Presented By:

Corporate Trainer

Mallikarjuna G D

gdmallikarjuna@gmail.com

CONTENTS

- ✓ INTRODUCTION
- ✓ WEB PROGRAMMING
- ✓ SERVLET
- ✓ JSP
- ✓ CASE STUDY
- ✓ MCQ
- ✓ INTERVIEW PREPARATION
- ✓ CAPSTONE PROJECT

- World Wide Web:
  HTTP (Hypertext Transfer Protocol) is similar to FTP because it is a protocol to transfer files from the server to the client. HTTP was created in conjunction with related HTML standards.

- HTML (Hypertext Markup Language) is a document display language that lets users link from one document to another.

- HTML permits images and other media objects to be embedded in an HTML document. The media objects are stored in files on a server.

- HTTP also retrieves these files. Therefore HTTP can be used to transmit any file that confirms to Multipurpose Internet Mail Extensions (MIME) specification

- ## World Wide Web:

  HTTP (Hypertext Transfer Protocol) is similar to FTP because it is a protocol to transfer files from the server to the client. HTTP was created in conjunction with related HTML standards.

- ## Web Browsers and Web Servers:

  To view an HTML document with rich media content, a graphical user interface was built on top of the client-side HTTP.
  This GUI is called a web browser. The server-side HTTP component is called a web server.

- ## Web Applications:

  Early in the development of HTML, the designers created a mechanism to permit a user to invoke a program on the web server. This mechanism was called as Common Gateway Interface (CGI). When a website includes CGI processing, this is called a web application.

CGI PROGRAMS ON THE WEBSERVER

- The browser needs to send data to the CGI program on the server. The CGI specification defines how the data is packaged and sent in the HTTP request to the server. This data is usually typed into the web browser in an HTML form.

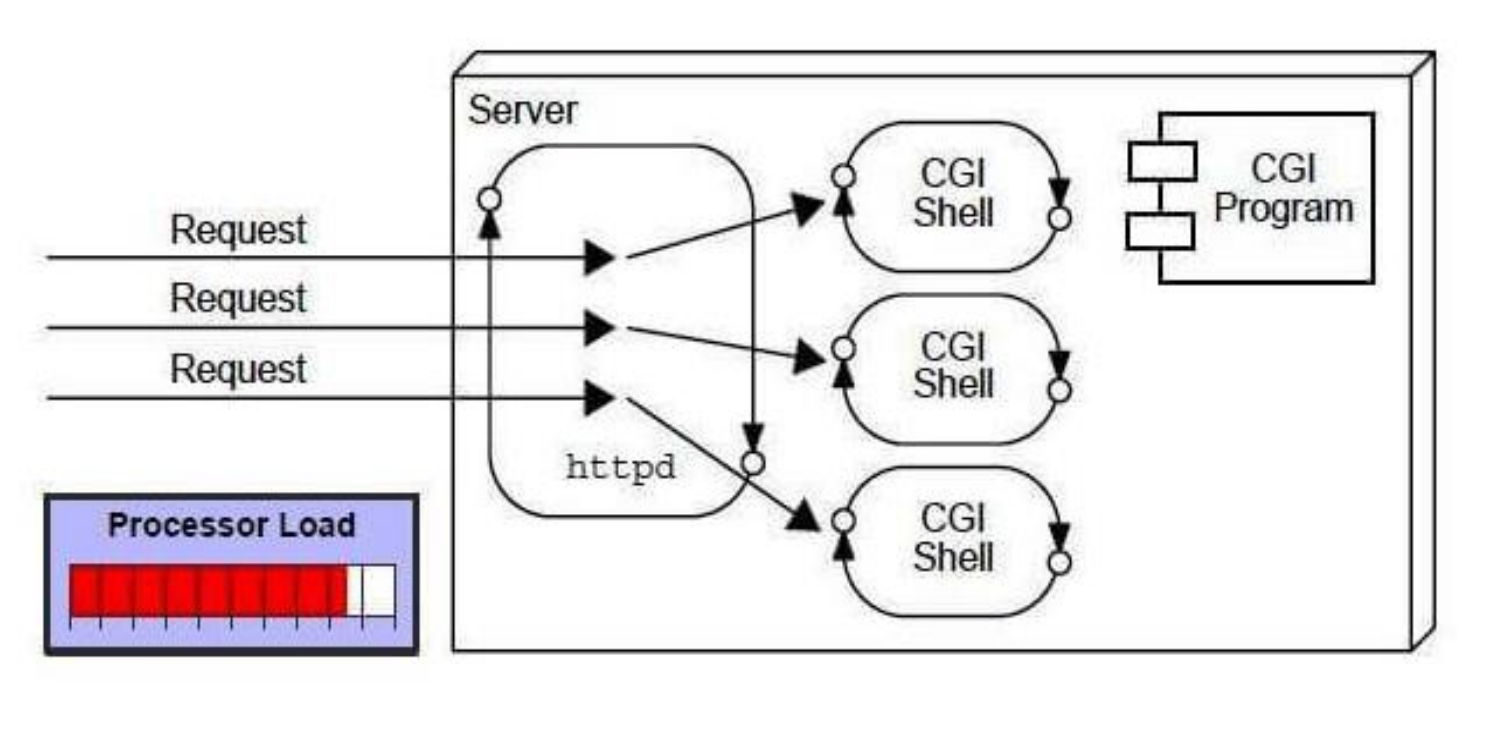
- The URL determines which CGI program to execute.
This might be a script or an executable file.

- The CGI program parses the CGI data in the request, processes the data, and generates a response.

- The CGI response is sent back to the web server, which wraps the response in an HTTP response.

- The HTTP response is sent back to the web browser.

CGI PROGRAMS ON THE WEBSERVER



ADVANTAGES

- The Programs can be written in a variety of languages, although they are primarily written in Perl.
- A CGI program with bugs does not crash the web server.
- Programs are easy for a web designer to reference. When the script is written, the designer can reference it in one line in a web page.
- Because CGI programs execute in their own OS Shell, these programs do not have concurrency conflicts with other HTTP requests executing the same CGI program.
- All service provides support CGI programs.

DISADVANTAGES

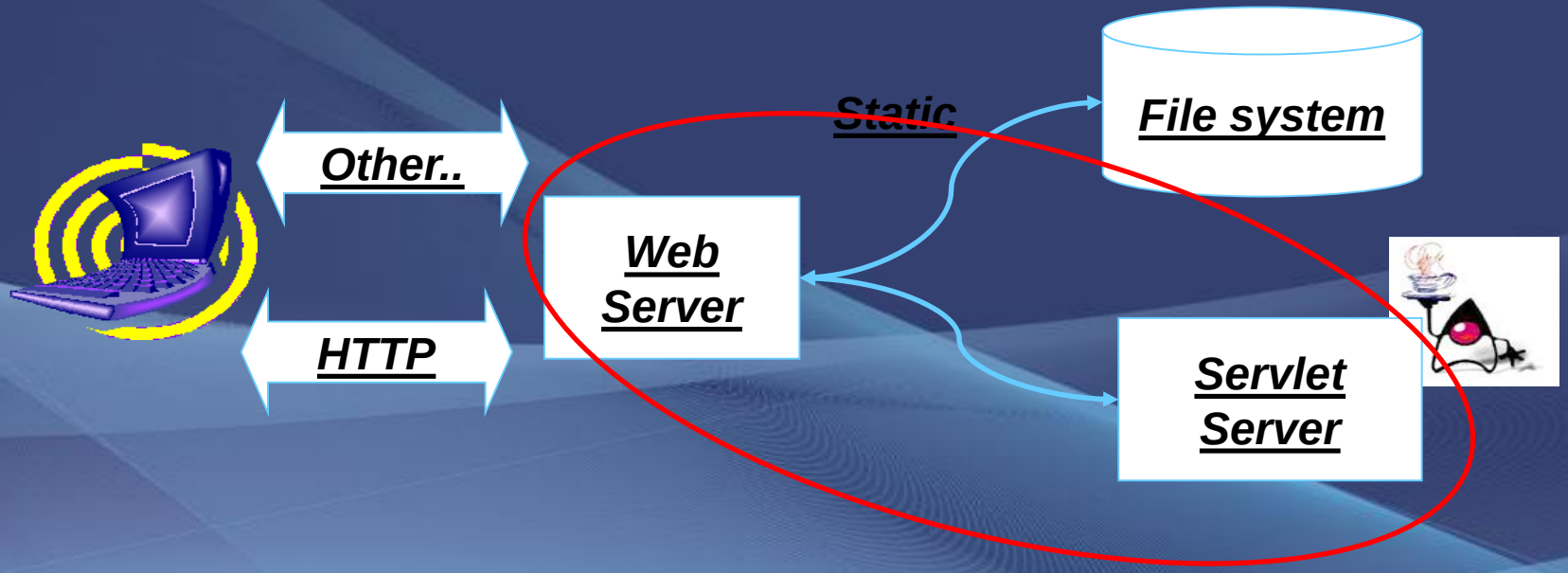
- The response time of CGI programs is high because CGI programs execute in their own OS shell. The creation of an OS shell is a heavyweight activity for the OS.
- CGI is not scalable.
- The languages for CGI are not always secure or object-oriented.
- The CGI script has to generate an HTML response, so the CGI code is mingled with HTML. This is not good separation of presentation and business logic.
- Scripting languages are often platform-dependent.

