ADVANCE JAVA

UNIT 2

By Shivani Deopa

Servlets:

Introduction

- Today we all are aware of the need to create dynamic web pages i.e. the ones that can change the site contents according to the time or can generate the content according to the request received by the client. If you like coding in Java, then you will be happy to know that using Java there also exists a way to generate dynamic web pages and that way is Java Servlet.
- Servlets are the Java programs that runs on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, then send response back to the web server.

Properties of Servlets :

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from web server.

Execution of Servlets involves six basic steps:

- The clients send the request to the web server.
- The web server receives the request.
- The web server passes the request to the corresponding servlet.
- The servlet processes the request and generates the response in the form of output.
- The servlet sends the response back to the web server.
- The web server sends the response back to the client and the client browser displays it on the screen.

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

public class servlet1 extends GenericServlet
{
   public void doGet(HttpServletRequest request, HttpServletResponse
response)
      throws ServletException, IOException {
            .
            .
            .
   }
}
```

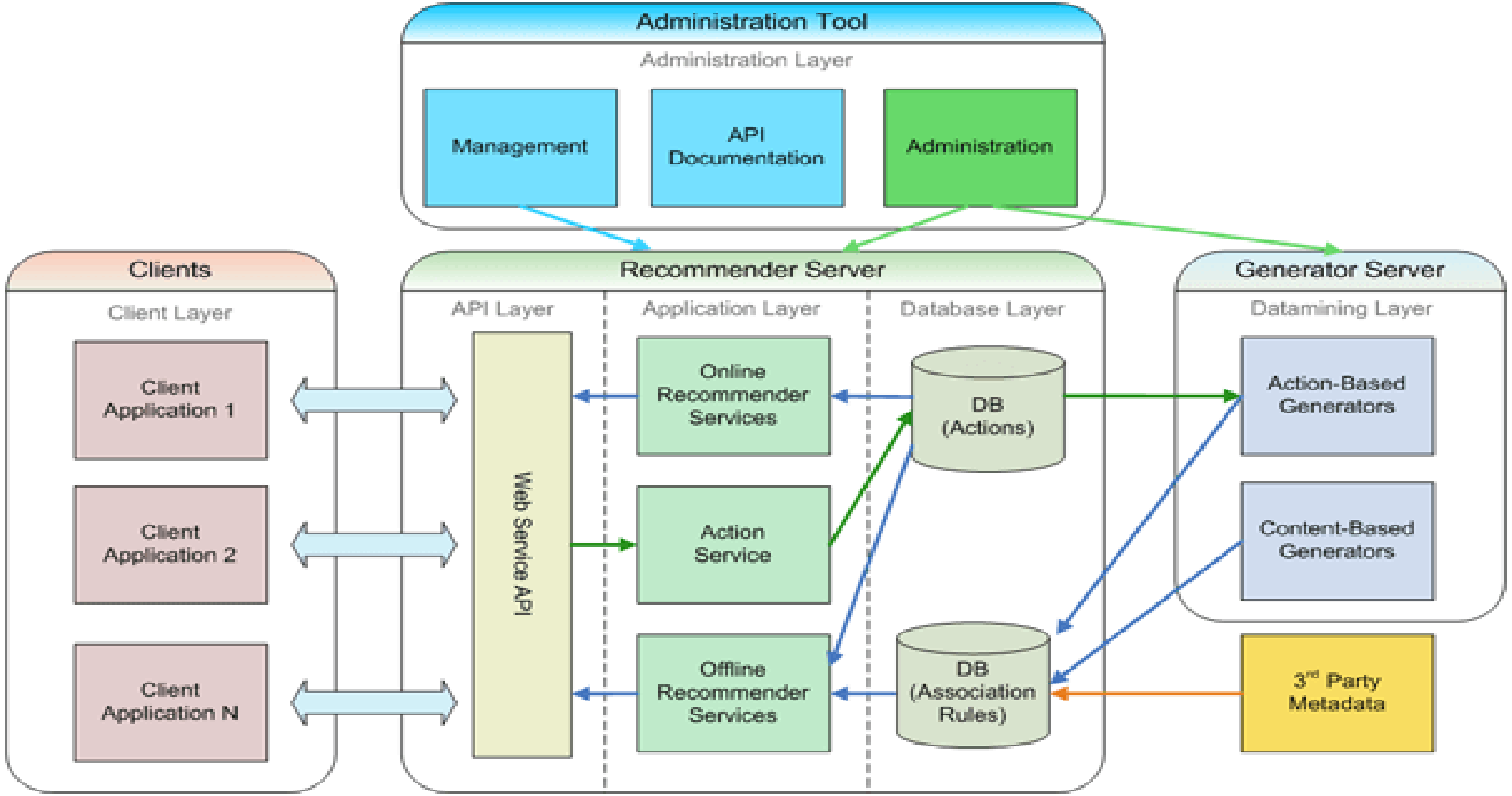
Advantages of a Java Servlet

- Servlet is **faster** than CGI(Common Gateway Interface) as it doesn't involve the creation of a new process for every new request received.
- Servlets as written in Java are **platform independent**.
- Removes the overhead of creating a **new process** for each request as Servlet doesn't run in a separate process. There is only a single instance which handles all requests concurrently. This also saves the memory and allows a Servlet to easily manage client state.
- It is a server-side component, so Servlet inherits the **security** provided by the Web server.
- The **API** designed for Java Servlet automatically acquires the advantages of Java platform such as platform independent and portability. In addition, it obviously can use the wide range of APIs created on Java platform such as **JDBC** to access the database.

Web Application Architecture

- Web application architecture defines the interactions between applications, middleware systems and databases to ensure multiple applications can work together. When a user types in a URL and taps “Go,” the browser will find the Internet-facing computer the website lives on and requests that particular page.
