1.What Is a Servlet?

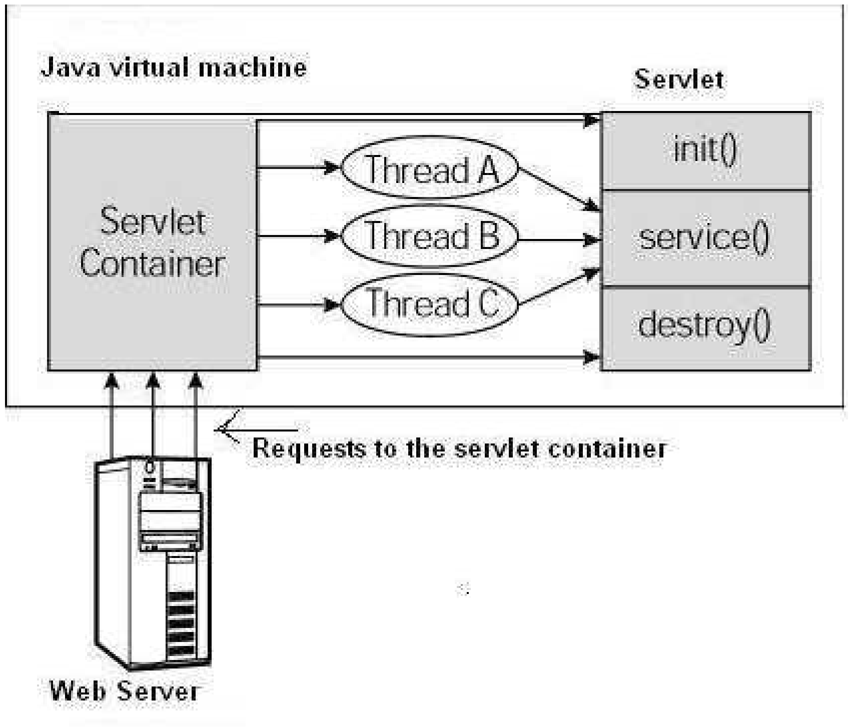
- A servlet is a small Java program that runs within a Web server.
- Servlets receive and respond to requests from Web clients, usually across HTTP, the Hyper Text Transfer Protocol.
- Servlet is an opposite of applet as a server-side applet.
- Applet is an application running on client while servlet is running on server.
- Servlets are server side components that provide a powerful mechanism for developing web applications.
- Using servlets we can create fast and efficient server side applications and can run it on any servlet enabled web server.
- Servlet runs entirely inside the JVM (Java Virtual Machine).
- Since the servlet runs on server side so it does not depend on browser compatibility.

Advantages of using Servlets

- Less response time because each request runs in a separate thread.
- Servlets are scalable.
- Servlets are robust and object oriented.
- Servlets are platform independent.

2. Servlet Architecture:

- The following figure depicts a typical servlet life-cycle scenario.
    - First the HTTP requests coming to the server are delegated to the servlet container.
    - The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the



servlet.

- User sends request for a servlet by clicking a link that has URL to a servlet.
- The container finds the servlet using **deployment descriptor** and creates two objects