WHAT IS SERVLET ?

- The Servlet is a program that runs on server and acts as a middle layer between a request coming from web-browser or other HTTP Clients and database or applications on the Server.
- Improved substitute for CGI Scripts.
- Are executed in the context of web server.
- Part of JEE (Servlet API 2.4)
- Helps client-server communication.

JOB OF SERVLET ?

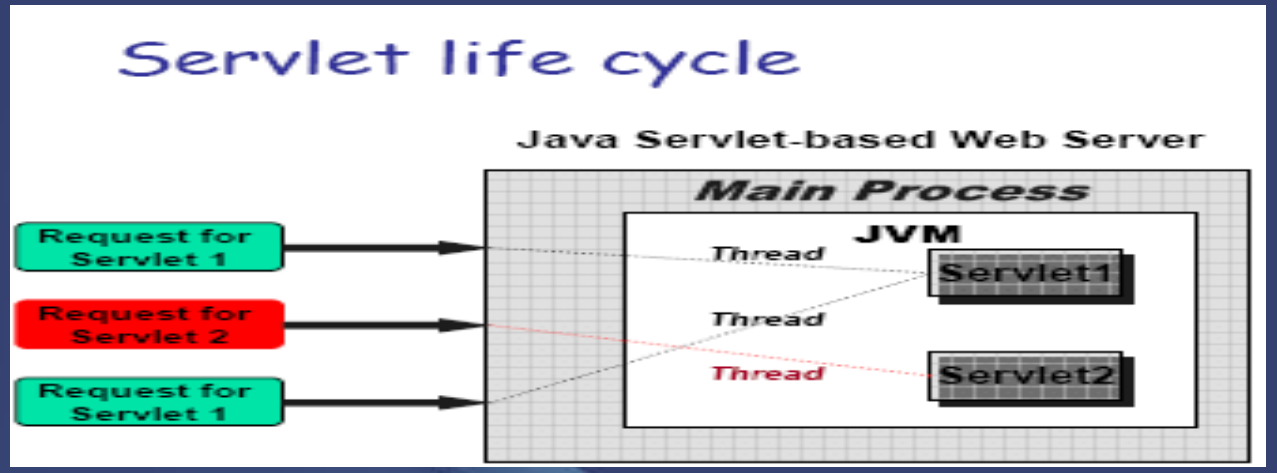
- Read explicit data sent by client (form data)
- Read implicit data sent by client (request header)
- Generate the result
- Send the explicit data back to client (e.g. HTML format)
- Send the implicit data to client (status code and response header)



Tomcat = Web Server + Servlet Server

ADVANTAGES?

- Efficient/Scalable-handled by a separate thread within the web-server process.



- Convenient - lot of high-level utilities.
- Powerful – Sharing data, pooling, and persistence.
- Portable – Run virtually on all operating systems and servers.
- Security – No shell escapes, no buffer flow.
- Inexpensive.

DISADVANTAGES?

- Servlets often contain both business logic and presentation logic. Presentation logic is anything that controls how the application presents information to the user. Generating the HTML response within the servlet code is presentation logic.

  Business logic is anything that manipulates data to accomplish something such as storing data.
- Servlets must handle concurrency issues.
- Mixing presentation and business logic means that whenever a web page changes the servlets must be rewritten, recompiled and redeployed.
- This disadvantage led to the development of template pages, including Java Server Pages technology.

WHY BUILD PAGES DYNAMICALLY?

- The web page is based on data submitted by the user.
  e.g.:- result in search engines, online shopping so on.

- The web pages are derived from the data changes frequently.
  the e.g.:-weather report, stock exchange, news headline.

- The web pages are derived from data from databases or other server side resources.
  e.g.:- an e-commerce page shows the availability of a product with price information.

Servlet Container/Servlet Engine?

- The servlet container is compiled executable program. The container acts as an intermediator between the webserver and servlet in the container.
- The container loads, initializes, and executes the servlet. When a request arrives container maps to the servlet.
- The container is designed to perform well by serving a large number of requests. A container can hold any number of servlets, filters and listener.
- Both the container and object contained in it are multithreaded. The container handles multiple requests concurrently, and more than one thread may enter an object at a time. Therefore, each object within a container must be thread safe

There are three kinds of servlet engine
- standalone
- add on
- embedded

- Standalone Servlet Engines
  - Built-in support for servlets.
  - Hard to keep latest version of servlet.

- Add on servlet Servlet-engines
  - Plug-in to an existing servlets.

- Embeddable servlet engines
  - lightweight servlet-deployment platform that can be embedded in another application.

Non-commercial:

- Apache Tomcat (formerly Jakarta Tomcat) is an open source web container available under the Apache Software License.
- Apache Geronimo -is a full Java EE implementation by Apache.
- Jetty
- Jaminid - contains a higher abstraction than servlets.
- Enhydra
- Winstone - supports specification v2.4, has a focus on minimal configuration and the ability to strip the container down to only what you need.

Commercial:

- Apache BEA WebLogic Server or Weblogic Express, from BEA Systems
- Borland Enterprise Server
- GlassFish (open source)
- Java System Application Server, from Sun Microsystems
- Java System Web Server, from Sun Microsystems
- JBoss (open source)
- JRun, from Adobe Systems (formerly developed by Allaire Corporation

Commercial:

- LiteWebServer (open source)
- Oracle Application Server, from Oracle Corporation
- Orion Application Server, from IronFlare
- Caucho's Resin Server
- ServletExec, from New Atlanta Communications
- WebObjects, from Apple Inc.
- WebSphere, from IBM

- The Servlet API, contained in the Java package hierarchy javax.servlet, defines the expected interactions of a Web container and a servlet.

- A Web container is essentially the component of a Web server that interacts with the servlets. The Web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

- A Servlet is an object that receives a request and generates a response based on that request.

The basic servlet package defines Java objects to represent servlet requests and responses, as well as objects to reflect the servlet's configuration parameters and execution environment

JAVA Servlet API?