- The server then responds by sending files over to the browser. After that action, the browser executes those files to show the requested page to the user. Now, the user gets to interact with the website. Of course, all of these actions are executed within a matter of seconds. Otherwise, users wouldn’t bother with websites.

Working of Web Application Architecture

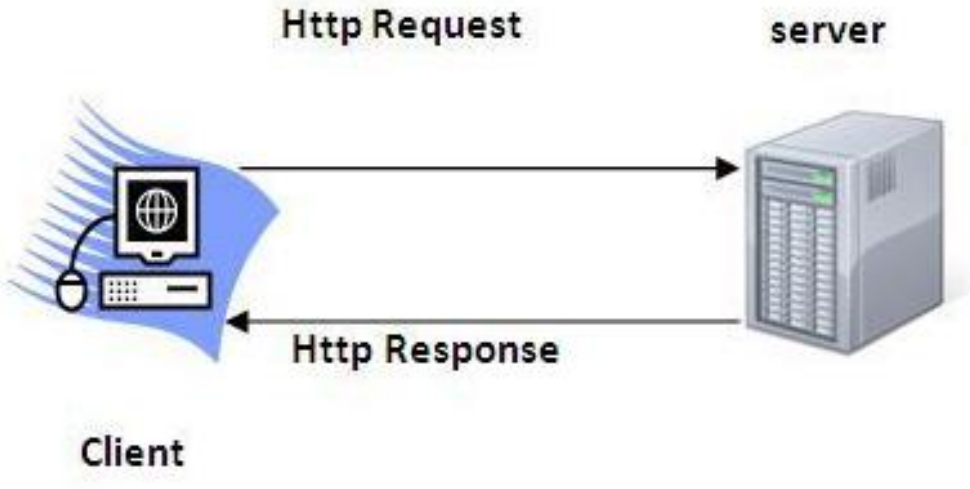


- With web applications, you have the server vs. the client side. In essence, there are two programs running concurrently:

1. The code which lives in the browser and responds to user input
2. The code which lives on the server and responds to HTTP requests

- When writing an app, it is up to the web developer to decide what the code on the server should do in relation to what the code on the browser should do. With server-side code, languages include:
- PHP, C#, Java, Python, Javascript
- In fact, any code that can respond to HTTP requests has the capability to run on a server. Here are a few other attributes of server-side code:
- Is never seen by the user (except within a rare malfunction)
- Stores data such as user profiles, tweets, pages, etc…
- Creates the page the user requested
- With client-side code, languages used include:
- CSS, Javascript, HTML
- These are then parsed by the user's browser. Moreover, client-side code can be seen and edited by the user. Plus, it has to communicate only through HTTP requests and cannot read files off of a server directly. Furthermore, it reacts to user input

Http Protocol & Http Methods



- The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.
- HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.

- The Basic Characteristics of HTTP (Hyper Text Transfer Protocol):

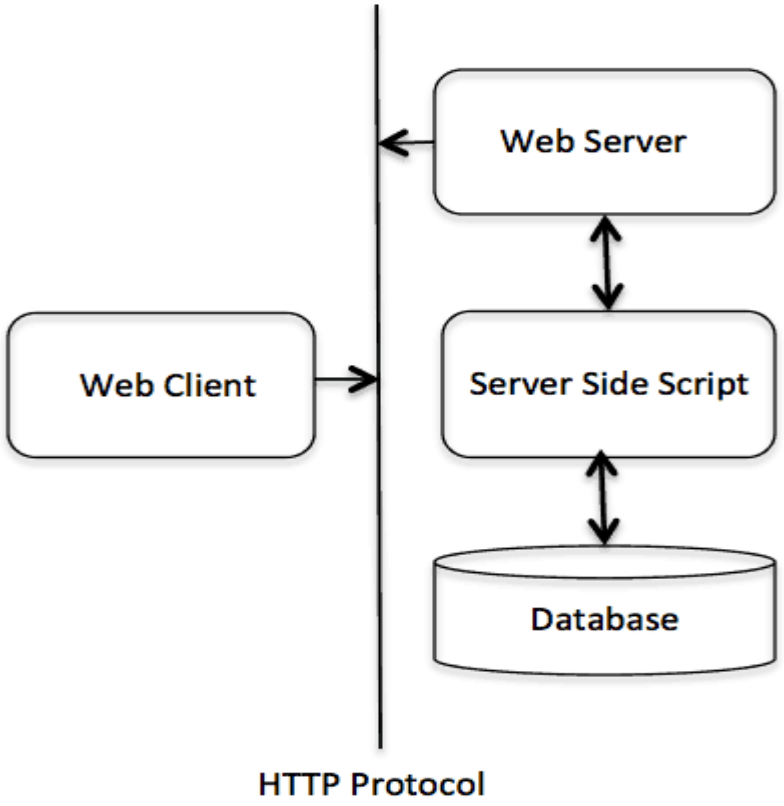
1. It is the protocol that allows web servers and browsers to exchange data over the web.
2. It is a request response protocol.
3. It uses the reliable TCP connections by default on TCP port 80.
4. It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

- The Basic Features of HTTP (Hyper Text Transfer Protocol):

1. There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:
2. HTTP is media independent: It specifies that any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
3. HTTP is connectionless: It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.
4. HTTP is stateless: The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

The Basic Architecture of HTTP (Hyper Text Transfer Protocol):

- HTTP is request/response protocol which is based on client/server based architecture. In this protocol, web browser, search engines, etc. behave as HTTP clients and the Web server like Servlet behaves as a server



HTTP REQUEST

- The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.
