern and both will have their own init params values.

How can we extend JSP technology?

We can extend JSP technology with custom tags to avoid scripting elements and java code in JSP pages.

Provide some JSP Best Practices?

Some of the JSP best practices are:

Avoid scripting elements in JSP pages. If JSP EL, action elements and JSTL not serve your needs then create custom tags.

Use comment properly, use JSP comments for code level or debugging purpose so that it’s not sent to client.

Avoid any business logic in JSP page, JSP pages should be used only for response generation for client.

Disable session creation in JSP page where you don’t need it for better performance.

Use page, taglib directives at the start of JSP page for better readability.

Proper use of jsp include directive or include action based on your requirements, include directive is good for static content whereas include action is good for dynamic content and including resource at runtime.

Proper exception handling using JSP error pages to avoid sending container generated response incase JSP pages throw exception in service method.

If you are having CSS and JavaScript code in JSP pages, it’s best to place them in separate files and include them in JSP page.

Most of the times JSTL is enough for our needs, if you find a scenario where it’s not then check your application design and try to put the logic in a servlet that will do the processing and then set attributes to be used in JSP pages.