Servlet API history

Servlet API version	Released	Specification	Platform	Important Changes
Jakarta Servlet 6.0	Oct 15, 2021	6.0	Jakarta EE 10	remove deprecated features and implement requested enhancements
Jakarta Servlet 5.0	Oct 9, 2020	5.0	Jakarta EE 9	API moved from package `javax.servlet` to `jakarta.servlet`
Jakarta Servlet 4.0.3	Sep 10, 2019	4.0	Jakarta EE 8	Renamed from "Java" trademark
Java Servlet 4.0	Sep 2017	JSR 369	Java EE 8	HTTP/2
Java Servlet 3.1	May 2013	JSR 340	Java EE 7	Non-blocking I/O, HTTP protocol upgrade mechanism (WebSocket)[14]
Java Servlet 3.0	December 2009	JSR 315	Java EE 6, Java SE 6	Pluggability, Ease of development, Async Servlet, Security, File Uploading
Java Servlet 2.5	September 2005	JSR 154	Java EE 5, Java SE 5	Requires Java SE 5, supports annotation
Java Servlet 2.4	November 2003	JSR 154	J2EE 1.4, J2SE 1.3	web.xml uses XML Schema
Java Servlet 2.3	August 2001	JSR 53	J2EE 1.3, J2SE 1.2	Addition of `Filter`
Java Servlet 2.2	August 1999	JSR 902, JSR 903	J2EE 1.2, J2SE 1.2	Becomes part of J2EE, introduced independent web applications in .war files
Java Servlet 2.1	November 1998	2.1a	Unspecified	First official specification, added `RequestDispatcher`, `ServletContext`
Java Servlet 2.0	December 1997	N/A	JDK 1.1	Part of April 1998 Java Servlet Development Kit 2.0[15]
Java Servlet 1.0	December 1996	N/A		Part of June 1997 Java Servlet Development Kit (JSDK) 1.0[9]

Servlets are first standard to extension to java including two packages:

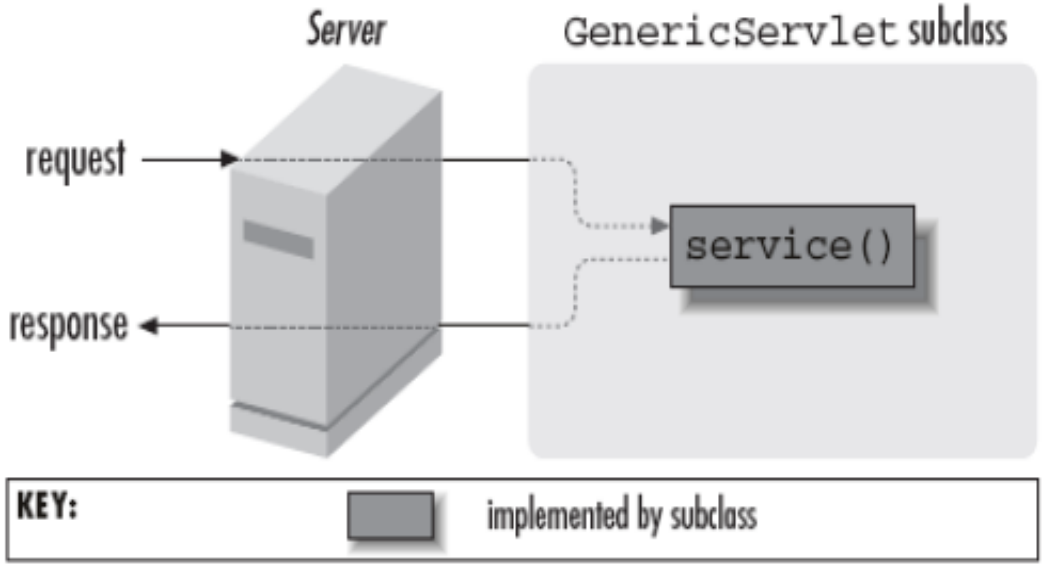
- -javax.servlet
- -javax.servlet.http

In general, servlet

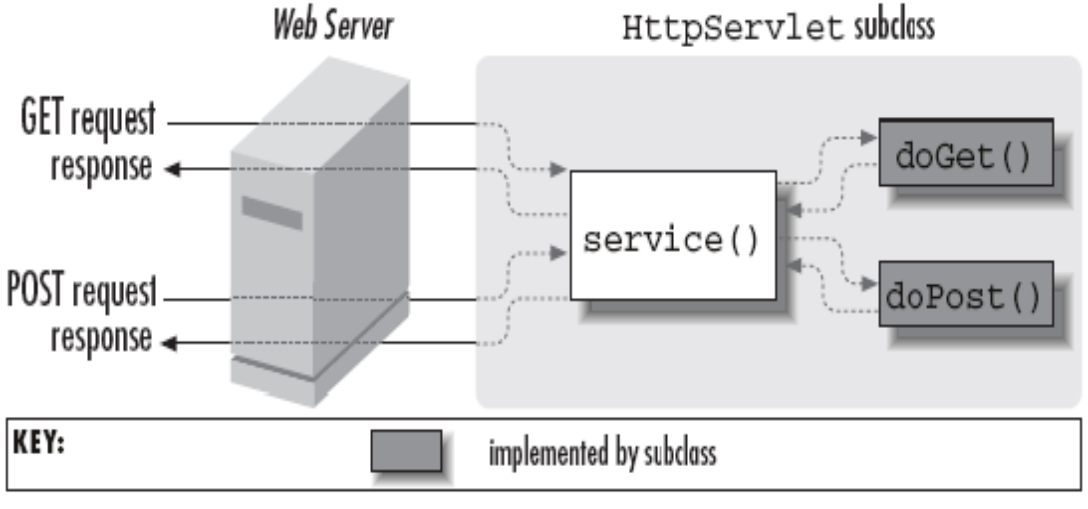
1. Generic Servlet
   - must extend javax.servlet.Servlet
   - protocol independent

2. Http Servlet
   - must extend javax.servlet.http.HttpServlet
   - Handling Http request/reply

Generic Servlets



HTTP Servlets



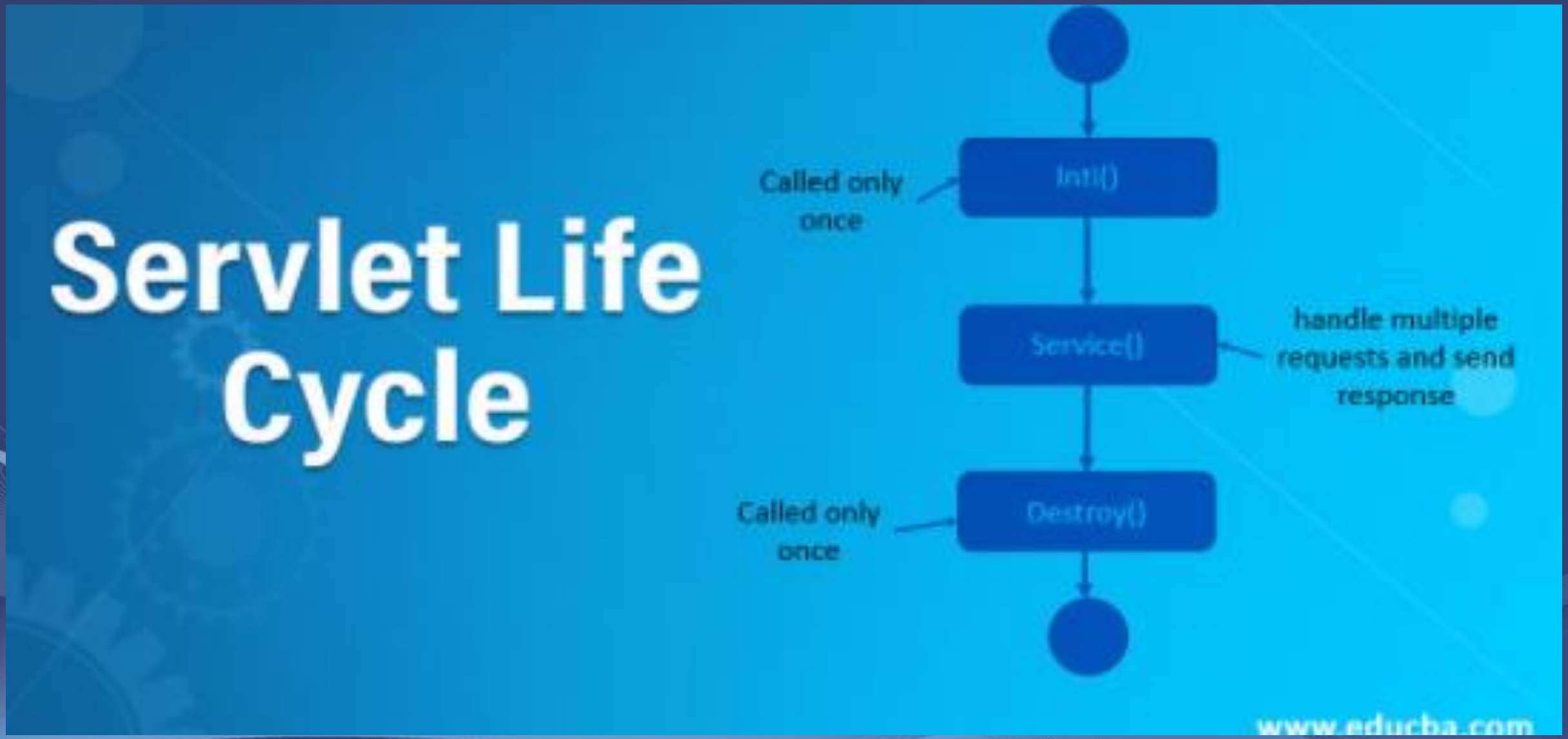
- The life cycle of a servlet is controlled by the container in which the servlet has been deployed.

