1. **What are the advantages of JSP over Servlet?**

JSP is a server side technology to make content generation a simple appear. The advantage of JSP is that they are document-centric. Servlets, on the other hand, look and act like programs. A Java Server Page can contain Java program fragments that instantiate and execute Java classes, but these occur inside an HTML template file and are primarily used to generate dynamic content. Some of the JSP functionality can be achieved on the client, using JavaScript. The power of JSP is that it is server-based and provides a framework for Web application development.

**2. What is the life-cycle of JSP?**

When a request is mapped to a JSP page for the first time, it translates the JSP page into a servlet class and compiles the class. It is this servlet that services the client requests.   
A JSP page has seven phases in its lifecycle, as listed below in the sequence of occurrence:

* Translation
* Compilation
* Loading the class
* Instantiating the class
* jspInit() invocation
* \_jspService() invocation
* jspDestroy() invocation

**3. What is the jspInit() method?**

The jspInit() method of the javax.servlet.jsp.JspPage interface is similar to the init() method of servlets. This method is invoked by the container only once when a JSP page is initialized. It can be overridden by a page author to initialize resources such as database and network connections, and to allow a JSP page to read persistent configuration data.

**4. What is the \_jspService() method?**

SThe \_jspService() method of the javax.servlet.jsp.HttpJspPage interface is invoked every time a new request comes to a JSP page. This method takes the HttpServletRequest and HttpServletResponse objects as its arguments. A page author cannot override this method, as its implementation is provided by the container.

**5.What is the jspDestroy() method?**

The jspDestroy() method of the javax.servlet.jsp.JspPage interface is invoked by the container when a JSP page is about to be destroyed. This method is similar to the destroy() method of servlets. It can be overridden by a page author to perform any cleanup operation such as closing a database connection.

**6.What JSP lifecycle methods can I override?**

You cannot override the \_jspService() method within a JSP page. You can however, override the jspInit() and jspDestroy() methods within a JSP page. jspInit() can be useful for allocating resources like database connections, network connections, and so forth for the JSP page. It is good programming practice to free any allocated resources within jspDestroy().

**7.How can I override the jspInit() and jspDestroy() methods within a JSP page?**

The jspInit() and jspDestroy() methods are each executed just once during the lifecycle of a JSP page and are typically declared as JSP declarations:

<%!

public void jspInit() {

. . .

}

%>

<%!

public void jspDestroy() {

. . .

}

%>

**8. What are implicit objects in JSP?**

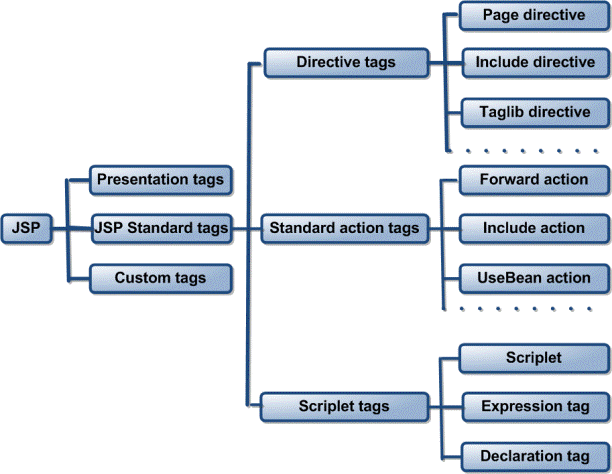
Implicit objects in JSP are the Java objects that the JSP Container makes available to developers in each page. These objects need not be declared or instantiated by the JSP author. They are automatically instantiated by the container and are accessed using standard variables; hence, they are called implicit objects.The implicit objects available in JSP are as follows:

* request
* response
* pageContext
* session
* application
* out
* config
* page
* exception

The implicit objects are parsed by the container and inserted into the generated servlet code. They are available only within the jspService method and not in any declaration.