- The HTTP client sends the request to the server in the form of request message which includes following information:

1. The Request-line
2. The analysis of source IP address, proxy and port
3. The analysis of destination IP address, protocol, port and host
4. The Requested URI (Uniform Resource Identifier)
5. The Request method and Content
6. The User-Agent header
7. The Connection control header
8. The Cache control header

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

Web servers

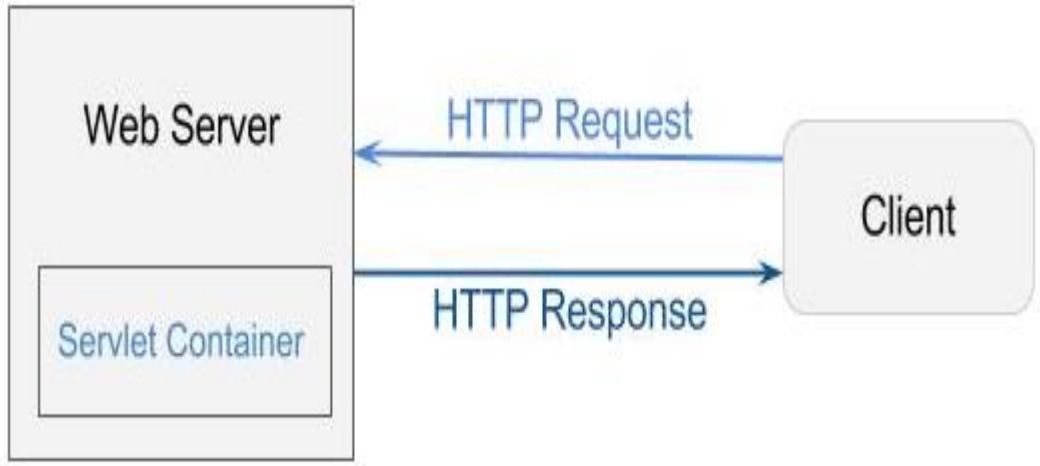
- A Web server is software or hardware that uses HTTP (Hypertext Transfer Protocol) and other protocols to respond to client requests made over the World Wide Web (WWW). Web server software controls how a user accesses hosted files. It is accessed through the domain names of websites and ensures the delivery of the site's content to the requesting user. As hardware, a Web server is a computer that holds web server software and other files related to a website, such as HTML documents, images and JavaScript files. Web server hardware is connected to the internet and allows data to be exchanged with other connected devices.

Examples:

- Apache HTTP Server, Internet Information Services, Sun Java System Web Server, Jigsaw Server

Web Container(Servlet Container)

- It provides the runtime environment for JavaEE (j2ee) applications. The client/user can request only a static WebPages from the server. If the user wants to read the web pages as per input then the servlet container is used in java.
- The servlet container is the part of web server which can be run in a separate process. We can classify the servlet container states in three types:



- **Standalone:** It is typical Java-based servers in which the servlet container and the web servers are the integral part of a single program. For example:- Tomcat running by itself
- **In-process:** It is separated from the web server, because a different program runs within the address space of the main server as a plug-in. For example:- Tomcat running inside the JBoss.
- **Out-of-process:** The web server and servlet container are different programs which are run in a different process. For performing the communications between them, web server uses the plug-in provided by the servlet container.
- **The Servlet Container performs many operations that are given below:**
- Life Cycle Management
- Multithreaded support
- Object Pooling
- Security etc.

Servlet Interface

- Servlet interface provides common behaviour to all the servlets.
- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