When a request is mapped to a servlet, the container performs the following steps:

- If an instance of the servlet does not exist, the Web container loads the servlet class. Creates an instance of the servlet class. Initializes the servlet instance by calling the init method.

- Invokes the service method, passing a request and response object.

- If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method.


Servlet Life Cycle
Called only once
Init()
Service()
handle multiple requests and send response
Called only once
Destroy()
www.educba.com

The Java class that implements the Servlet interface

- Servlet Interface
  - Defines the Servlet's life cycle methods.
  - init(ServletConfig config)

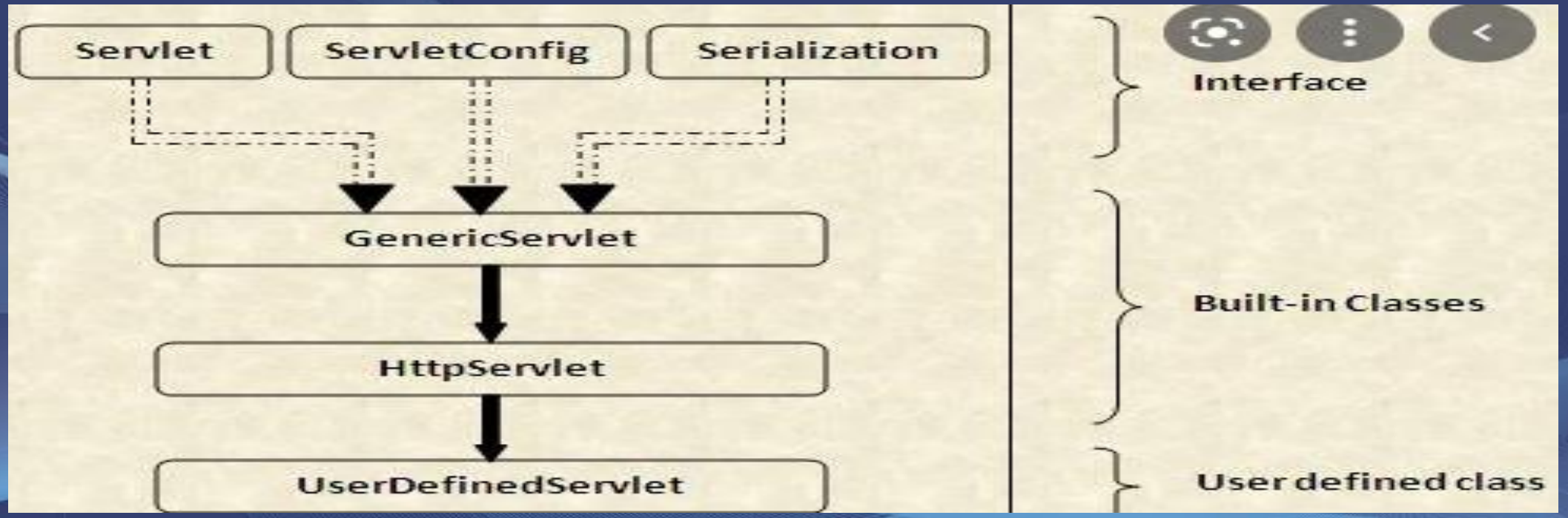
- service(ServletRequest req, ServletResponse res)
  - doGet handles GET request.
    - doPost handles POST request.

- destroy()
  Called by the servlet container before servlet is taken out of service

SingleThreadModel?

- The Container may send multiple request to single instance , using java threads. So service methods (doGet, doPost..) are all thread safe. It means two or more threads operating without interfere each other.
- If service methods not thread safe. We have to use single thread model.
- The single thread model ensures that only one service method runs at time

```
Servlet interface {
    void init(ServletConfig sc) throws ServletException;
        void service(ServletRequest req,ServletResponse res) throws
                ServletException,IOException;
         void destroy();
}
```



Methods of Servlet

Method	Description
`void init( ServletConfig config )`	The servlet container calls this method once during a servlet's execution cycle to initialize the servlet. The `ServletConfig` argument is supplied by the servlet container that executes the servlet.
`ServletConfig getServletConfig( )`	This method returns a reference to an object that implements interface `ServletConfig`. This object provides access to the servlet's configuration information such as servlet initialization parameters and the servlet's `ServletContext`, which provides the servlet with access to its environment (i.e., the servlet container in which the servlet executes).
`String getServletInfo()`	This method is defined by a servlet programmer to return a string containing servlet information such as the servlet's author and version.
`void service( ServletRequest request, ServletResponse response )`	The servlet container calls this method to respond to a client request to the servlet.
`void destroy()`	This "cleanup" method is called when a servlet is terminated by its servlet container. Resources used by the servlet, such as an open file or an open database connection, should be deallocated here.

HTTP SERVLET?

- The Override the servlet class

- There are two types of requests- Get and Post
  - doGet responds to Get request
  - doPost responds to Post request

- It has HttpServletRequest and HttpServletResponse Objects

Method	Description
doDelete	Called in response to an HTTP *delete* request. Such a request is normally used to delete a file from a server. This may not be available on some servers, because of its inherent security risks (e.g., the client could delete a file that is critical to the execution of the server or an application).
doHead	Called in response to an HTTP *head* request. Such a request is normally used when the client only wants the headers of a response, such as the content type and content length of the response.
doOptions	Called in response to an HTTP *options* request. This returns information to the client indicating the HTTP options supported by the server, such as the version of HTTP (1.0 or 1.1) and the request methods the server supports.
doPut	Called in response to an HTTP *put* request. Such a request is normally used to store a file on the server. This may not be available on some servers, because of its inherent security risks (e.g., the client could place an executable application on the server, which, if executed, could damage the server—perhaps by deleting critical files or occupying resources).
doTrace	Called in response to an HTTP *trace* request. Such a request is normally used for debugging. The implementation of this method automatically returns an HTML document to the client containing the request header information (data sent by the browser as part of the request).

HTTPServletRequest Interface?

- The Web server
  - creates an HttpServletRequest object.
  - passes it to the servlet's service method.

- HttpServletRequest object contains the request from the client

Method	Description
String getParameter(String name)	Obtains the value of a parameter sent to the servlet as part of a get or post request. The name argument represents the parameter name.
Enumeration getParameterNames()	Returns the names of all the parameters sent to the servlet as part of a post request.
String[] getParameterValues(String name)	For a parameter with multiple values, this method returns an array of strings containing the values for a specified servlet parameter.
Cookie[] getCookies()	Returns an array of Cookie objects stored on the client by the server. Cookie objects can be used to uniquely identify clients to the servlet.
HttpSession getSession(boolean create)	Returns an HttpSession object associated with the client's current browsing session. This method can create an HttpSession object (true argument) if one does not already exist for the client. HttpSession objects are used in similar ways to Cookies for uniquely identifying clients.