**9. What are the different types of JSP tags?**

The different types of JSP tags are as follows:

  
  
**10. What are JSP directives?**

* JSP directives are messages for the JSP engine. i.e., JSP directives serve as a message from a JSP page to the JSP container and control the processing of the entire page
* They are used to set global values such as a class declaration, method implementation, output content type, etc.
* They do not produce any output to the client.
* Directives are always enclosed within <%@ ….. %> tag.
* Ex: page directive, include directive, etc.

**11. What is page directive?**

* A page directive is to inform the JSP engine about the headers or facilities that page should get from the environment.
* Typically, the page directive is found at the top of almost all of our JSP pages.
* There can be any number of page directives within a JSP page (although the attribute – value pair must be unique).
* The syntax of the include directive is: <%@ page attribute="value">
* Example:<%@ include file="header.jsp" %>

**12. What are the attributes of page directive?**

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

* **import**: It specifies the packages that are to be imported.
* **session**: It specifies whether a session data is available to the JSP page.
* **contentType**: It allows a user to set the content-type for a page.
* **isELIgnored**: It specifies whether the EL expressions are ignored when a JSP is translated to a servlet.

**13. What is the include directive?**

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

* The include directive is used to statically insert the contents of a resource into the current JSP.
* This enables a user to reuse the code without duplicating it, and includes the contents of the specified file at the translation time.
* The syntax of the include directive is as follows:  
  <%@ include file = "FileName" %>
* This directive has only one attribute called file that specifies the name of the file to be included.

**14. What are the JSP standard actions?**

* The JSP standard actions affect the overall runtime behavior of a JSP page and also the response sent back to the client.
* They can be used to include a file at the request time, to find or instantiate a JavaBean, to forward a request to a new page, to generate a browser-specific code, etc.
* Ex: include, forward, useBean,etc. object

**15. What are the standard actions available in JSP?**

The standard actions available in JSP are as follows:

* **<jsp:include>**: It includes a response from a servlet or a JSP page into the current page. It differs from an include directive in that it includes a resource at request processing time, whereas the include directive includes a resource at translation time.
* **<jsp:forward>**: It forwards a response from a servlet or a JSP page to another page.
* **<jsp:useBean>**: It makes a JavaBean available to a page and instantiates the bean.
* **<jsp:setProperty>**: It sets the properties for a JavaBean.
* **<jsp:getProperty>**: It gets the value of a property from a JavaBean component and adds it to the response.
* **<jsp:param>**: It is used in conjunction with <jsp:forward>;, <jsp:, or plugin>; to add a parameter to a request. These parameters are provided using the name-value pairs.
* **<jsp:plugin>**: It is used to include a Java applet or a JavaBean in the current JSP page.

**16. What is the <jsp:useBean> standard action?**

The <jsp:useBean> standard action is used to locate an existing JavaBean or to create a JavaBean if it does not exist. It has attributes to identify the object instance, to specify the lifetime of the bean, and to specify the fully qualified classpath and type.

**17. What are the scopes available in <jsp:useBean>?**

The scopes available in <jsp:useBean> are as follows:

* **page scope:**: It specifies that the object will be available for the entire JSP page but not outside the page.
* **request scope**: It specifies that the object will be associated with a particular request and exist as long as the request exists.
* **application scope**: It specifies that the object will be available throughout the entire Web application but not outside the application.
* **session scope**: It specifies that the object will be available throughout the session with a particular client.

**18. What is the <jsp:forward> standard action?**

* The <jsp:forward> standard action forwards a response from a servlet or a JSP page to another page.
* The execution of the current page is stopped and control is transferred to the forwarded page.
* The syntax of the <jsp:forward> standard action is :    
  <jsp:forward page="/targetPage" />  
  Here, targetPage can be a JSP page, an HTML page, or a servlet within the same context.
* If anything is written to the output stream that is not buffered before <jsp:forward>, an IllegalStateException will be thrown.