1. **void destroy():** This method is called by Servlet container at the end of servlet life cycle. Unlike service() method that gets called multiple times during life cycle, this method is called only once by Servlet container during the complete life cycle. Once destroy() method is called the servlet container does not call the service() method for that servlet.
2. **void init(ServletConfig config):** When Servlet container starts up (that happens when the web server starts up) it loads all the servlets and instantiates them. After this init() method gets called for each instantiated servlet, this method initializes the servlet.
3. **void service(ServletRequest req, ServletResponse res):** This is the only method that is called multiple times during servlet life cycle. This methods serves the client request, it is called every time the server receives a request.
4. **ServletConfig getServletConfig():** Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.
5. **java.lang.String getServletInfo():** Returns information about the servlet, such as author, version, and copyright.

GenericServlet

- **GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.

- There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.

6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

HttpServlet

- The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

- There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.

6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

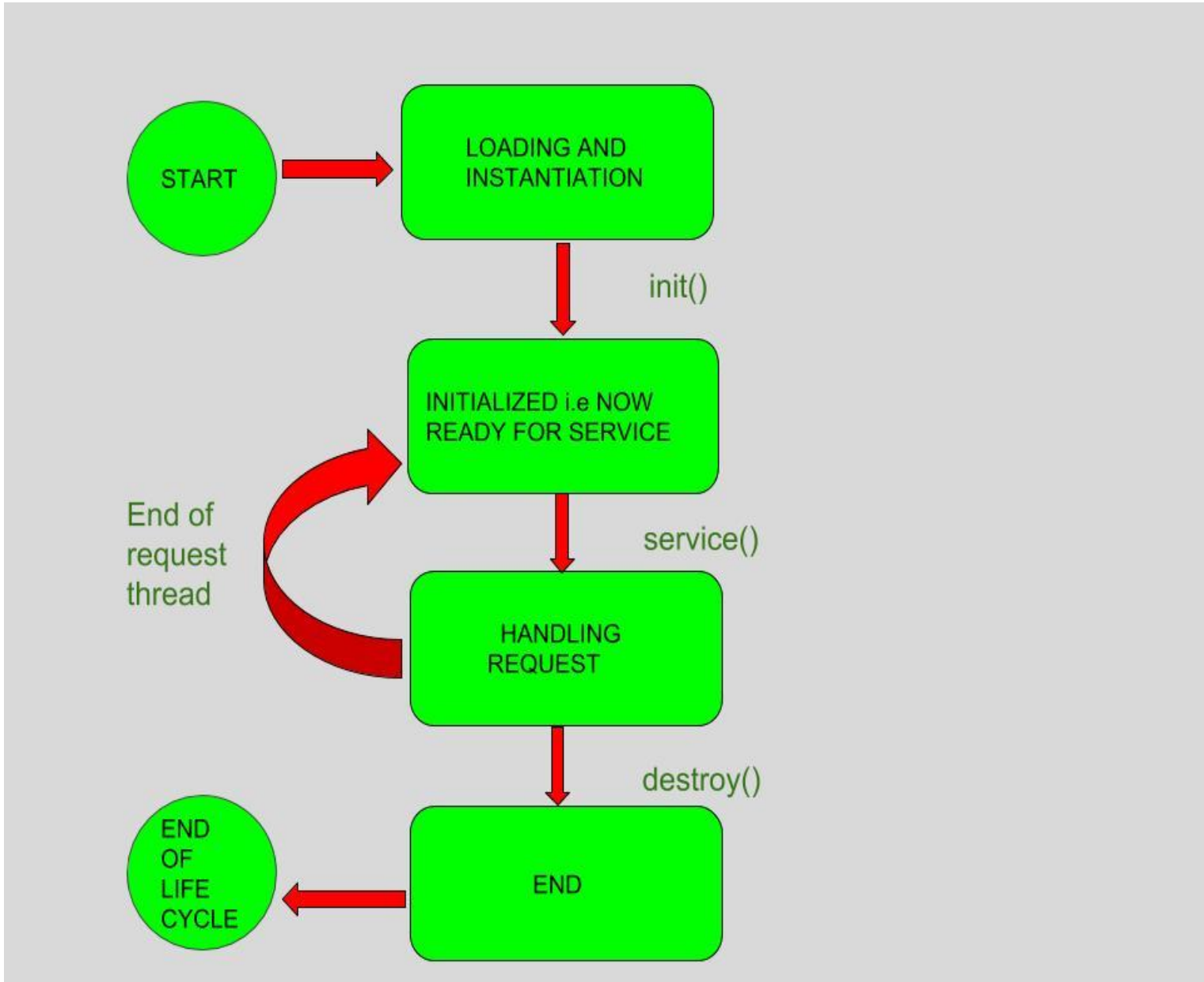
GenericServlet vs HttpServlet

- GenericServlet defines a generic, protocol-independent servlet.
- GenericServlet gives a blueprint and makes writing servlet easier.
- GenericServlet provides simple versions of the lifecycle methods init and destroy and of the methods in the ServletConfig interface.
- GenericServlet implements the log method, declared in the ServletContext interface.
- To write a generic servlet, it is sufficient to override the abstract service method.
- HttpServlet defines a HTTP protocol specific servlet.
- HttpServlet gives a blueprint for Http servlet and makes writing them easier.
- HttpServlet extends the GenericServlet and hence inherits the properties GenericServlet.

Servlet Life Cycle

- A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

1. The servlet is initialized by calling the **init()** method.
2. The servlet calls **service()** method to process a client's request.
3. The servlet is terminated by calling the **destroy()** method.
4. Finally, servlet is garbage collected by the garbage collector of the JVM.


START
LOADING AND INSTANTIATION
init()
INITIALIZED i.e NOW READY FOR SERVICE
End of request thread
service()
HANDLING REQUEST
destroy()
END
END OF LIFE CYCLE

- **init() method:** The Servlet.init() method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.