- The Web server
  - creates an HttpServletResponse object.
  - passes it to the servlet's service method.

Method	Description
`void addCookie( Cookie cookie )`	Used to add a `Cookie` to the header of the response to the client. The `Cookie`'s maximum age and whether `Cookies` are enabled on the client determine if `Cookies` are stored on the client.
`ServletOutputStream getOutputStream()`	Obtains a byte-based output stream for sending binary data to the client.
`PrintWriter getWriter()`	Obtains a character-based output stream for sending text data to the client.
`void setContentType( String type )`	Specifies the MIME type of the response to the browser. The MIME type helps the browser determine how to display the data (or possibly what other application to execute to process the data). For example, MIME type `"text/html"` indicates that the response is an HTML document, so the browser displays the HTML page.

Handling client request: HTTP request header?

General:

getHeader, getHeaders, getHeaderNames.

Specialized:

getCookies, getAuthType, getRemoteUser, getContentLength, getContentType, getDateHeader, getIntHeader

Related Info:

getMethod, getRequestURI, getProtocol

Common HTTP Request Header 1.1?

- Accept
  - MIME Types of browser can handle.
  - Can send different content to different clients.

- Accept-Encoding
  - Indicate encoding (e.g gzip) browser can handle.

- Authorization
  - User identification for password protected pages.

- Connection
    - In Http 1.1, keep alive means browser can handle persistent connection. Persistent connection is default. Persistent means same socket is reused for same type of client request.

Common HTTP Request Header 1.1?

- Cookie
  - Give cookies previously sent to client.

- Host
  - It gives host given in original URL.

- If-Modified-Since
  - Client wants pages, only after changed on some specified dates.
- Referrer
  - URL of referring web page.
  - Useful for tracking traffic; logged by many servers.

- User Agent
  - String identifying the browser making request.

Purpose: The variety of status code that are essentially indicate failure.
e.g.:- HTTP/1.1 200 OK

- Changing a status code perform a lot of tasks.
  - Forward client to another page.
  - Indicate a missing resource.
  - Instruct a browser to use cached copy.
- Set status before sending document.
  public void setStatus (int statusCode)
  - use constant for the code. Not an explicit int.
    constants are HttpServletResponse.
  - Names derived from standard message.
    e.g SC_OK, SC_NOT_FOUND so on.
  - public sendError(int code, String msg)
  - Wraps the message inside HTML document

- Public void sendRedirect(String url)
  - Relative url is permitted in 2.2/2.3
    - Also sets location header.

Common HTTP 1.1 Status codes:

- 200(OK)
  - Everything is fine, document follows, default for servlets.

- 204(No content)
  - Browser should keep displaying previous document.

- 301(Moved document)
  - Requested document moved elsewhere.
  - Browsers go to new location automatically.

- 401(unauthorized)
  Browser tried to access the protected page without proper authorization header.
- 404(Not found)
  No such page.

In general

100-199 -> codes in the 100s are informational, indicating that the client should respond with some other action.

200-299 -> values in the 200's signify that the request was successful.

300-399 -> values in the 300's are used for files that have moved and usually include location header indicating the new address.

400-499 -> values in the 400's indicate an error by the client.

500-599 ->codes in the 500's signify an error by the server.

- Common MIME TYPE

Type	Meaning
application/msword	Microsoft word document
application/pdf	Pd f file
application/vnd_ms_excel	Excel
application/vnd_ms_powerpoint	Power point
application/zip	Zip archive
text/css	HTML Cascading sheet
text/html	HTML Document
text /plain	Plain/text
Image/gif	GIF image
Image/png	PNG image
Video/mpeg	MPEG Video clip

- A filters are Java classes in which it does preprocess and post-processing of the servlet. It intercepts the request and does pre-requisite or post-requisite activities of the servlet.
- It may be authentication, logging, data compression, encryption and decryption, input validation so on.
- It is pluggable and Independent
- Filter API includes
  - Filter
  - Filter Chain
  - Filter Config
- Filter
  - Init method – initialize filter executed only once
  - doFilter-invokes every time user requests resources and perform the task
  - Destroy – only once when the filter is taken out of the service
- Filter chain – invoke the next filter or servlet

- A server site typically needs to maintain two types of persistent(remembered) information:
  - Information about the Session.
  - Information about the User.

Servlet capabilities:

Servlets, like Applets, can be trusted or untrusted.

- A servlet can use a unique ID to store and retrieve information about a given session.
- User information usually requires a login ID and a password.
  - Since servlets don't quit between requests, any servlet can maintain information in its internal data structures, as long as the server keeps running.

A trusted servlet can read and write files on the server, hence can maintain information about sessions and users even when the server is stopped and restarted.

- An untrusted servlet will lose all information when the servlet or server stops for any reason.
  - This is sometimes good enough for session information.
  - This is almost never good enough for user information.
- HTTP is stateless: When it gets a page request, it has no memory of any previous requests from the same client.

- A Cookies are small files that the servlet can store on the client computer, and retrieve later.
- URL rewriting: You can append a unique ID after the URL to identify the user.
- Hidden <form> fields can be used to store a unique ID.
- Java's Session Tracking API can be used to do most of the work for you

- A Cookies are small bits of textual information that a web server sends to the browser and that the browser returns unchanged when later visiting the same website or domain.

- Cookies are a security threat.

- Cookies can be a privacy threat.
  - Cookies can be used to customize advertisements.
  - Outlook Express allows cookies to be embedded in the email.
  - A servlet can read your cookies.
    Incompetent companies might keep your credit card info in a cookie.
    Netscape lets you refuse cookies to sites other than that to which you connected.

- A Cookies are a very useful tool in maintaining state (persistence) on the Web. Unless something special is done, web servers are only aware of users when a transaction -- sending or receiving information -- is in process. The moment the transaction is complete the server forgets about the user and makes no attempt to correlate subsequent transactions with previous exchanges.

- Since the web protocol, HTTP, is a "stateless" (non-persistent) protocol, it is impossible to differentiate between visits to a given web site, unless the server can somehow "mark" a visitor.

- This is done by storing a piece of information in the visitor's browser, called a "cookie". Cookies can contain database information, custom page settings, or just about anything that would make a site customizable.

- import javax.servlet.http.*;
- Constructor: Cookie(String name, String value)
- Assuming request is an HttpServletRequest and response is an HttpServletResponse,

```
response.addCookie(cookie);
Cookie[ ] cookies = request.getCookies();
  String name = cookies[i].getName();
  String value = cookies[i].getValue();
```

- There are, of course, many more methods in the HttpServletRequest, HttpServletResponse, and Cookie classes in the javax.servlet.http package.
- **public void setComment(String purpose)**
  - **public String getComment()**
- **public void setMaxAge(int expiry)**
  - **public int getMaxAge()**
  - **Max age in seconds after which cookie will expire.**
  - **If expiry is negative, delete when browser exits.**
  - **If expiry is zero, delete cookie immediately.**
- **setSecure(boolean flag)**
  - **public boolean get Secure()**
  - **Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.**

Here is the code snippet for initializing a cookie and adding the same to the response object.

```
.....
Cookie cookie = new Cookie ("otncookiename",userName);
cookie.setMaxAge(86400);
response.addCookie(cookie);
Cookie onemorecookie = new     Cookie("otncookiepassword",password);
onemorecookie.setMaxAge(86400);
response.addCookie (onemorecookie);
.....
```

Code snippet:

Login.html

```
<form name="login" method="post" action="./LoginCookieServlet">
<p>please enter login credentials</p>
User Name : <input type="text" name="txtUserName"> <br>
password : <input type="password" name="txtPassword"> <input
type="submit" value='submit'>
</form>
```

LoginCookieServlet

```
String name= request.getParameter("txtUserName");
String pwd = request.getParameter("txtPassword");
Cookie userNameCookie = new Cookie("USERNAME", name);
response.addCookie(userNameCookie);
Cookie passwordCookie = new Cookie("PASSWORD",pwd);
response.addCookie(passwordCookie);
request.getRequestDispatcher("/LoginNextCookie").forward(request, response);
```

Code snippet:

LoginNextCookie

```
Cookie[] cookies = request.getCookies();
pw.println("cookies[0].getValue()"+cookies[0].getValue());
pw.println("cookies[1].getValue()"+cookies[1].getValue());

for(Cookie cookie: cookies) {
pw.println(cookie.getValue());
}
```

LIMITATIONS

- The browser should be enabled to accept cookies.
- Cookie values must never have spaces, commas or semicolons.
- Cookies can store upto 4KB of value.