*Note* : Whenever we intend to use <jsp:forward> or <jsp:include> in a page, buffering should be enabled. By default buffer is enabled.

**19. What is the <jsp:include> standard action?**

The <jsp:include> standard action enables the current JSP page to include a static or a dynamic resource at runtime. In contrast to the include directive, the include action is used for resources that change frequently. The resource to be included must be in the same context.The syntax of the <jsp:include> standard action is as follows:  
<jsp:include page="targetPage" flush="true"/>   
Here, targetPage is the page to be included in the current JSP.

**20. What is the difference between include directive and include action?**

|  |  |
| --- | --- |
| **Include directive** | **Include action** |
| The *include* directive, includes the content of the specified file during the translation phase–when the page is converted to a servlet. | The *include* action, includes the response generated by executing the specified page (a JSP page or a servlet)  during the request processing phase–when the page is requested by a user. |
| The include directive is used to statically insert the contents of a resource into the current JSP. | The include standard action enables the current JSP page to include a static or a dynamic resource at runtime. |
| Use the include directive if the file changes rarely. It’s the fastest mechanism. |  |

**21. Differentiate between pageContext.include and jsp:include?**

The <jsp:include> standard action and the pageContext.include() method are both used to include resources at runtime. However, the pageContext.include() method always flushes the output of the current page before including the other components, whereas <jsp:include> flushes the output of the current page only if the value of flush is explicitly set to true as follows:

<jsp:include page="/index.jsp" flush="true"/>

**22. What is the jsp:setProperty action?**

You use jsp:setProperty to give values to properties of beans that have been referenced earlier. You can do this in two contexts. First, you can use jsp:setProperty after, but outside of, a jsp:useBean element, as below:

<jsp:useBean id="myName" ... />

...

<jsp:setProperty name="myName" property="myProperty" ... />

In this case, the jsp:setProperty is executed regardless of whether a new bean was instantiated or an existing bean was found.   
  
A second context in which jsp:setProperty can appear is inside the body of a jsp:useBean element, as below:

<jsp:useBean id="myName" ... >

...

<jsp:setProperty name="myName"

property="someProperty" ... />

</jsp:useBean>

Here, the jsp:setProperty is executed only if a new object was instantiated, not if an existing one was found.

**23. What is the jsp:getProperty action?**

The <jsp:getProperty> action is used to access the properties of a bean that was set using the <jsp:getProperty> action. The container converts the property to a String as follows:

* If it is an object, it uses the toString() method to convert it to a String.
* If it is a primitive, it converts it directly to a String using the valueOf() method of the corresponding Wrapper class.
* The syntax of the <jsp:getProperty> method is: <jsp:getProperty name="Name" property="Property" />

Here, name is the id of the bean from which the property was set. The property attribute is the property to get. A user must create or locate a bean using the <jsp:useBean> action before using the <jsp:getProperty> action.

**24. What is the <jsp:param> standard action?**

The <jsp:param> standard action is used with <jsp:include> or <jsp:forward> to pass parameter names and values to the target resource. The syntax of the <jsp:param> standard action is as follows:   
<jsp:param name="paramName" value="paramValue"/>

**25. What is the jsp:plugin action ?**

This action lets you insert the browser-specific OBJECT or EMBED element needed to specify that the browser run an applet using the Java plugin.

**26. What are scripting elements?**

JSP scripting elements let you insert Java code into the servlet that will be generated from the current JSP page. There are three forms:

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

**27. What is a scriptlet?**

A scriptlet contains Java code that is executed every time a JSP is invoked. When a JSP is translated to a servlet, the scriptlet code goes into the service() method. Hence, methods and variables written in scriptlets are local to the service() method. A scriptlet is written between the **<% and %>**tags and is executed by the container at request processing time.

**28. What are JSP declarations?**

As the name implies, JSP declarations are used to declare class variables and methods in a JSP page. They are initialized when the class is initialized. Anything defined in a declaration is available for the whole JSP page. A declaration block is enclosed between the **<%! and %>**tags. A declaration is not included in the service() method when a JSP is translated to a servlet.

