ASSIGNMENT-4

SERVLET JSP INTERVIEW QUESTIONS

SERVLET

1. What is web application or client / server architecture?

Web Application Architecture:

Web application architecture typically follows a client/server model. Here's a simple breakdown with an illustration:

1. **Client Side (Front End):**
    - **Web Browser**: This is the client where users interact with the application.
    - **User Interface (UI)**: Includes elements like buttons, forms, and menus that users see and interact with.
    - **Client-Side Scripting**: Technologies like HTML, CSS, and JavaScript are used to create the UI and handle user interactions.
2. **Server Side (Back End):**
    - **Server**: This is where the application logic resides and data processing happens.

- **Business Logic**: Handles the core functionality of the application, such as calculations, validations, and database interactions.
- **Database**: Stores and manages data that the application needs (e.g., user information, content).
- **Server-Side Scripting**: Technologies like PHP, Python, Ruby, Java, etc., are used to implement server-side logic.

3. **Communication:**
   - **Request-Response Cycle**: The client sends requests (e.g., for a webpage, data submission) to the server.
   - **Processing**: The server processes the request, executes necessary actions (e.g., querying a database), and generates a response.
   - **Response**: The server sends back data (e.g., HTML for a webpage, JSON for an API) to the client.
4. **State Management:**
   - **Stateless Protocol**: HTTP is a stateless protocol, meaning each request from the client is independent.

- **Session Management**: Techniques like cookies or sessions are used to maintain state across multiple requests from the same client.

Key Points:

- **Scalability**: Servers can be scaled by adding more hardware or distributing the load across multiple servers.
- **Separation of Concerns**: Clients focus on presentation and user interaction, while servers handle data processing and business logic.
- **Flexibility**: Different client types (web browsers, mobile apps) can interact with the same server-side logic, providing a unified experience.

This architecture allows for efficient development, maintenance, and scaling of web applications, making it a fundamental approach in modern software engineering.

2.What is a Servlet?

Servlet is a Java class used in web development to handle requests and produce responses on the server-side. Here's a simpler breakdown:

- **Java Class**: It's a Java programming language class that runs on a web server.
- **Handles Requests**: Receives requests from web browsers or other clients.
- **Generates Responses**: Produces dynamic content (like web pages) based on the request.
- **Uses Java EE**: Part of Java Enterprise Edition (Java EE), now Jakarta EE, for building web applications.
- **Lifecycle**: Managed by the server, it initializes, processes requests, and cleans up after processing.

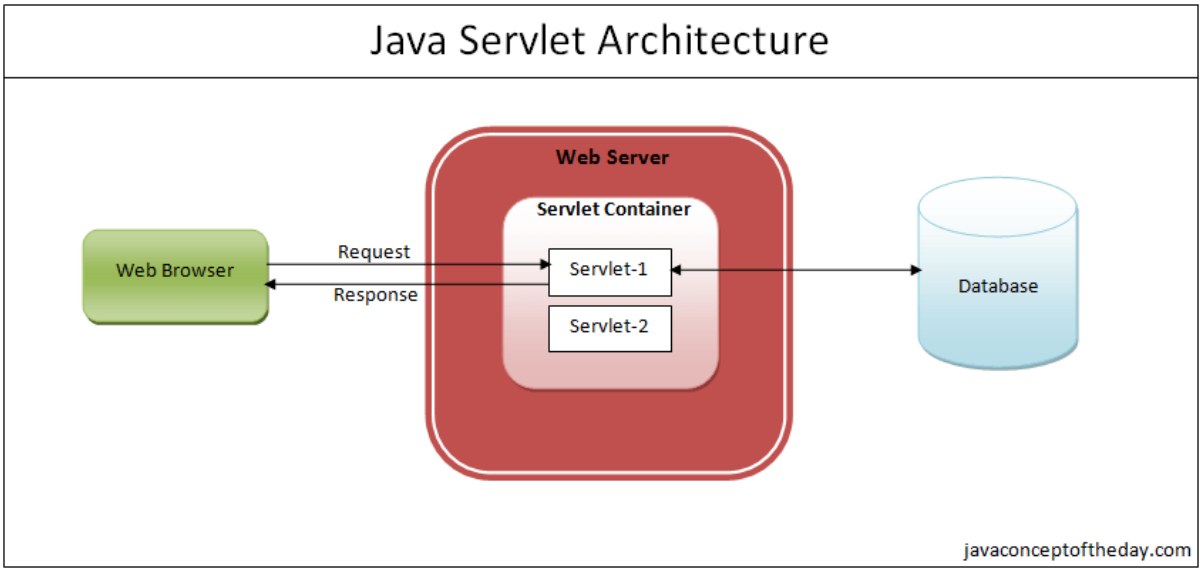
In essence, Servlets help manage the interaction between users and web applications on the server side, making web development in Java robust and flexible.