- The URL rewriting is another way to support anonymous session tracking.
- With URL rewriting, every local URL the user might click on is dynamically modified, or rewritten, to include extra information.
- The extra information can be in the form of extra path information, added parameters, or some custom, server-specific URL change.
- Due to the limited space available in rewriting a URL, the extra information is usually limited to a unique session ID.

- code snippet

Loginurlrewriting.html

```
<form name="login" method="post" action="./URLRewriting">
<p>please enter login credentials</p>
User Name : <input type="text" name="txtUserName"> <br>
password : <input type="password" name="txtPassword"> <input
type="submit" value='submit'>
</form>
```

URLRewriting.java

```
response.getWriter().append("Served at: ").append(request.getContextPath());
String name= request.getParameter("txtUserName");
String pwd = request.getParameter("txtPassword");
request.getRequestDispatcher("/LoginUrlReceive?uname="+name+"&&pwd="+pwd).forward(request, response);
```

LoginUrlReceive.java

```
String username = request.getParameter("uname");
String password = request.getParameter("pwd");
pw.println("LoginUrlReceive username::"+username+"password::"+password);
```

- One way to support anonymous session tracking is to use hidden form fields. As the name implies, these are fields added to an HTML form that are not displayed in the client's browser. They are sent back to the server when the form that contains them is submitted. You include hidden form fields with HTML like this:

- <FORM ACTION="/servlet/MovieFinder" METHOD="POST">

...

<INPUT TYPE=hidden NAME="zip" VALUE="94040">

<INPUT TYPE=hidden NAME="level" VALUE="expert">

..

</FORM>

In a sense, hidden form fields define constant variables for a form. To a servlet receiving a submitted form, there is no difference between a hidden field and a visible field.

- code snippet

Loginhidform.html

```
<form name="login" method="post" action="./HiddenServlet">
<p>please enter login credentials</p>
User Name : <input type="text" name="txtUserName"> <br>
password : <input type="password" name="txtPassword"> <input
type="submit" value='submit'>
</form>
```

HiddenServlet.java

```
String name = request.getParameter("txtUserName");
String pwd = request.getParameter("txtPassword");
out.print("<form action='./LoginUrlReceive'>");
out.print("<input type='hidden' name='uname' value='" + name + "'>");
out.print("<input type='hidden' name='pwd' value='" + pwd + "'>");
out.print("<input type='submit' value='go'>");
out.print("</form>");
```

- code snippet

LoginUrlReceive.java

```
String username = request.getParameter("uname");
String password = request.getParameter("pwd");
pw.println("LoginUrlReceive username::"+username+"password::"+password);
```

- The session tracking API is in javax.servlet.http.HttpSession and is built on top of cookies.

- To use the session tracking API:
  - - Create a session:
    - HttpSession session = request.getSession();
      - Returns the session associated with this request
      - If there was no associated session, one is created.

  - - Store information in the session and retrieve it as needed:
    - session.setAttribute (name, value);
    - Object obj = getAttribute(name);

- Session information is automatically maintained across requests

- getId - This method returns the unique identifier generated for each session. It is sometimes used as the key name when there is only a single value associated with a session, or when logging information about previous sessions.

- isNew - This returns true if the client (browser) has never seen the session, usually because it was just created rather than being referenced by an incoming client request. It returns false for preexisting sessions.

- getCreationTime - This returns the time, in milliseconds since the epoch, at which the session was made. To get a value useful for printing out, pass the value to the Date constructor or the setTimeInMillis method of GregorianCalendar.

- getLastAccessedTime - This returns the time, in milliseconds since the epoch, at which the session was last sent from the client.

- getMaxInactiveInterval - This returns the amount of time, in seconds, that a session should go without access before being automatically invalidated. A negative value indicates that the session should never timeout.

- Public Object getValue(String name)
- Public Object getAttribute (String name)
- Public Object setValue(String name, Object value)
- Public Object setAttribute (String name, Object value)
- Public Object removeValue(String name)
- Public Object removeAttribute(String name)

Login.html

```
<form name="login" method="post" action="./AdminServletSession">
<p>please enter login credentials</p>
User Name : <input type="text" name="txtUserName"> <br>
password : <input type="password" name="txtPassword"> <input
type="submit" value='submit'>
</form>
```

AdminServletSession.java

```
HttpSession session = request.getSession(true);
String reqUsername = request.getParameter("txtUserName");
String reqPassword =  request.getParameter("txtPassword");
out.println("AdminServletSession::"+"username::"+reqUsername+ "password ::"+reqPassword);
session.setAttribute("USERNAME", reqUsername);
session.setAttribute("PASSWORD", reqPassword);
request.getRequestDispatcher("./HRServletSession").forward(request, response);
```

Loginsession.html

```
< HttpSession session = request.getSession(true);
String sessionUsername =(String) session.getAttribute("USERNAME");
String sessionPassword = (String) session.getAttribute("PASSWORD");
out.println("HRServletSession::"+"username::"+sessionUsername+ "password ::"+sessionPassword);
```

WHAT IS JSP?

- It is referred to as Java Server Pages and it is a server-side technology.
- Separates the graphical design from the dynamic content.
- It helps to develop dynamic web applications
- It allows inserting of java code within HTML pages
- It first converts to a servlet to process the client request and it may be referred to the as advanced technology of servlet

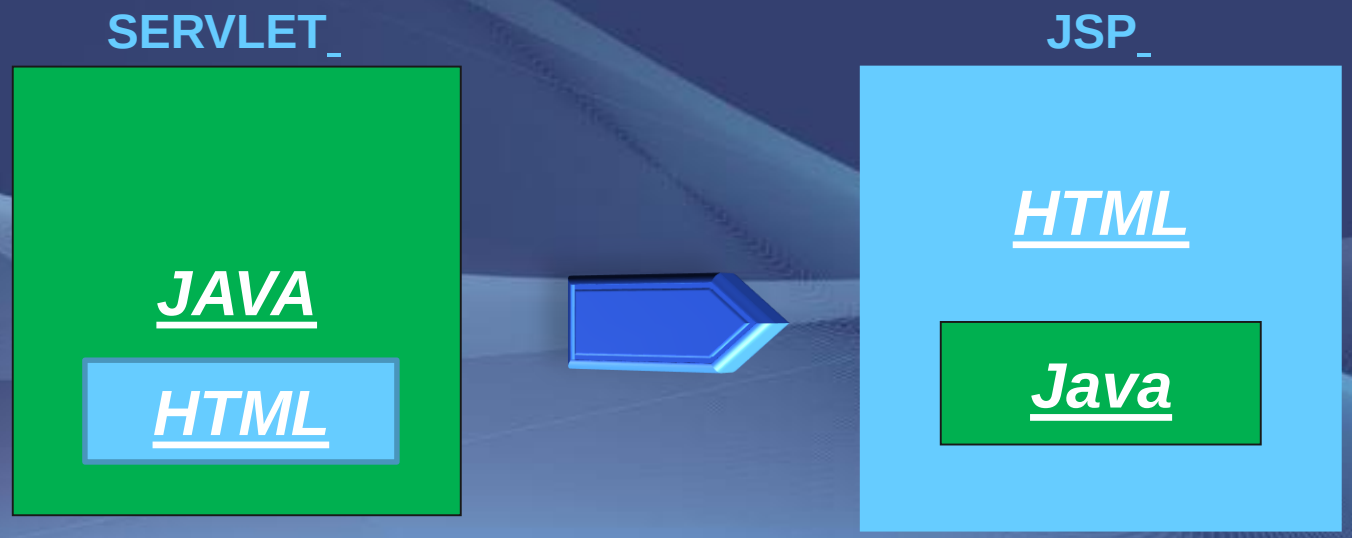
BENEFITS?