```
//init() method
public class MyServlet implements Servlet{
        public void init(ServletConfig config) throws ServletException {
                //initialization code
        }
        //rest of code
}
```

- **destroy() method:** The destroy() method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

```
//destroy() method
public void destroy()
```

As soon as the destroy() method is activated, the Servlet container releases the Servlet instance.

- **service() method:** The service() method of the Servlet is invoked to inform the Servlet about the client requests.
- This method uses ServletRequest object to collect the data requested by the client.
- This method uses ServletResponse object to generate the output content.

```
// service() method
public class MyServlet implements Servlet{
   public void service(ServletRequest res, ServletResponse res)
  throws ServletException, IOException {
            // request handling code
  }
            // rest of code
}
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class AdvanceJavaConcepts
extends HttpServlet
{
   private String output;
   public void init() throws
ServletException
   {
      output = "Advance Java Concepts";
   }
```

```
public void doGet(HttpServletRequest
req,  HttpServletResponse resp)
      throws ServletException,
IOException
      {
         resp.setContentType("text/html");
         PrintWriter out = resp.getWriter();
         out.println(output);
      }
      public void destroy()
      {
         System.out.println("Over");
      }
}
```

ServletConfig

- ServletConfig interface is used to assign properties to a Servlet at the time of Servlet object creation itself (known as Servlet Initialization Parameters) by the container; exactly the way you use constructor in case of applications.
- **web.xml** is known as deployment descriptor, where in, the deployment particulars (that also include **initialization parameters**) of a Servlet are written by the Programmer. The particulars are declared in <**init-param**> tag in **web.xml** file. **To read this, <init-param> tag data, ServletConfig interface is used.**
- Infact, an object of **ServletConfig** is created by the Container itself at the time it creates a Servlet object. **For every object of Servlet, the container creates, also an object of ServletConfig.** This ServletConfig object is used to communicate with the Servlet under execution. Eventhough, the contianer creates the ServletConfig object implicitly for its purpose, the Programmer can make use of the object in his code to read **initialization parameters** into the Servlet from **web.xml** file.
- If the parameter values change, the **web.xml** file need not be compiled. At the time reading, whatever values exists in the **web.xml** file, with those values or parameters only the Servlet is initialized.

- For each Servlet under execution, a ServletConfig object is created by the Servlet container and is used by the Programmer to read the Servlet specific data declared (written) in **web.xml** in the form of tags.
- ServletConfig is an interface from **javax.servlet** package.
- The **getServletConfig()** method of **GenericServlet** returns an object of ServletConfig.
- The **ServletConfig** object created by the Web container for a specific Servlet cannot read other servlet's **init-param** data.
- ServletConfig object is specific for a Servlet. Other way to say, if 100 servlet objects are being executed in the container, 100 ServletConfig objects are also created implicitly used by the container to communicate with the Servlet.
- When the container destroys the Servlet object, along with it its corresponding **ServletConfig** object also destroyed.
- Programmer can make use ServletConfig object to read tags of the Servlet.
- <init-param> tag data is known as **initialization parameters** and are used to initialize the Servlet with some special properties.
- The method of ServletConfig object used to read is "String getInitParameter(String)".
- The data, to be read by ServletConfig, is written within <servlet> tag.

SR.NO.	METHOD	SUMAMRY
1	String getInitParameter(String name)	Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
2	Enumeration getInitParameterNames()	Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
3	ServletContext getServletContext()	Returns a reference to the ServletContext in which the caller is executing.
4	String getServletName()	Returns the name of this servlet instance.

ServletContext

- The Servlet Container creates ServletConfig object for each Servlet during initialization. The main difference between ServletConfig and ServletContext is that unlike ServletConfig, the ServletContext is being created once per web application, i.e. ServletContext object is common to all the servlets in web application.
- This is how we can create ServletContext object. In this code we are creating object in init() method, however you can create the object anywhere you like.
- Once we have the ServletContext object, we can set the attributes of the ServletContext object by using the setAttribute() method. Since the ServletContext object is available to all the servlets of the Web application, other servlets can retrieve the attribute from the ServletContext object by using the getAttribute() method.

Context Initialization Parameter

- Context Initialization parameters are the parameter name and value pairs that you can specify in the deployment descriptor file (the web.xml file). Here you can specify the parameters that will be accessible to all the servlets in the web application.
- When we deploy the Web application, the Servlet container reads the initialization parameter from the web.xml file and initializes the ServletContext object with it. We can use the getInitParameter() and getInitParameterNames() methods of the ServletContext interface to get the parameter value and enumeration of parameter names respectively.
- For example, here I have specified the parameter email_id with the value, since this is common to all the servlets, you can get the parameter name and value in any servlet.

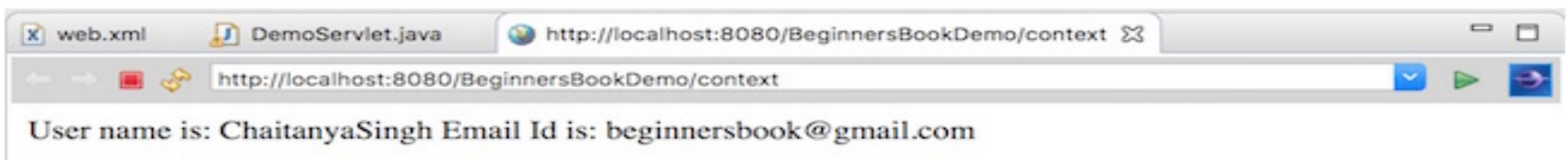
DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet extends HttpServlet{
   public void doGet(HttpServletRequest request,HttpServletResponse response)
   throws ServletException,IOException
   {
        response.setContentType("text/html");
        PrintWriter pwriter=response.getWriter();

        //ServletContext object creation
        ServletContext scontext=getServletContext();

        //fetching values of initialization parameters and printing it
        String userName=scontext.getInitParameter("uname");
        pwriter.println("User name is="+userName);
        String userEmail=scontext.getInitParameter("email");
        pwriter.println("Email Id is="+userEmail);
        pwriter.close();
   }
}
```

web.xml

```
<web-app>
<servlet>
   <servlet-name>BeginnersBook</servlet-name>
   <servlet-class>DemoServlet</servlet-class>
</servlet>
<context-param>
   <param-name>uname</param-name>
   <param-value>ChaitanyaSingh</param-value>
</context-param>
<context-param>
   <param-name>email</param-name>
   <param-value>beginnersbook@gmail.com</param-value>
</context-param>
<servlet-mapping>
   <servlet-name>BeginnersBook</servlet-name>
   <url-pattern>/context</url-pattern>
   </servlet-mapping>
</web-app>
```

Output:



- **public String getInitParameter(String param)**: It returns the value of given parameter or null if the parameter doesn’t exist.
- **public Enumeration getInitParameterNames()**: Returns an enumeration of context parameters names.
- **public void setAttribute(String name,Object object)**: Sets the attribute value for the given attribute name.
- **public Object getAttribute(String name)**:Returns the attribute value for the given name or null if the attribute doesn’t exist.
- **public String getServerInfo()**: eturns the name and version of the servlet container on which the servlet is running.
- **public String getContextPath()**: Returns the context path of the web application.