  HttpServletRequest

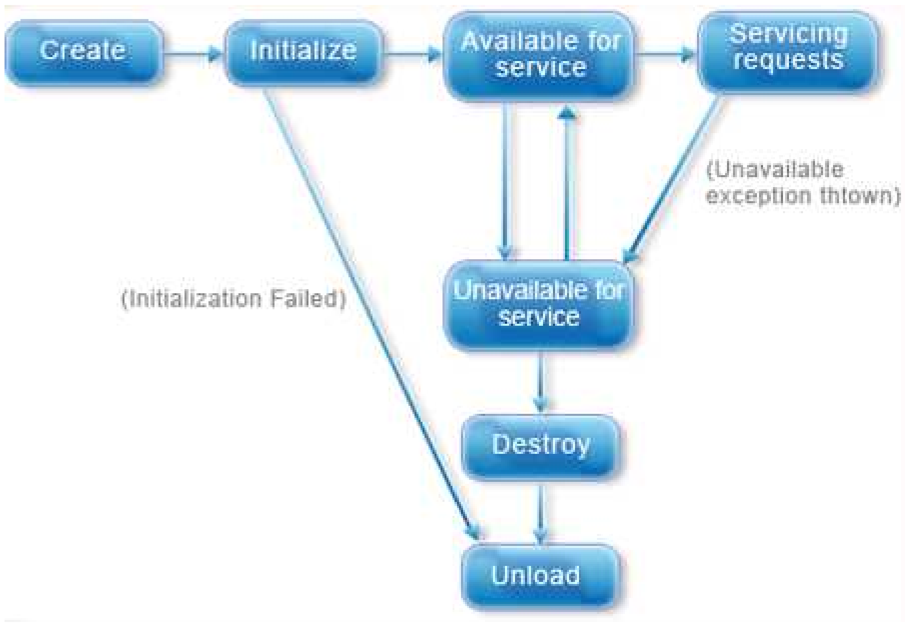
  HttpServletResponse

- Then the container creates or allocates a thread for that request and calls the Servlet's service() method and passes the **request, response** objects as arguments.
- The service() method, then decides which servlet method, doGet() or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.
- Then the Servlet uses response object to write the response back to the client.

- After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.

3.Life cycle of servlet:

- **The init() method**
- **The service() method**
- **The destroy() method**



The init() method :

- The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request.
- So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate.
- The init() method simply creates or loads some data that will be used throughout the life of the servlet.
- The init method definition looks like this:

```
public void init() throws ServletException
{
        // Initialization code...
}
```

The service() method :

- The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- Here is the signature of this method:

  ```
  public void service(ServletRequest request, ServletResponse response)
  throws ServletException, IOException
  {
  }
  ```
- The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.
- The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The destroy() method :

- The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

  ```
  public void destroy()
  ```

```
                {
               // Finalization code...
               }
```

4.General structure of servlet/SKELEton of servlet:

```
imort javax.servlet.*;
class className extends GenericServlet
{
     public void init() throws ServletException
      {
           // Initialization code...
     }
    public void service(ServletRequest request, ServletResponse response)throws
  ServletException, IOException
    {
    }
    public void destroy()
    {
    // Finalization code...
    }
    }
```

5.Simple Servlet program:

```
import java.io.*;
import javax.servlet.*;
public class A extends GenericServlet
{
public void service(ServletRequest req ,ServletResponse res)throws
ServletException,IOException
{
res.setContentType(“text/html”) ;
PrintWriter out=res.getWriter();
out.println(“<p> My First Servlet program </p> “);
}
}
```

Web deployment: (web.xml)

```
<servlet>
<servlet-name>CSA</servlet>
<servlet-class>A.class</sertlet-class>
</servlet>
<servlet-mapping>
<servlet-name>CSA</servlet>
<url-patter>*.dll</url-patter> </servlet-mapping>
```