- Web designers can design and update pages without learning java language.
- Programmers for java can write code without dealing with web page design.
- Allows java to be embedded directly into an HTML page.
- Change in the code of JSP compiles automatically.
- Using custom tags and tag libraries length of code is reduced.

- In servlets,
  - HTML code is printed from java code.

vs

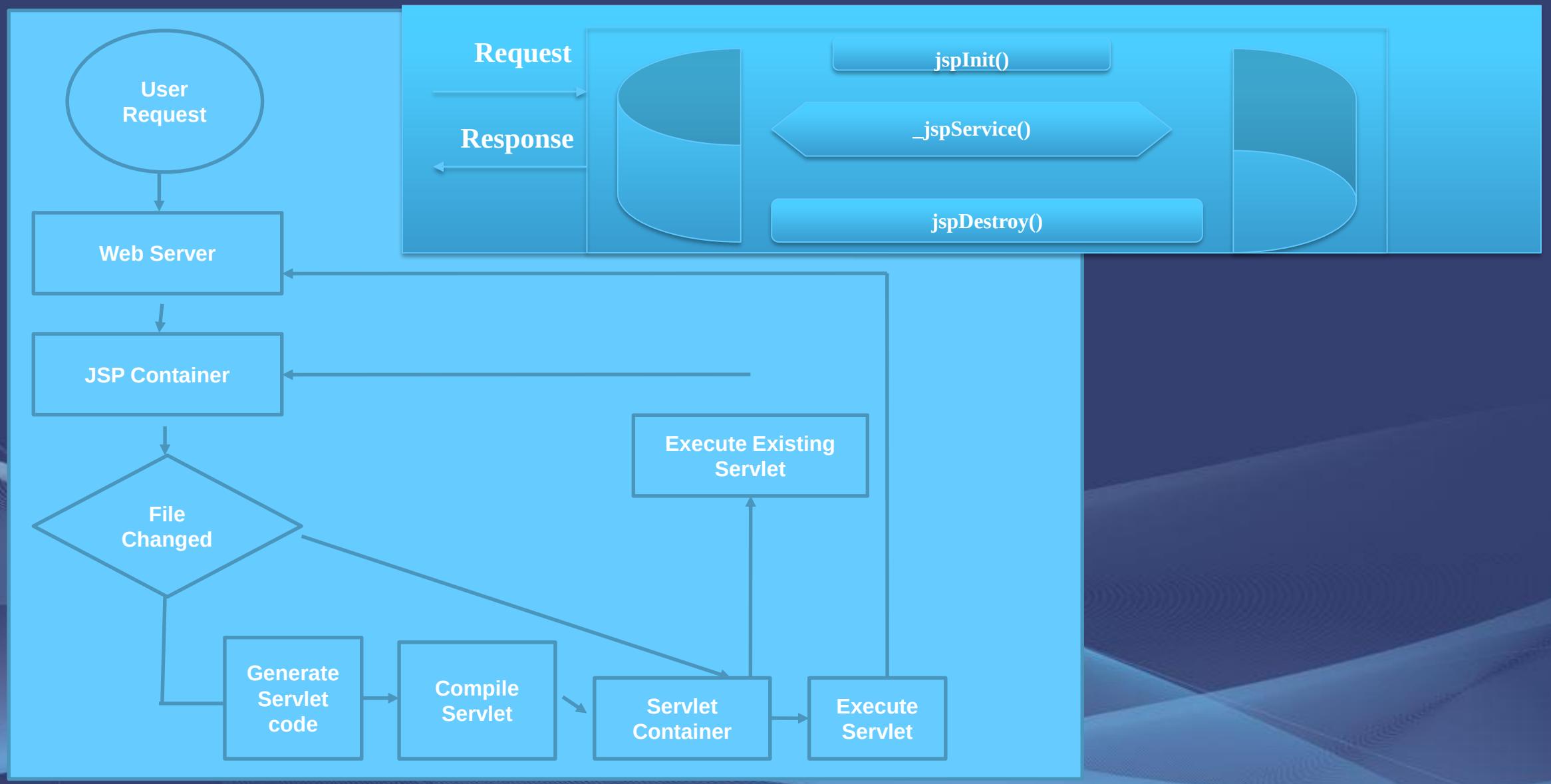
- In JSP pages
  - Java code is embedded in HTML code.



- **Web JSP-enabled web server**:
  - picks up .jsp page
  - parses it
  - converts it to runnable form
  - runs it.

- Converts page to a Java servlet (JspPage), with your code inside the _jspService() method
  - compiles it
  - runs servlet protocol.

JSP PROCESS?


User Request
Web Server
JSP Container
File Changed
Generate Servlet code
Compile Servlet
Servlet Container
Execute Servlet
Execute Existing Servlet
Request
Response
jspInit()
_jspService()
jspDestroy()

- Has Seven phases.
- Slow in loading first time since it has to be translated into servlet, but becomes subsequently faster..

Phase Name	Description
Page Translation	The page is parsed and a java file containing the corresponding servlet created.
Page Compilation	The java file is compiled.
Load Class	The compiled class is loaded.
Create instance	An instance of the servlet is created.
Call jspInit()	This method is called before any other method to allow initialization.
Call _jspService()	This method is called for each request
Call jspDestroy()	Called when the servlet container decides to take the servlet out of service.

- Has Seven phases. Slow in loading first time since it has to be translated into servlet, but becomes subsequently faster..

```
<H1>Current Date</H1>
<%= new Date()%>
```

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
throws ServletException, IOException {

        request.setContentType ("text/html");
        HttpSession session = request.getSession (true);
        JspWriter out = response.getWriter();
        out.println("<H1>Current Date</H1>");
        out.println (new Date());
        ...
}
```

JSP FEATURES

- Script language declarations, scriptlets, and expressions for including Java fragments that embed dynamic content
- Standard directives guiding the translation of a JSP page to a servlet
- Standard actions in the form of predefined JSP tags
- A portable tag extension mechanism, for building tag libraries—effectively extending the JSP language

- Script language declarations, scriptlets, and expressions for including Java fragments that embed dynamic content
- Standard directives guiding the translation of a JSP page to a servlet
- Standard actions in the form of predefined JSP tags
- A portable tag extension mechanism, for building tag libraries—effectively extending the JSP language

- JSP scripting elements lets you insert Java code into the servlet that will be generated from the JSP page.

- There are three forms:

1. Expression 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, and

3. Declarations of the form <%! code %> that are inserted into the body of the servlet class, outside of any existing methods

- JSP scripting elements lets you insert Java code into the servlet that will be generated from the JSP page.

- There are three forms:

1. Expression 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, and

3. Declarations of the form <%! code %> that are inserted into the body of the servlet class, outside of any existing methods

- A JSP expression is used to insert Java values directly into the output.

- It has the following form:

  <%= Java Expression %>

- The Java expression is
  - evaluated,
  - converted to a string, and
  - inserted into the page
  - Not terminated by " ; "

- This evaluation is performed at runtime (when the page is requested), and thus has full access to information about the request.

- A JSP expression is used to insert Java values directly into the output.