Servlet Communication

- Communication between Java servlets is called as Servlet communication. It is nothing but, sending users request and the response object passed to a servlet to another servlet. You may ask, what is the use of passing these objects.
- Well, i have an answer. As seen in the servlet examples written till date, one common method we are using, **getParameter(),** used to get the user input value in a specified field.
- So, when a request object is passed from one servlet to another servlet, then you can use this method to get the input that the user has given in a HTML/JSP form.

- Following example, gives better understand of what is said now.

Folder Structure

webapps

|_ servletcom

|_ index.html

|_ WEB-INF

|_ web.xml

|_ classes

|_ FirstServlet.java

|_ FirstServlet.class

|_ FinalServlet.java

|_ FinalServlet.class

Session Tracking Mechanisms

- Session simply means a particular interval of time.
- Session Tracking is a way to maintain state (data) of a user. It is also known as session management in servlet.
- Http protocol is a stateless, so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So, we need to maintain the state of a user to recognize to particular user.
- Session Tracking Techniques
- There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HTTP Session

Cookies

- A cookie is a small piece of information that is persisted between the multiple client requests.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
- **How Cookie works**
- By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

- There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

- **Non-persistent cookie**

It is valid for single session only. It is removed each time when user closes the browser.

- **Persistent cookie**

It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

- **Advantage of Cookies**

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

- **Disadvantage of Cookies**

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

- javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.
- Other methods required for using Cookies
- For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces.
- They are:

1. **public void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

- Let's see the simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

- Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

- Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

- Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

- There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Hidden Form Field

- In case of Hidden Form Field, a hidden (invisible) text field is used for maintaining the state of a user.
- In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- Let's see the code to store value in hidden field.

  <input type="hidden" name="uname" value="Vimal Jaiswal">

- Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Real application of hidden form field

- It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

- Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

- Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

URL rewriting

- In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

  url?name1=value1&name2=value2&??

- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.
- From a Servlet, we can use getParameter() method to obtain a parameter value.

- Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

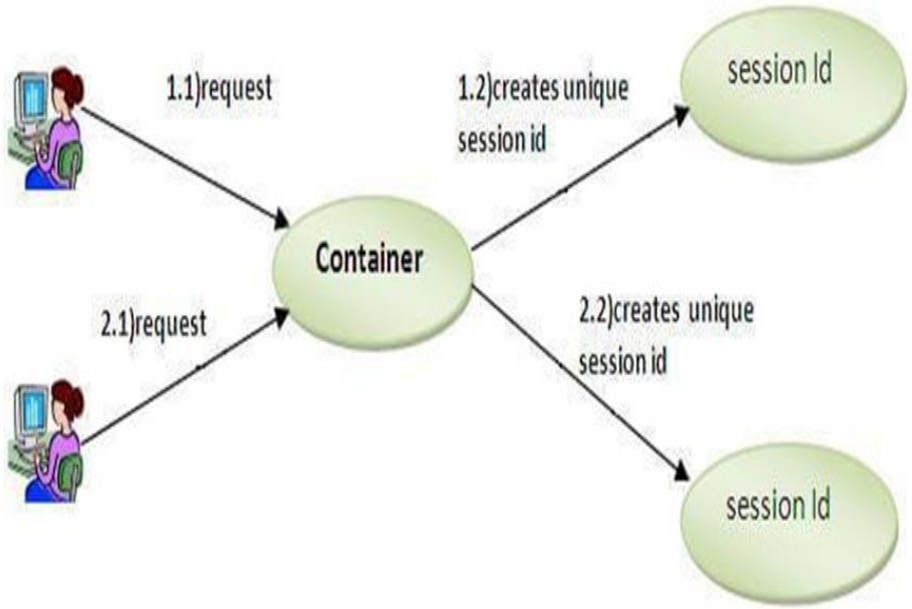
- Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

HTTP Session

- Here, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

- The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. public HttpSession getSession():Returns the current session associated with this request, or if the request does not have a session, creates one.
2. public HttpSession getSession(boolean create):Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

- Commonly used methods of HttpSession interface

1. public String getId():Returns a string containing the unique identifier value.
2. public long getCreationTime():Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. public long getLastAccessedTime():Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. public void invalidate():Invalidates this session then unbinds any objects bound to it.

JSP:

Introduction

- JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.
- Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.
- JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

- **Advantage**

1. It does not require advanced knowledge of JAVA
2. It is capable of handling exceptions
3. Easy to use and learn
4. It has tags which are easy to use and understand
5. Implicit objects are there which reduces the length of code
6. It is suitable for both JAVA and non JAVA programmer

- **Disadvantage**

1. Difficult to debug for errors.
2. First time access leads to wastage of time
3. It’s output is HTML which lacks features.