3. How PrintWriter works?

- **Purpose**: PrintWriter is used in Java to write text to different output destinations like files, network connections, or the console.
- **Usage**: You create a PrintWriter object by connecting it to an output destination, like a file (FileWriter) or the console (System.out).
- **Writing**: You can use methods like println (prints a line of text) or printf (prints formatted text) to write data to the destination.
- **Automatic Flushing**: PrintWriter automatically sends data to its destination whenever you use println or explicitly call flush.
- **Closing**: Always close the PrintWriter using the close method when you're done writing to release resources properly.

In essence, PrintWriter simplifies writing text to files or other output streams in Java applications.

4. What is servlet architecture?



Step 1 : Client i.e. web browser sends the request to the web server.

Step 2 : Web server receives the request and sends it to the servlet container. Servlet container is also called web container or servlet engine. It is responsible for handling the life of a servlet.

Step 3 : Servlet container understands the request's URL and calls the particular servlet. Actually, it creates a thread for execution of that servlet. If there are multiple requests for the same servlet, then for each request, one thread will be created.

Step 4 : Servlet processes the request object and prepares response object after interacting with the database or performing any other operations and sends the response object back to the web server.

Step 5 : Then web server sends the response back to the client.

5.What are life cycle methods of Servlet?

Java Servlet Life-Cycle

The Java Servlet Life cycle includes three stages right from its start to the end until the Garbage Collector clears it. These three stages are described below.

1. init()
2. service()
3. destroy()

1. init()

The init() is the germinating stage of any Java Servlet. When a URL specific to a particular servlet is triggered, the init() method is invoked.

Another scenario when the init() method gets invoked is when the servers are fired up. With every server starting up, the corresponding servlets also get started, and so does the init() method.

One important specialty of the init() method is the init() method only gets invoked once in the entire life cycle of the Servlet, and the init() method will not respond to any of the user's commands.

The init() method Syntax:

```
public void init() throws ServletException {


//init() method initializing
}
```

2. service()

The service() method is the heart of the life cycle of a Java Servlet. Right after the Servlet's initialization, it encounters the service requests from the client end.

The client may request various services like:

- GET
- PUT
- UPDATE
- DELETE

The service() method takes responsibility to check the type of request received from the client and respond accordingly by generating a new thread or a set of threads per the requirement and implementing the operation through the following methods.

- doGet() for GET
- doPut() for PUT
- doUpdate() for UPDATE

- doDelete() for DELETE

The service() method Syntax:

```
public void service(ServletRequest request, ServletResponse
response)

throws ServletException, IOException {

}
```

3. destroy()

Like the init() method, the destroy() method is also called only once in the Java Servlet's entire life cycle.

When the destroy() method is called, the Servlet performs the cleanup activities like,

- Halting the current or background threads
- Making a recovery list of any related data like cookies to Disk.

After that, the Servlet is badged, ready for the Garbage collector to have it cleared.

The destroy() method Syntax:

```
public void destroy() {


//destroy() method finalizing


}
```

6.What is difference between doGet and doPost()?

The doGet() and doPost() methods are two HTTP-specific methods defined in the javax.servlet.http.HttpServlet class, which is an abstract subclass of javax.servlet.GenericServlet. These methods are used to handle GET and POST requests, respectively, in Java servlets. Here are the main differences between doGet() and doPost():

1. **Purpose**:

- **doGet()**: This method is used to handle HTTP GET requests sent by a client (typically a web browser). GET requests are used to request data from a server. In servlets, doGet() is often used to fetch information from a server.
- **doPost()**: This method is used to handle HTTP POST requests sent by a client. POST requests are used to submit data to be processed by a server. In servlets, doPost() is commonly used when submitting forms or uploading files.