```
<% if (true) { %>
You <B>Are Rocking</B> the game!
<% } else { %>
Better <B>Luck </B>  Next time!
<% } %>
```

```
if (true) {
    out.println (" You <B>Are Rocking</B> the game! ");
}
else {
    out.println (" Better <B>Luck </B>  Next time! ");
}
```

HTML Code in Expression and Scriptlets

- A JSP expression is used to insert Java values directly into the output.

< %= calculate() %>
< % calculate(); % >

```
public void _jspService(HttpServletRequest request,
          HttpServletResponse response)
    throws ServletException, IOException {
        request.setContentType ("text/html");
        HttpSession session = request.getSession(true);
        JspWriter out = response.getWriter();
        out.println(calculate());
        calculate();
    ...
}
```

JSP Declarations

- A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (outside of the service method processing the request)
- It has the following form:

  <%! Variable or method %>
- Declarations do not produce output
-

```
<b>Declaration of Variable</b>
<%! int varCount=1000; %>   <br>
<%= "Value is:"+varCount %>
<br>

<b>Declaration of Method</b> <br>
<%! int add(int firstNum, int secondNum){
return (firstNum+secondNum);
};
%>
<%= "Additional Value is:"+add(10,20) %>
</body>
```

PREDEFINED OR IMPLICIT OBJECTS

- As we have seen before, there are variables that can be used in the code.
- There are eight automatically defined variables, sometimes called implicit objects.
- The available variables are request, response, out, session, application, config, pageContext, and page.

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

request: This is the HttpServletRequest associated with the request

It lets you

- ✓ look at the request parameters (via getParameter),
- ✓ the request type (GET, POST, HEAD, etc.), and
- ✓ the incoming HTTP headers (cookies, etc.)

response:

- ✓ This is the HttpServletResponse associated with the response to the client
- ✓ The output stream is buffered,
- ✓ Thus, it is legal to set HTTP status codes and response headers, even though this is not permitted in regular servlets once any output has been sent to the client

out:

- This is the PrintWriter used to send output to the client.
- However, in order to make the response object useful, this is a buffered version of PrintWriter called JspWriter.
- Note that you can adjust the buffer size, or even turn buffering off, through use of the buffer attribute of the page directive.

session:

- This is the HttpSession object associated with the request
- Sessions are created automatically, so this variable is bound even if there was no incoming session reference (unless session was turned off using the session attribute of the page directive)

application:

- This is the ServletContext as obtained via getServletConfig(). getContext()
- Servlets and JSP pages can hold constant data in the ServletContext object
- Getting and setting attributes is with getAttribute and setAttribute
- The ServletContext is shared by all the servlets in the server

PageContext:

- JSP introduced a new class called PageContext.
- It encapsulate use of server-specific features like higher performance JspWriters.
- The idea is that, if you access the server-specific features through this class rather than directly, your code will still run on "regular" servlet/JSP engines.

Page

- This is simply a synonym for this.
- page is not very useful in Java codes in JSP pages.
- It was created as a placeholder for the time when the scripting language could be something other than Java.

```
<a href="./jsp/pre_implicit.jsp">Implicit objects</a>

Pre_implicit.jsp

<form name="pre_implicitFrm" id="pre_implicitFrm" method="post"
action="./implicitobjects.jsp">
<%
session.setAttribute("userSession", "Mallikarjuna");
pageContext.setAttribute("USER", "Mallik", PageContext.APPLICATION_SCOPE);
pageContext.setAttribute("PASSWORD", "Arjun", PageContext.REQUEST_SCOPE);
%>
<Table>
<caption>LOGIN ::</caption>
<th bgcolor=red colspan=2></th>
<tr><td>Username</td><td><input type="text" name="txtUsername"></td>
</tr>
<tr><td>Password</td><td><input type="text" name="txtPassword"></td>
</tr></Table>
<input type="submit" value="submit">

</form>
</body>
</html>
```

```
<style>
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}

<Table>
<caption>IMPLICIT OBJECTS ::</caption>
<tr><td>out</td><td>JspWriter</td><td><% out.print("Today is:"+java.util.Calendar.getInstance().getTime());
    %></td></tr>
<tr><td>request</td><td>HttpServletRequest</td><td><%= request.getParameter("txtUsername") %></td></tr>
<tr><td>response</td><td>HttpServletResponse</td><td><a href="./implicit_send_redirect.jsp">Test
    SendRedirect</a></td></tr>
<tr><td>session</td><td>HttpSession</td><td><%= (String)session.getAttribute("userSession") %></td></tr>
<tr><td>config</td><td>ServletConfig</td><td><a href="../sampleImplicitObj">Test Config</td></tr>
<tr><td>application</td><td>ServletContext</td><td><%= application.getInitParameter("DBConn") %></td></tr>
<tr><td>pageContext</td><td>pageContext</td><td><%=
    (String)pageContext.getAttribute("USER",PageContext.APPLICATION_SCOPE) %></td></tr>
<tr><td>exception</td><td>Throwable</td><td><a href="./working_implicitobj.jsp">Test Error Page</a></td></tr>
</Table>
</body>
```

implicit_send_redirect.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 <% response.sendRedirect("https://www.google.com"); %>
</body>
</html>
```

web.xml

```
<servlet>
<servlet-name>sampleImplicitObj</servlet-name>
<jsp-file>/jsp/implicit_config.jsp</jsp-file>
<init-param>
<param-name>Driver</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>sampleImplicitObj</servlet-name>
<url-pattern>/sampleImplicitObj</url-pattern>
</servlet-mapping>
```

implicit_config.jsp

```
Implicit Config: <%= config.getInitParameter("Driver") %>
```

working_implicitobj.jsp

```
<form action="./working.jsp">
Number1:<input type="text" name="first" >
Number2:<input type="text" name="second" >
<input type="submit" value="divide">
</form>
```

Working.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  pageEncoding="ISO-8859-1"
    errorPage="./error.jsp"%>

<%
  String num1 = request.getParameter("first");
String num2 = request.getParameter("second");
 // extracting numbers from request
int x = Integer.parseInt(num1);
int y = Integer.parseInt(num2);
int z = x / y; // dividing the numbers
out.print("division of numbers is: " + z); // result
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
  h3>Sorry an exception occured!</h3>   Exception is: <%= exception.getMessage() %>
```

A JSP directive affects the overall structure of the servlet class that is created from the JSP page

It usually has the following form:

```
<%@ directive attribute="value" %>
```

Multiple attribute settings for a single directive can be combined:

```
<%@ directive
attribute1="value1"
attribute2="value2"
...
attributeN="valueN" %>
```

There are three main types of the directive:

page, which lets you do things like:
- import classes
- customize the servlet superclass

include, which lets you:
- insert a file into the servlet class at the time the JSP file is translated into a servlet

taglib directive:
- indicates a library of custom tags that the page can include

isThreadSafe="true|false"
Normal servlet processing or implementing SingleThreadModel
session="true|false"
Allowing/disallowing sessions
buffer="sizekb|none"
specifies the buffer size for the JspWriter out
autoflush="true|false"
Flush buffer when full or throws an exception when buffer isfull
extends="package.class"
info="message"
A message for the getServletInfo method
errorPage="url"
Define a JSP page that handles uncaught exceptions
isErrorPage="true|false"
language="java"

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" errorPage="./error.jsp"%>
```

INCLUDE DIRECTIVE

- This directive lets you include files at the time the JSP page is translated into a servlet.
- Content of the include file is parsed by the JSP at translation time.
- Includes a static file.
- The directive looks like this:
- <%@ include file="relative url" %>

<%@ include file="header.html" %>
<%@ include file="footer.html" %>

- JSP taglib directive is used to define the tag library with "taglib" as the prefix, which we can use in JSP.
- More detail will be covered in JSP Custom Tags section
- JSP taglib directive is used in the JSP pages using the JSP standard tag libraries
- It uses a set of custom tags, identifies the location of the library and provides means of identifying custom tags in JSP page

```
<%@ taglib uri="uri" prefix="value"%>
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
    <%@ taglib prefix="malliktag" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Directive JSP</title>
< malliktag :hello/>
</head>
<body>
</body>
</html>
```

THANK YOU