**29. What is a JSP expression?**

A JSP expression is used to write an output without using the out.print statement. It can be said as a shorthand representation for scriptlets. An expression is written between the **<%= and %>** tags. It is not required to end the expression with a semicolon, as it implicitly adds a semicolon to all the expressions within the expression tags.

**30. How is scripting disabled?**

Scripting is disabled by setting the scripting-invalid element of the deployment descriptor to true. It is a subelement of jsp-property-group. Its valid values are true and false. The syntax for disabling scripting is as follows:

<jsp-property-group>

<url-pattern>\*.jsp</url-pattern>

<scripting-invalid>true</scripting-invalid>

</jsp-property-group>

1. [What is JSP and why do we need it?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-benefits)
2. [What are the JSP lifecycle phases?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-life-phases)
3. [What are JSP lifecycle methods?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-lifecycle-methods)
4. [Which JSP lifecycle methods can be overridden?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-method-override)
5. [How can we avoid direct access of JSP pages from client browser?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-restricted-access)
6. [What are different types of comments in JSP?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-comments)
7. [What is Scriptlet, Expression and Declaration in JSP?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-scriptlet)
8. [What are JSP implicit objects?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-implicit-objects)
9. [Can we use JSP implicit objects in a method defined in JSP Declaration?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-declaration)
10. [Which implicit object is not available in normal JSP pages?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-exception-object)
11. [What are the benefits of PageContext implicit object?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-pagecontext)
12. [How do we configure init params for JSP?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-init-params)
13. [Why use of scripting elements in JSP is discouraged?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-scripting)
14. [Can we define a class in a JSP Page?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-nested-class)
15. [How can we disable java code or scripting in JSP page?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-disable-scripting)
16. [Explain JSP Action Elements or Action Tags?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-actions)
17. [What is difference between include directive and jsp:include action?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-include-action-vs-directive)
18. [What is JSP Expression Language and what are it’s benefits?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-el)
19. [What are JSP EL implicit objects and how it’s different from JSP implicit Objects?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-el-objects)
20. [How to use JSP EL to get HTTP method name?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-el-http-method)
21. [What is JSP Standard Tag Library, provide some example usage?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jstl-example)
22. [What are the types of JSTL tags?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsl-tags)
23. [What is JSP Custom Tag and what are it’s components?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-custom-tags)
24. [Give an example where you need JSP Custom Tag?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-custom-tags-example)
25. [Why don’t we need to configure JSP standard tags in web.xml?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-jstl-configure)
26. [How can we handle exceptions thrown by JSP service method?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-error-page)
27. [How do we catch exception and process it using JSTL?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-catch)
28. [How do we print “<br> creates a new line in HTML” in JSP?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-out-escapexml)
29. [What is jsp-config in deployment descriptor?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-config)
30. [How to ignore the EL expression evaluation in a JSP?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-ignore-el)
31. [When will Container initialize multiple JSP/Servlet Objects?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-multiple-instance)
32. [Can we use JavaScript with JSP Pages?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-javascript)
33. [How can we prevent implicit session creation in JSP?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-session)
34. [What is difference between JspWriter and Servlet PrintWriter?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jspwriter-vs-printwriter)
35. [How can we extend JSP technology?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-extend)
36. [Provide some JSP Best Practices?](http://www.journaldev.com/2110/jsp-interview-questions-and-answers#jsp-best-practices)

=====================================================================================

1. 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.

1. 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:

* 1. **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
  2. **Compilation** – JSP container compiles the jsp class source code and produce class file in this phase.
  3. **Class Loading** – Container loads the class into memory in this phase.
  4. **Instantiation** – Container invokes the no-args constructor of generated class to load it into memory and instantiate it.
  5. **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.
  6. **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.
  7. **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.