2. **Parameters**:
   - **doGet(HttpServletRequestrequest, HttpServletResponse response)**: This method receives an HttpServletRequest object that contains the request parameters sent by the client in the URL's query string. The HttpServletResponse object is used to send a response back to the client.
   - **doPost(HttpServletRequest request, HttpServletResponse response)**: This method also receives an HttpServletRequest object, but in this case, the parameters are sent in the body of the

HTTP request (e.g., form data). Again, the HttpServletResponse object is used to send the response back to the client.

3. **Usage**:
    - **doGet()**: Typically used when the request parameters can be appended to the URL (e.g., in the form of query parameters) or when the request does not change server state (idempotent operations). For example, fetching a list of items or retrieving details of a specific resource.
    - **doPost()**: Used when submitting sensitive information (such as login credentials) or when the request modifies server state (non-idempotent operations). For example, submitting a registration form or updating a database record.
4. **HTML Form Handling**:
    - When an HTML form is submitted using GET (method="GET"), the form data is appended to the URL as query parameters, and the browser sends a GET request to the server. The servlet can then handle this request using doGet().

- When an HTML form is submitted using POST (method="POST"), the form data is sent in the body of the HTTP request, and the browser sends a POST request to the server. The servlet can then handle this request using doPost().

In summary, doGet() and doPost() are methods used in servlets to handle different types of HTTP requests (GET and POST). The choice between them depends on the type of data being sent or retrieved and the nature of the operation being performed on the server.

7. When does destroy get called?

The destroy() method in a servlet is called when the servlet container decides to stop or remove the servlet from service. This happens during these situations:

1. **Web Application Shutdown**: When the web application containing the servlet is stopped or undeployed.
2. **Servlet Container Shutdown**: When the entire servlet container (like Tomcat) is stopped.

3. **Web Application Reload**: Some servers reload web applications without stopping the server. The servlet's destroy() method is called before reloading.

The purpose of destroy() is to allow the servlet to clean up resources it has used during its lifecycle. For example, closing database connections or stopping background tasks.

In essence, destroy() ensures that a servlet releases resources properly when it's no longer needed, based on server or application shutdown events.

8. What are ways to. Implements servlet?

1.Extending HttpServlet (for HTTP servlets):

- Create a Java class that extends javax.servlet.http.HttpServlet.
- Override doGet(HttpServletRequest req, HttpServletResponse res) for handling GET requests and/or doPost(HttpServletRequest req, HttpServletResponse res) for handling POST requests.

Example:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

public class MyServlet extends HttpServlet {
   protected void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException {
      // Servlet logic for handling GET requests
   }

   protected void doPost(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException {
      // Servlet logic for handling POST requests
   }
}
```

2.Implementing GenericServlet (for generic servlets):

- Create a Java class that extends javax.servlet.GenericServlet.
- Override the service(ServletRequest req, ServletResponse res) method to handle all types of requests (GET, POST, etc.).

Example:

```
import javax.servlet.*;
import java.io.IOException;

public class MyServlet extends GenericServlet {
    public void service(ServletRequest req,
ServletResponse res) throws ServletException,
IOException {
        // Servlet logic for handling requests
    }
}
```

These approaches provide the basic structure for implementing servlets in Java. The choice between

HttpServlet and GenericServlet depends on whether building an HTTP-based web application or need a more generic servlet that can handle various types of requests.

9. What are http methods?

HTTP methods, also known as HTTP verbs, are actions that indicate what a client (such as a web browser) wants to do with a resource on a server. These methods define the operations that can be performed on resources identified by URLs. The most commonly used HTTP methods are:

1. **GET**:
    - Requests data from a specified resource.
    - Should only retrieve data and should not have any other effect.
    - Example: Fetching a web page, fetching an image or document.
2. **POST**:
    - Submits data to be processed to a specified resource.
    - Typically used to send data to the server to update or create a resource.

- The data sent to the server is included in the body of the request.
- Example: Submitting a form, uploading a file.

3. **PUT**:
   - Updates a resource or creates a new resource if it doesn't exist at a specified URL.
   - The entire resource is replaced with the new representation provided in the request.
   - Example: Updating a user profile, uploading a new version of a document.
4. **DELETE**:
   - Deletes the specified resource at the given URL.
   - Example: Deleting a user account, deleting a file.
5. **HEAD**:
   - Requests headers that are returned if the specified resource would be requested with a GET method.
   - Useful for obtaining metadata about a resource without transferring the entire content.
   - Example: Checking if a resource has been modified since a certain date.
6. **OPTIONS**:

- Returns the HTTP methods that the server supports for a specified URL.
- Used to describe the communication options for the target resource.
- Example: Checking which methods are supported for a particular endpoint.

7. **PATCH**:
   - Applies partial modifications to a resource at a specified URL.
   - It is used to apply a set of changes described in the request entity to the resource identified by the URL.
   - Example: Partially updating a resource, such as updating specific fields of a user profile.

These HTTP methods provide a standardized way for clients (such as browsers, mobile apps, etc.) to interact with web servers and perform actions on resources. Each method has specific semantics and guidelines for its usage, which are defined by the HTTP specification (RFC 7231). The appropriate method to use depends on the action you want to perform and the semantics associated with that action.

10. What http method used by Servlet?

In Java Servlets, the primary methods used to handle HTTP requests are doGet() and doPost():

1. **doGet() Method**:
    - Used to handle HTTP GET requests.
    - Override this method in your servlet class to process requests that retrieve data from the server.
    - **Example:**

