# 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.

# 2)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.

# 3)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.

# 4)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.

# 5)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>