JSP Life Cycle

- A Java Server Page life cycle is defined as the process 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.
- Following steps are involved in 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)

- **Translation of JSP page to Servlet :**

  This is the first step of JSP life cycle. This translation phase deals with Syntactic correctness of JSP. Here test.jsp file is transllated to test.java.
- **Compilation of JSP page :**

  Here the generated java servlet file (test.java) is compiled to a class file (test.class).
- **Classloading :**

  Servlet class which has been loaded from JSP source is now loaded into container.
- **Instantiation :**

  Here instance of the class is generated. The container manages one or more instance by providing response to requests.
- **Initialization :**

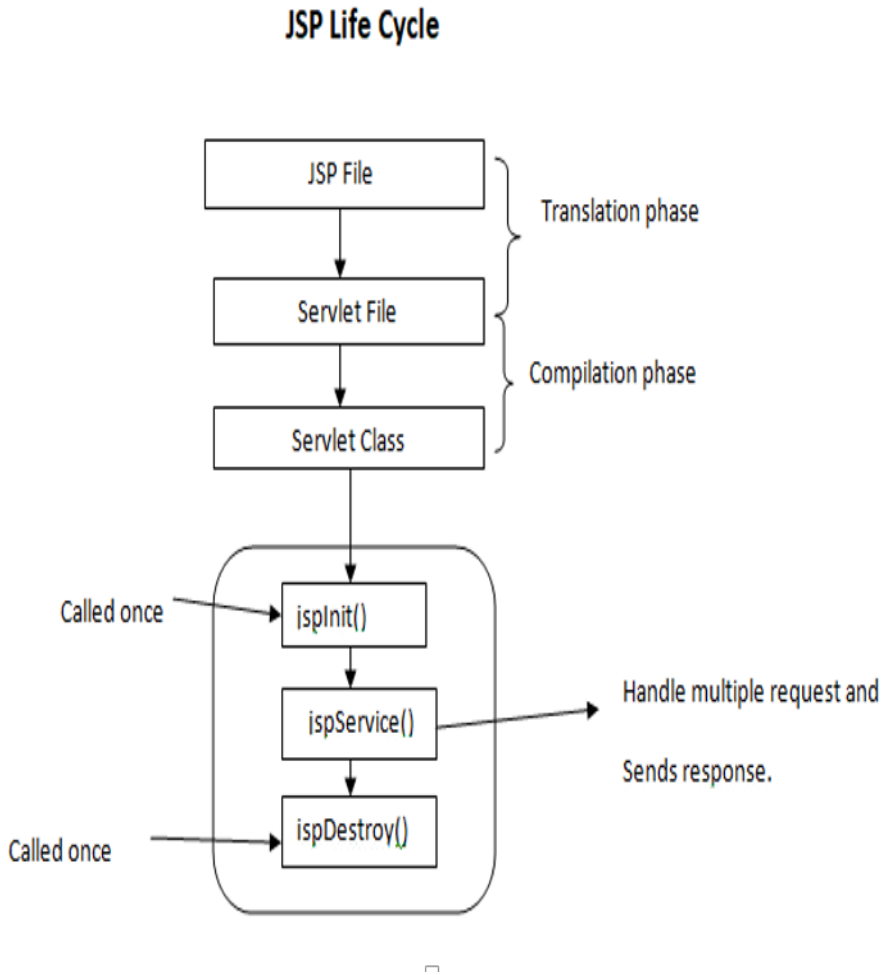
  jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

- **Request processing :**

  _jspService() method is used to serve the raised requests by JSP.It takes request and response object as parameters.This method cannot be overridden.

- **JSP Cleanup :**

  In order to remove the JSP from use by the container or to destroy method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.



JSP Scope Object

- The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object. Every object created in a JSP page will have a scope. Object scope in JSP is segregated into four parts and they are page, request, session and application.
- **page**
  ‘page’ scope means, the JSP object can be accessed only from within the same page where it was created. The default scope for JSP objects created using <jsp:useBean> tag is page. JSP implicit objects out, exception, response, pageContext, config and page have ‘page’ scope.

- **request**
  A JSP object created using the ‘request’ scope can be accessed from any pages that serves that request. More than one page can serve a single request. The JSP object will be bound to the request object. Implicit object request has the ‘request’ scope.
- **session**
  ‘session’ scope means, the JSP object is accessible from pages that belong to the same session from where it was created. The JSP object that is created using the session scope is bound to the session object. Implicit object session has the ‘session’ scope.
- **application**
  A JSP object created using the ‘application’ scope can be accessed from any pages across the application. The JSP object is bound to the application object. Implicit object application has the ‘application’ scope.

JSP Implicit Object

- These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared. JSP Implicit Objects are also called **pre-defined variables**.
- Following table lists out the nine Implicit Objects that JSP supports –

1. Request
2. Response
3. Out
4. Session
5. Application
6. Config
7. pageContext
8. Page
9. Exception

- **The request Object**

1. The request object is an instance of a javax.servlet.http.HttpServletRequest object. Each time a client requests a page the JSP engine creates a new object to represent that request.
2. The request object provides methods to get the HTTP header information including form data, cookies, HTTP methods etc.

- **The response Object**

1. The response object is an instance of a javax.servlet.http.HttpServletResponse object. Just as the server creates the request object, it also creates an object to represent the response to the client.
2. The response object also defines the interfaces that deal with creating new HTTP headers. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes, etc.

- **The out Object**

1. The out implicit object is an instance of a javax.servlet.jsp.JspWriter object and is used to send content in a response.
2. The initial JspWriter object is instantiated differently depending on whether the page is buffered or not. Buffering can be easily turned off by using the buffered = 'false' attribute of the page directive.
3. The JspWriter object contains most of the same methods as the java.io.PrintWriter class. However, JspWriter has some additional methods designed to deal with buffering. Unlike the PrintWriter object, JspWriter throws IOExceptions.

4. Following table lists out the important methods that we will use to write boolean char, int, double, object, String, etc.