1. What are JSP lifecycle methods?

JSP lifecycle methods are:

* 1. **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.
  2. **\_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.
  3. **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.

1. 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.

[sociallocker id=”2713″]

1. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <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> |

1. What are different types of comments in JSP?

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

* 1. **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.
  2. **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.

1. 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;

|  |  |
| --- | --- |
| 1  2  3  4 | <%  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; %>.

1. 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.

* 1. out Object
  2. request Object
  3. response Object
  4. config Object
  5. application Object
  6. session Object
  7. pageContext Object
  8. page Object
  9. exception Object
  10. JSP Implicit Objects Example

Read in detail about each one of them at [**JSP Implicit Objects**](http://www.journaldev.com/2038/jsp-implicit-objects-with-examples).

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <%!  private static class NestedClass { //static is better because Servlet is multi-threaded    private final int num = 0;    public int getNum() {      return num;    }  }  %> |

Or

|  |  |
| --- | --- |
| 1  2  3  4  5 | <%      class Person {          //this will go inside method body, so can't be public      }  %> |

1. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <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.

1. 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**](http://www.journaldev.com/2082/jsp-standard-action-tags-quick-tutorial).

1. 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.

1. 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**](http://www.journaldev.com/2064/jsp-expression-language-el-example-tutorial).

1. 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.

|  |  |  |
| --- | --- | --- |
| **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. |

Below table provides list of implicit object in JSP EL.

1. 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}.

1. 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**](http://www.journaldev.com/2090/jstl-tutorial-with-examples-jstl-core-tags).

1. What are the types of JSTL tags?

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

* 1. Core Tags – Core tags provide support for iteration, conditional logic, catch exception, url, forward or redirect response etc.
  2. Formatting and Localization Tags – These tags are provided for formatting of Numbers, Dates and i18n support through locales and resource bundles.
  3. SQL Tags – JSTL SQL Tags provide support for interaction with relational databases such as Oracle, MySql etc.
  4. XML Tags – XML tags are used to work with XML documents such as parsing XML, transforming XML data and XPath expressions evaluation.
  5. 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.

1. 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:

* 1. JSP Custom Tag Handler
  2. Creating Tag Library Descriptor (TLD) File
  3. Deployment Descriptor Configuration for TLD

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

1. 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**](http://www.journaldev.com/2099/jsp-custom-tags-example-tutorial).

1. 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.

1. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <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**](http://www.journaldev.com/2049/jsp-exception-handling-with-error-pages).

1. 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <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.

1. How do we print “<br> 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.

|  |  |
| --- | --- |
| 1 | <c:out value="<br> creates a new line in HTML" escapeXml="true"></c:out> |

1. 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:

* 1. Configuring tag libraries for the web application like below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <jsp-config>          <taglib>              <taglib-uri>http://journaldev.com/jsp/tlds/mytags</taglib-uri>              <taglib-location>/WEB-INF/numberformatter.tld</taglib-location>          </taglib>  </jsp-config> |

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

1. How to ignore the EL expression evaluation in a JSP?

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

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

1. 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.

|  |  |
| --- | --- |
| 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> |

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

1. 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.

1. 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.

|  |  |
| --- | --- |
| 1 | <%@ page session ="false" %> |

1. 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.

1. How can we extend JSP technology?

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

1. Provide some JSP Best Practices?

Some of the JSP best practices are:

* 1. Avoid scripting elements in JSP pages. If JSP EL, action elements and JSTL not serve your needs then create custom tags.
  2. Use comment properly, use JSP comments for code level or debugging purpose so that it’s not sent to client.
  3. Avoid any business logic in JSP page, JSP pages should be used only for response generation for client.
  4. Disable session creation in JSP page where you don’t need it for better performance.
  5. Use page, taglib directives at the start of JSP page for better readability.
  6. 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.
  7. Proper exception handling using JSP error pages to avoid sending container generated response incase JSP pages throw exception in service method.
  8. 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.
  9. 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.