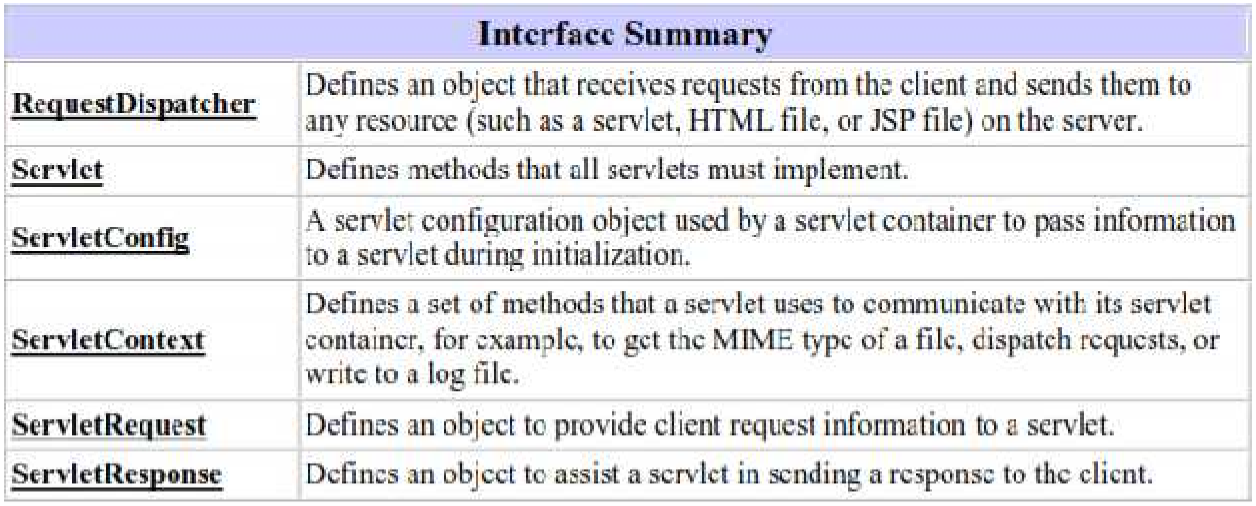
6.servlet API:

javax.servlet - The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

javax.servlet.http-The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container

7.The javax.servlet Package

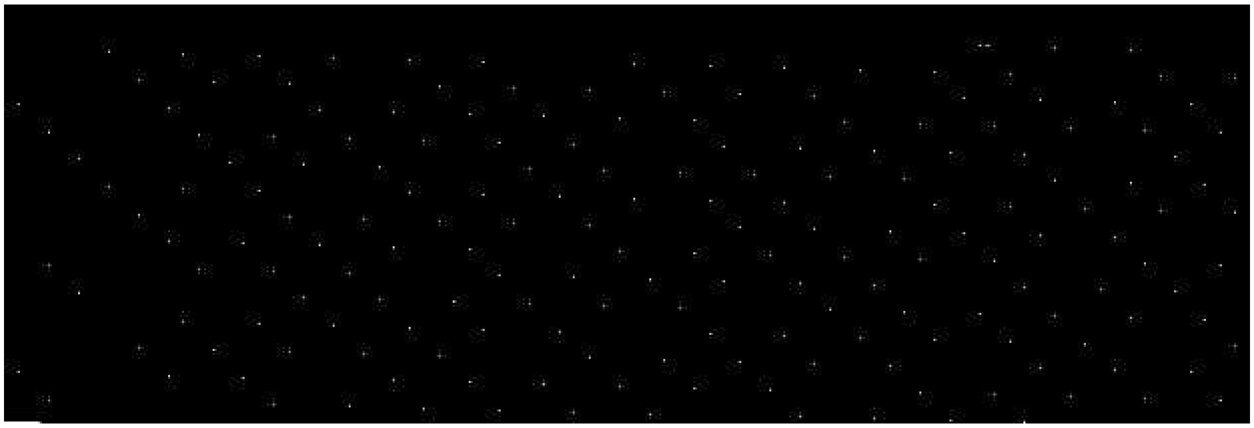
The javax.servlet package contains a number of interfaces and classes that establish the framework in which servlets operate.



Interface Summary	
RequestDispatcher	Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server.
Servlet	Defines methods that all servlets must implement.
ServletConfig	A servlet configuration object used by a servlet container to pass information to a servlet during initialization.
ServletContext	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.
ServletRequest	Defines an object to provide client request information to a servlet.
ServletResponse	Defines an object to assist a servlet in sending a response to the client.

8.The javax.servlet.http Package

The javax.servlet.http package contains a number of interfaces classes that are commonly used by servlet developers.