```
protected void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException {
    // Servlet logic for handling GET requests
}
```

2. **doPost() Method**:
    - Used to handle HTTP POST requests.
    - Override this method in your servlet class to process requests that submit data to be processed by the server.

      Example:

```
protected void doPost(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException {
    // Servlet logic for handling POST requests
}
```

These methods are part of the HttpServlet class in Java, and they allow servlets to handle different types of HTTP requests from clients (like web browsers or mobile apps). By implementing these methods and defining the appropriate logic inside them, servlets can respond to various actions requested by users, such as retrieving data, submitting forms, updating resources, and more.

11. What is difference between Generic Servlet and HttpServle

- **GenericServlet**: Handles requests from any protocol (like FTP, SMTP), but requires more manual handling of protocol specifics.
- **HttpServlet**: Specifically designed to handle HTTP requests (like GET, POST). It provides built-in methods (doGet, doPost) tailored for common web applications.

In essence, GenericServlet is versatile but requires more customization for different protocols, while HttpServlet is focused on HTTP and simplifies handling typical web interactions.

12. What is the use of RequestDispatcher Interface?

The RequestDispatcher interface in Java Servlets is used to:

- Forward a request from one servlet to another servlet, JSP page, or HTML file.
- Include the content of another servlet, JSP page, or HTML file in the response of the current servlet.

It helps in organizing and reusing code across servlets and web components by allowing them to work together to generate a cohesive response for clients (like web browsers).

13. What is the difference between ServletConfig and ServletContext?

- **ServletConfig**:
    - Configuration specific to an individual servlet instance.

    - Accessed using init() method of the servlet.
    - Used for initializing parameters unique to each servlet.
    - Example: Setting up database credentials for a particular servlet.
- **ServletContext**:
    - Configuration and resources shared across all servlets in a web application.
    - Accessed using getServletContext() method.
    - Used for storing application-wide parameters, attributes, and resources.
    - Example: Storing application-wide settings like the application name or shared database connection pool.

In essence, ServletConfig is for individual servlet configuration, while ServletContext is for global application configuration and resource sharing among all servlets.

14. What do you mean by InterServlet communication?

InterServlet communication simply means that servlets within the same web application can talk to each

other and share information. This communication allows them to work together to handle tasks more effectively. For example, one servlet might pass data to another servlet for processing, or they might collaborate to serve a client request by sharing resources like database connections or cached data. This interaction helps in building cohesive and efficient web applications.

15. What is significance of web.xml?

The web.xml file is important for Java web applications because it acts as a configuration file. It tells the web server how to handle different parts of the application, like which Java classes should handle specific web addresses (URLs) and how to set up security rules. Basically, it's like a map that tells the web server how everything in your web app should work together.

16. Explain Web Container?

A web container, also known as a servlet container or a servlet engine, is a runtime environment for web applications in the Java Enterprise Edition (Java EE)

platform. It provides the necessary infrastructure to execute Java Servlets, JavaServer Pages (JSP), and other related technologies that conform to the Java Servlet API specification.

Here are the key aspects and functionalities of a web container:

1. **Servlet Management**: A web container manages the lifecycle of servlets. It initializes servlets when they are first requested or when the container itself starts up, handles concurrent requests to servlets, and manages the destruction of servlet instances.