S.No.	Method & Description
1	out.print(dataType dt) Print a data type value
2	out.println(dataType dt) Print a data type value then terminate the line with new line character.
3	out.flush() Flush the stream.

- **The session Object**

1. The session object is an instance of javax.servlet.http.HttpSession and behaves exactly the same way that session objects behave under Java Servlets.
2. The session object is used to track client session between client requests

- **The application Object**

1. The application object is direct wrapper around the ServletContext object for the generated Servlet and in reality an instance of a javax.servlet.ServletContext object.
2. This object is a representation of the JSP page through its entire lifecycle. This object is created when the JSP page is initialized and will be removed when the JSP page is removed by the jspDestroy() method.
3. By adding an attribute to application, you can ensure that all JSP files that make up your web application have access to it.

- **The config Object**

1. The config object is an instantiation of javax.servlet.ServletConfig and is a direct wrapper around the ServletConfig object for the generated servlet.
2. This object allows the JSP programmer access to the Servlet or JSP engine initialization parameters such as the paths or file locations etc.
3. The following config method is the only one you might ever use, and its usage is trivial −
4. config.getServletName();

This returns the servlet name, which is the string contained in the <servlet-name> element defined in the WEB-INF\web.xml file.

- **The pageContext Object**

1. The pageContext object is an instance of a javax.servlet.jsp.PageContext object. The pageContext object is used to represent the entire JSP page.
2. This object is intended as a means to access information about the page while avoiding most of the implementation details.
3. This object stores references to the request and response objects for each request. The application, config, session, and out objects are derived by accessing attributes of this object.
4. The pageContext object also contains information about the directives issued to the JSP page, including the buffering information, the errorPageURL, and page scope.
5. The PageContext class defines several fields, including PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, and APPLICATION_SCOPE, which identify the four scopes. It also supports more than 40 methods, about half of which are inherited from the javax.servlet.jsp.JspContext class.
6. One of the important methods is removeAttribute. This method accepts either one or two arguments. For example, pageContext.removeAttribute ("attrName") removes the attribute from all scopes, while the following code only removes it from the page scope −
7. pageContext.removeAttribute("attrName", PAGE_SCOPE);

- **The page Object**

1. This object is an actual reference to the instance of the page. It can be thought of as an object that represents the entire JSP page.
2. The page object is really a direct synonym for the this object.

- **The exception Object**

1. The exception object is a wrapper containing the exception thrown from the previous page. It is typically used to generate an appropriate response to the error condition.

JSP Scripting Elements

- The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:
- scriptlet tag
- expression tag
- declaration tag
- JSP scriptlet tag

- A **scriptlet tag** is used to execute java source code in JSP. Syntax is as follows:

<% java source code %>

- The code placed within **JSP expression tag** is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.
- Syntax of JSP expression tag

<%= statement %>

- The **JSP declaration tag** is used to declare fields and methods.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
- The syntax of the declaration tag is as follows:

<%! field or method declaration %>

JSP Directives

- JSP directives are the elements of a JSP source code that guide the web container on how to translate the JSP page into it’s respective servlet.

Syntax : *<%@ directive attribute = “value” %>*

- Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.
- The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

- Different types of JSP directives : There are three different JSP directives available. They are as follows:

1. Page Directives : JSP page directive is used to define the properties applying the JSP page, such as the size of the allocated buffer, imported packages and classes/interfaces, defining what type of page it is etc.

- The syntax of JSP page directive is as follows:

<%@ page attribute = “value” %>

- Different properties/attributes : The following are the different properties that can be defined using page directive
- **import:** This tells the container what packages/classes are needed to be imported into the program.

Syntax: *<%@ page import = “value” %>*

- **contentType:** This defines the format of data that is being exchanged between the client and the server. It does the same thing as the setContentType method in servlet used to.

Syntax: *<%@ page contentType = “value” %>*

- **info:** Defines the string which can be printed using the ‘getServletInfo()’ method.

Syntax: *<%@page contentType=“text/html” %>*

<= getServletInfo() %>

- **buffer:** Defines the size of the buffer that is allocated for handling the JSP page. The size is defined in Kilo Bytes.

Syntax: <%@ page buffer=“size in kb” %>

- **errorPage:** Defines which page to redirect to, in case the current page encournters an exception.

Syntax: <%@page errorPage=“error.jsp” %>

- **iserrorPage:** It classifies whether the page is an error page or not. By classifying a page as an error page, it can use the implicit object ‘exception’ which can be used to display exceptions that have occured.

Syntax: <%@page iserrorPage=“true/false” %>

2. Include directive : JSP include directive is used to include other files into the current jsp page. These files can be html files, other sp files etc. The advantage of using an include directive is that it allows code re-usability.

The syntax of an include directive is as follows: *<%@ include file = “file location” %>*

3. Taglib Directive : The taglib directive is used to mention the library whose custom-defined tags are being used in the JSP page. It’s major application is JSTL(JSP standard tag library).

Syntax: *<%@ taglib uri=“library url” prefix=“the prefix to identify the tags to this library with”%>*

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

- In the above code, we’ve used to taglib directive to point to the JSTL library which is a set of some custom-defined tags in JSP that can be used in place of the scirptlet tag (). The prefix attribute is used to define the prefix that is used to identify the tags of this library.
- Here, the prefix c is used in the <c:out> tag to tell the container that this tag belongs to the library mentioned above.

JSP Actions

- These actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.
- There is only one syntax for the Action element, as it conforms to the XML standard −

<jsp: action_name attribute="value"/>

- Action elements are basically predefined functions

Standard Action

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.