Following are the class :

class	description
Cookie	Allow state information to be stored on client machine
HttpServlet	Provide Methods to handle Http Request and response
HttpSessionEvent	Encapsulate a session changed event
HttpSessionBindingEvent	Indicate when a listener is bounded to or unbounded from session value

9.Reading Servlet Parameters

- The ServletRequest class includes methods that allow you to read the names and values of parameters that are included in a client request.
- The example contains two files. A Web page is defined in PostParameters.htm and a servlet is defined in PostParametersServlet.java.
- Different mathods to read parameter are as follows:
    - getParameter(string)
    - getParamaterNames();
    - getParamaterValues();

getParameter()-returns a value of parameter in the string form

getParameterNames()-returns an enumeration of the parameter names.These are processed in loop

Program: To display greeting message on the browser Hello UserName How Are You accept username from the client.

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http*;
public class A extends GenericServlet
{
public void service(ServletRequest req ,ServletResponse res)throws
ServletException,IOException
{
res.setContentType(“text/html”) ;
PrintWriter out=res.getWriter();
String msg=req.getParameter("t1");
out.println(“hello”+msg+”how are you”);
}
}
```

HTML code

```
<html>
<body>
<form action=http://localhost:8080/A >
<input type=”text box” name=”t1” value=” “>
<input type=”submit” name=”submit”>
</form>
</body>
</html>
```

<u>using getParameterName() method:</u>

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http*;
public class A extends GenericServlet
{
public void service(ServletRequest req ,ServletResponse res)throws
ServletException,IOException
{
res.setContentType("text/html") ;
PrintWriter out=res.getWriter();

Enumeration e=req.getParameterNames();
while(e.hasMoreElements())
{
Sting a=e.nextElement();
String msg=request.getParameter(a);
out.println(msg);
}
}
```

```
<html>
<body>
<form action=http://localhost:8080/A>
<input type="text box" name="t1" value=" ">
<input type="text box" name="t2" value=" ">
<input type="submit" name="submit">
</form>
</body>
</html>
```

10.Handling Http request and Http response:

- The HttpServlet class provide a specialized methods that handle the various types of HTTP request.
- The different methods are:

doGet(),doPost(),doOperation(),doPut(),doTrace(),doDelete()

HTTP doGet() method:

- The doGet() method is the method inside a servlet that gets called every time a request from a html or jsp page is submitted.
- The control first reaches the doGet() method of the servlet and then the servlet decides what functionality to invoke based on the submit request. The get method called when the type of page submission is "GET".
- doGet is used when there is are requirement of sending data appended to a query string in the URL.
- The doGet models the GET method of Http and it is used to retrieve the info on the client from some server as a request to it.
- The doGet cannot be used to send too much info appended as a query stream. GET puts the form values into the URL string.
- GET is limited to about 256 characters (usually a browser limitation) and creates really ugly URLs.

Program:

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http*;
public class A extends HttpServlet {
public void doGet(HttpServletRequest  request, HttpServletResponse  response)throws
ServletException,IOException
{
res.setContentType(“text/html”) ;
PrintWriter out=res.getWriter();
String msg=req.getParameter("t1");
out.println("hello"+msg+"how are you");
}
}
<html>
<body>
<form action=http://localhost:8080/A method=GET >
<input type="text box" name="t1" value=" ">
<input type="submit" name="submit">
</form>
</body>
</html>
```

HTTP doPost() method:

- The doPost() method is the method inside a servlet that gets called every time a requests from a HTML or jsp page calls the servlet using "POST" method.
- doPost allows you to have extremely dense forms and pass that to the server without clutter or limitation in size. e.g. you obviously can't send a file from the client to the server via doGet.
- doPost has no limit on the amount of data you can send and because the data does not show up on the URL you can send passwords.
- But this does not mean that POST is truly secure. It is more secure in comparison to doGet method.

Program:

```
import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http*;
public class A extends HttpServlet {
public void doPost(HttpServletRequest  request, HttpServletResponse  response)throws
ServletException,IOException
{
```

```
res.setContentType("text/html") ;
PrintWriter out=res.getWriter();
String msg=req.getParameter("t1");
out.println("hello"+msg+"how are you");
}
}

<html>
<body>
<form action=http://localhost:8080/A  method=POST>
<input type="text box" name="t1" value=" ">
<input type="submit" name="submit">
</form>
</body>
</html>
```

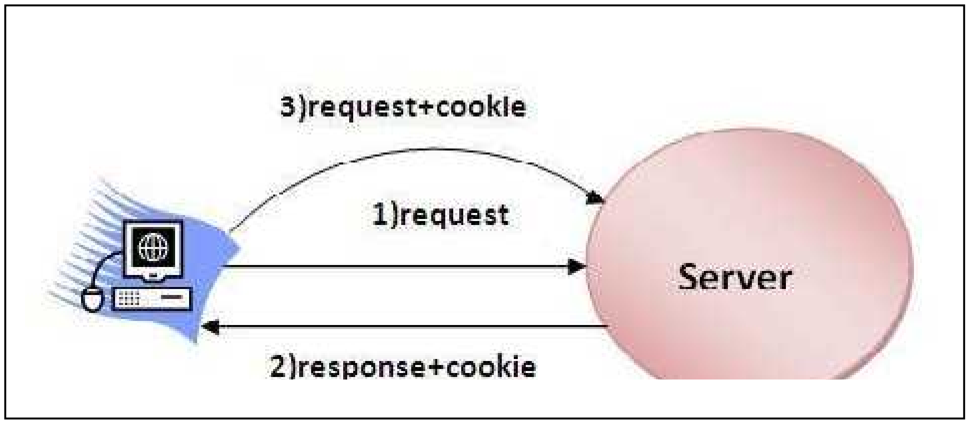
Difference between HTTP doGet and HTTP doPost methods of Servlet

Difference Type	**GET (doGet())**	**POST (doPost())**
HTTP Request	The request contains only the request line and HTTP	Along with request line and header it also contains HTTP body.
URL Pattern	Query string or form data is simply appended to the URL as name-value pairs.	Form name-value pairs are sent in the body of the request, not in the URL itself.
Parameter passing	The form elements are passed to the server by appending at the end of the URL.	The form elements are passed in the body of the HTTP request.
Size	The parameter data is limited (the limit depends on the container normally	Can send huge amount of data to the server.
Idempotency	GET is Idempotent(can be applied multiple times without changing the values	POST is not idempotent(warns if applied multiple times without changing the values

Usage	Generally used to fetch some information from the	Generally used to process the sent data.
Security	Not Safe - A person standing over your shoulder can view your userid/pwd if submitted via Get (Users can see data in address bar.)	Safe - No one will be able to view what data is getting submitted (Data hidden from users.)
Data Format	Supports ASCII.	Supports ASCII + Binary.

11.Using 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.



- Different methods in cookie class are:

1. **String getName()-** Returns a name of cookie
2. **String getValue()-**Returns a value of cookie
3 **.int getMaxAge()-**Returns a maximum age of cookie in millisecond
4. **String getDomain(**)-Returns a domain
5. **boolean getSecure()-**Returns true if cookie is secure otherwise false
6. **String getPath()-**Returns a path of cookie

7.**void setPath(Sting)-** set the path of cookie

8.**void setDomain(String)-**set the domain of cookie

9.**void setMaxAge(int)-**set the maximum age of cookie

10.**void setSecure(Boolean)-**set the secure of cookie.

Creating cookie:

- Cookie are created using cookie class constructor.
- Content of cookies are added the browser using addCookies() method.

Reading cookies:

- Reading the cookie information from the browser using getCookies() method.
- Find the length of cookie class.
- Retrive the information using different method belongs the cookie class.

Program: To create and read the cookie for the given cookie name as "EMPID" and its value as"AN2356".(vtu program)