2. **Request Handling**: It receives HTTP requests from clients (typically web browsers or other web-based clients), parses these requests, and directs them to the appropriate servlet or JSP based on the URL mapping defined in the web application's deployment descriptor (web.xml).
3. **Thread Management**: Web containers use threads to handle multiple requests concurrently. They manage thread pools to efficiently allocate threads for incoming

requests, ensuring optimal performance and resource utilization.

4. **Lifecycle Management**: Web containers support the lifecycle of web applications, including startup, shutdown, and maintenance tasks. They handle initialization of servlets and filters, as well as cleaning up resources when the application is undeployed.
5. **Session Management**: They manage HTTP sessions, which allow web applications to maintain stateful interactions with clients across multiple requests. Session tracking mechanisms, such as cookies or URL rewriting, are often used by web containers to maintain session information.
6. **Security**: Web containers provide mechanisms for securing web applications. This includes authentication (verifying the identity of users), authorization (controlling access to resources based on user roles or permissions), and handling secure communications over HTTPS.
7. **Integration with Web Servers**: Web containers can be integrated with web servers (like Apache HTTP Server or nginx) to handle dynamic content generation and servlet

execution. This integration is typically achieved through connectors or plugins that forward requests from the web server to the servlet container.

Popular examples of Java web containers include Apache Tomcat, Jetty, and Red Hat JBoss Web (formerly known as JBoss AS). These containers comply with the Java Servlet API specification, ensuring compatibility with servlets and JSPs developed according to Java EE standards.

In summary, a web container is a crucial component in the Java EE ecosystem, providing the runtime environment and infrastructure necessary to execute dynamic web applications built using Java servlets and JSP technology.

17. What do you mean by the Servlet Chaining?

Servlet chaining is the process of connecting multiple servlets together so that each servlet in the chain can process a part of a client's request in a sequence. Each servlet does its job and then passes the request to the next servlet in line. This approach helps in breaking down complex tasks into smaller, manageable parts and promotes reusability of code components in web applications.

18. Why do we use sendredirect() method?

sendRedirect() method redirects the response to another resource, inside or outside the server. It makes the client/browser to create a new request to get to the resource. It sends a temporary redirect response to the client using the specified redirect location URL.

Syntax:

void sendRedirect(java.lang.String location) throws java.io.IOException

- sendRedirect() accepts the respective URL to which the request is to be redirected.
- Can redirect the request to another resource like Servlet, HTML page, or JSP page that are inside or outside the server.
- It works on the HTTP response object and always sends a new request for the object.
- A new URL that is being redirected can be seen in the browser.

19. What Servlet filters?

Servlet filters in Java web development act as intermediary components that intercept and modify requests and responses between clients and servlets. They're like gatekeepers for web applications, allowing you to:

- **Pre-process** incoming requests before they reach servlets.
- **Post-process** outgoing responses before they reach clients.

Filters are useful for tasks like logging, authentication, data compression, and modifying headers. They're configured in the web.xml file or with annotations (@WebFilter) and executed in a defined order based on URL patterns.

20.when to use Servelet Filter?

Servlet filters are used in Java web applications to perform various tasks related to request and response processing. Here are some common scenarios when you might want to use servlet filters:

1. **Authentication and Authorization**: Filters can check if a user is logged in (authentication) and has the necessary

permissions (authorization) before allowing access to servlets or resources. This helps enforce security policies across your application.

2. **Logging and Auditing**: Filters can log incoming requests, outgoing responses, and other relevant information. This is useful for debugging, performance monitoring, and auditing purposes.
3. **Data Compression**: Filters can compress the response data before sending it to the client, which reduces bandwidth usage and improves application performance, especially for large payloads.
4. **Encryption**: Filters can encrypt sensitive data in the request or response to ensure secure communication between clients and servers.
5. **URL Rewriting**: Filters can modify URLs or URL parameters for various purposes such as session tracking, URL normalization, or redirecting requests to different locations based on certain conditions.
6. **Content Transformation**: Filters can modify the content of requests or responses. For example, transforming XML

data into JSON format or vice versa based on client preferences.

7. **Caching**: Filters can implement caching strategies to store and reuse responses for identical requests, which improves application performance by reducing server load and response time.
8. **Request Filtering**: Filters can intercept requests based on specific criteria (e.g., IP address, headers, parameters) and take actions such as blocking requests, redirecting to other URLs, or modifying requests before they reach servlets.
9. **Cross-cutting Concerns**: Filters help manage cross-cutting concerns such as logging, security, and performance monitoring in a modular and reusable manner without cluttering servlet code.
10. **Performance Optimization**: Filters can preprocess requests (e.g., validating input data) and post-process responses (e.g., adding caching headers) to optimize application performance and scalability.

In summary, servlet filters are versatile tools that allow developers to intercept and manipulate requests and

responses at various stages of processing. They provide a clean and reusable way to implement cross-cutting concerns and enhance the functionality, security, and performance of Java web applications.

JSP (JAVA SERVER PAGES)

21. How do we translate JSP?

To translate JSP (JavaServer Pages) for multiple languages, follow these simplified steps:

1. **Identify Text**: Find all text in your JSP that needs translation, like labels and messages.
2. **Create Properties Files**: Make .properties files for each language (e.g., messages_en.properties for English, messages_fr.properties for French).
3. **Use Keys**: Replace text in your JSP with keys that match entries in the .properties files.
4. **Access with JSTL**: Use <fmt:message> tags from JSTL to fetch translations based on the user's locale.

5. **Test**: Ensure translations work correctly by testing with different languages.

This process ensures your JSP pages can display content in various languages based on user preferences or settings.

22. What is JSP ?

In Java, **JSP** stands for **Jakarta Server Pages(** (**JSP**; formerly **JavaServer Pages**)**)**. It is a server-side technology which is used for creating web applications. It is used to create dynamic web content. JSP consists of both HTML tags and JSP tags. In this, JSP tags are used to insert JAVA code into HTML pages

23. Why do you use JSP?

JSP (JavaServer Pages) is used because:

1. **Integration**: It seamlessly integrates Java code with HTML for dynamic web page generation.
2. **Ease**: It's easy to learn and use, especially for developers familiar with HTML and Java.

3. **Separation**: It allows clear separation of presentation (HTML) and business logic (Java code).
4. **Scalability**: It scales well for building complex, data-driven web applications.
5. **Community**: It has a large community and ecosystem, making support and resources widely available.

24. What are implicit object in JSP?

In JSP (JavaServer Pages), implicit objects are predefined objects that are automatically available for use in every JSP page without needing to be explicitly declared or instantiated. These objects provide convenient access to various elements of the request, session, application, and other JSP-related information. Here are the commonly used implicit objects in JSP:

1. **request**: Represents the HttpServletRequest object, which encapsulates the client's request information. It provides methods to retrieve parameters, headers, cookies, and other request details.
2. **response**: Represents the HttpServletResponse object, which is used to manipulate the response sent to the

client. It provides methods to set headers, cookies, and send content back to the client.

3. **out**: Represents the JspWriter object, which is used to send output to the client's browser. It's typically used to print content within the HTML response.
4. **session**: Represents the HttpSession object, which provides session management functionality. It allows storing and retrieving session attributes across multiple requests from the same client.
5. **application**: Represents the ServletContext object, which provides access to the application-wide information and resources. It allows storing and retrieving attributes that are accessible to all servlets and JSP pages within the web application.
6. **config**: Represents the ServletConfig object associated with the current JSP. It provides configuration information for the JSP servlet, such as initialization parameters defined in the web.xml file.
7. **pageContext**: Represents the PageContext object, which encapsulates all aspects of the JSP page execution context. It provides methods to access various scopes

(request, session, application), handle page directives, and manage custom tags.

8. **exception**: Represents the Throwable object that holds information about any exception thrown during the execution of the JSP page, if applicable.

These implicit objects simplify the development of dynamic web applications by providing easy access to commonly used information and functionality within the JSP environment. They are automatically instantiated by the JSP container and are available for use throughout the lifecycle of the JSP page.

25. What are scriptlet in JSP?

Scriptlets in JSP are blocks of Java code enclosed within <% %> tags. They allow developers to embed dynamic server-side logic directly into HTML pages to generate dynamic content and handle requests.

26. What are directive?

Directives in JSP are special commands that provide configuration information to the JSP container. There are three types:

1. **Page Directive**: Defines attributes like the scripting language, content type, and session handling for the JSP page.
2. **Include Directive**: Includes contents of another file into the current JSP page during translation.
3. **Taglib Directive**: Declares and associates custom tag libraries with prefixes for use in the JSP page.

These directives help configure how JSP pages are processed and interact with resources and libraries.

27. How to execute Java code in JSP?

To execute Java code in JSP, you embed it directly into the page using scriptlet tags (<% %>), expression tags (<%=

%>), or declaration tags (<%! %>). These tags allow you to mix Java logic with HTML to generate dynamic content for web pages.

28. Why jsp is in when Servlet can do everything what jsp can do?

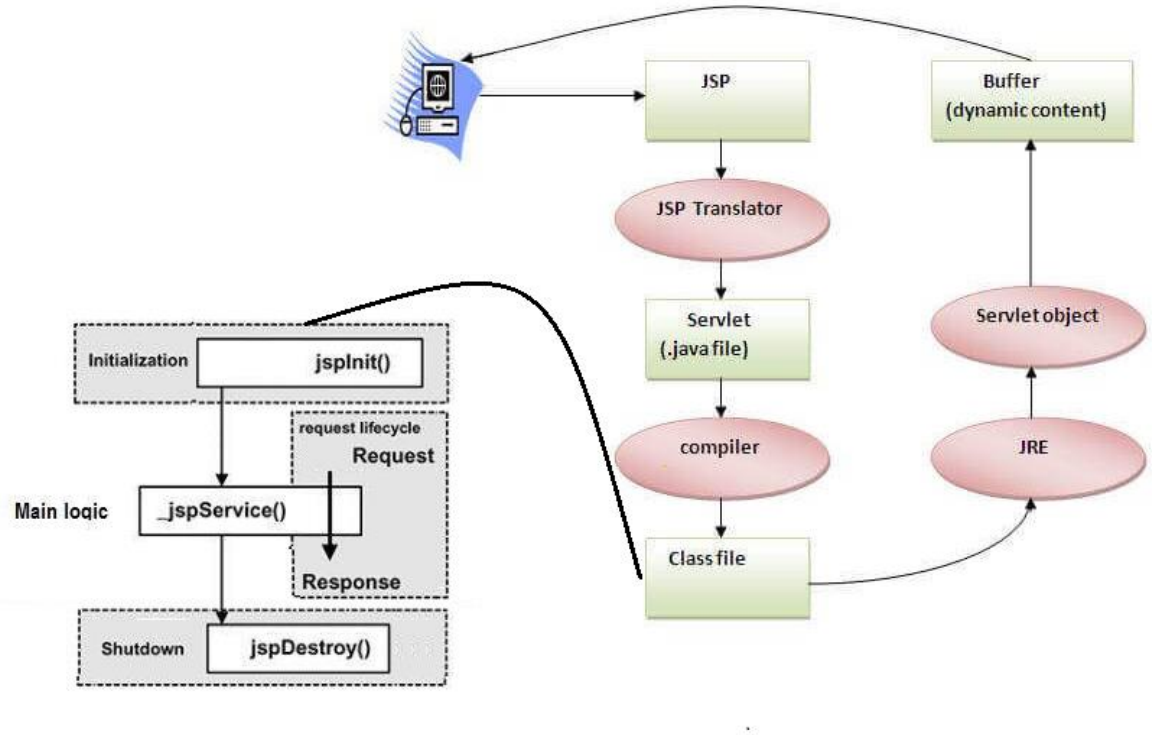
Servlets and JSP serve different purposes in Java web development:

- **Servlets**: Handle requests, execute Java code, manage application logic, and generate dynamic content by writing directly to the response stream.
- **JSP**: Simplifies the integration of dynamic content with HTML by allowing Java code to be embedded within HTML markup. It's used for creating web pages where dynamic data and logic are mixed with static content.

While Servlets can manage dynamic content, JSP makes it easier to combine Java logic with HTML, which is useful for separating concerns between presentation and business logic in web applications.

29. What is lifecycle phases of JSP

A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.



Lifecycle of JSP

Following steps are involved in the JSP life cycle:

1. Translation of JSP page to Servlet

2. Compilation of JSP page(Compilation of JSP into test.java)
3. Classloading (test.java to test.class)
4. Instantiation(Object of the generated Servlet is created)
5. Initialization(jspInit() method is invoked by the container)
6. Request processing(_jspService()is invoked by the container)
7. JSP Cleanup (jspDestroy() method is invoked by the container)

We can override jspInit(), jspDestroy() but we can't override _jspService() method.

30. What are the method used here

In the lifecycle of a JSP page, these methods are involved:

1. **init()**: Initializes the JSP page, typically used for setup tasks.
2. **service()**: Handles client requests and generates responses for the JSP page.
3. **_jspService()**: Automatically generated to execute the main logic of the JSP page.
4. **destroy()**: Cleans up resources when the JSP page is no longer needed.