```
public class A extends GenericServlet
{
public void service(ServletRequest req ,ServletResponse res)throws
ServletException,IOException
{
res.setContentType("text/html") ;
PrintWriter out=res.getWriter();
/* creating cookie object */
    Cookie c=new Cookie("EMPID","AN2356");
res.addCookie(c);//adding cookie in the response
/*reading cookies */
Cookie c[]=req.getCookies();
for(int i=0;i<c.length;i++)
{
String Name=c[i].getName();
String value= c[i].getValue();
out.println("name="+Name);
out.println("Value="+Value);
}
    }  }
```

12.What Is Session Tracking?

- Session tracking is the capability of a server to maintain the current state of a single client's sequential requests.
- Session simply means a particular interval of time.
- Session Tracking is a way to maintain state of a user.
- The HTTP protocol used by Web servers is stateless.
- 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.
- This type of stateless transaction is not a problem unless you need to know the sequence of actions a client has performed while at your site.
- Different methods of HttpSession interface are as follows:

1.**object getAttribute(String)-**Returns the value associated with the name passed as argument.
2.**long getCreationTime()-**Returns the time when session created.
3.**String getID()-**Returns the session ID
4.**long getAccessedTIme()-**returns the time when client last made a request for this session.
5.**void setAttribute(String,object)-**Associates the values passed in the object name passed.

Program:

```
   import javax.servelt.*;
   import java.io.*;
public class A extends GenericServlet
{
public void service(ServletRequest req ,ServletResponse res)throws
ServletException,IOException
{
res.setContentType("text/html") ;
PrintWriter out=res.getWriter();
HttpSession h=req.getSesssion(true);
Date d=(Date) h.getAttribute("Date");
out.println("last date and time"+d);
Date d1=new Date();
d1=h.setAttribute("date",d1);
out.println("current date and time="+d1);
}
}
```

13.Servelt Interface:

methods	**description**
void destroy()	Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.
void **init**(ServletConfig config)	Called by the servlet container to indicate to a servlet that the servlet is being placed into service.
void **service**(ServletRequest req, ServletResponse res)	Called by the servlet container to allow the servlet to respond to a request.
getServletInfo()	Returns information about the servlet, such as author, version, and copyright.
ServletConfig **getServletConfig**()	Returns a ServletConfig object, which contains initialization and startup parameters for this servlet

The GenericServlet Class

- The GenericServlet class provides implementations of the basic life cycle methods for a servlet.
- GenericServlet implements the Servlet and ServletConfig interfaces.
- In addition, a method to append a string to the server log file is available.The signatures of this method are shown here:

void log(String s)
void log(String s, Throwable e)

- Here, s is the string to be appended to the log, and e is an exception that occurred

13.using Tomcat for servlet Development:

certain steps taken to setup the tomcat

1.The examples here is Windows environment. The default location for Tomcat 5.5.17 is

C:\Program Files\Apache Software Foundation\Tomcat 5.5

2.to set the environmental variable JAVA_HOME to the top-level directory in which the Java Software Development Kit is installed.

3.To start Tomcat, select Start Tomcat in the Start | Programs menu, , and the n press Start in the Tomcat Properties dialog. The directory

C:\Program Files\Apache Software Foundation\Tomcat 5.5\common\lib\

Contain servlet.api.jar.

4.. To make this file accessible, update your CLASSPATH environment variable so that it includes

C:\Program Files\Apache Software Foundation\Tomcat5.5\common\lib\servlet.api.jar

5.First. Copy the servlet's class file into the following directory:

C:\Program Files\Apache Software Foundation\Tomcat5.5\webapps\servlets.examples\WEB-INF\classes

6.Next, add the servlet's name and mapping to the web.xml file in the following directory

C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\servlets.examples\WEB-INF

JSP(java Server Page)

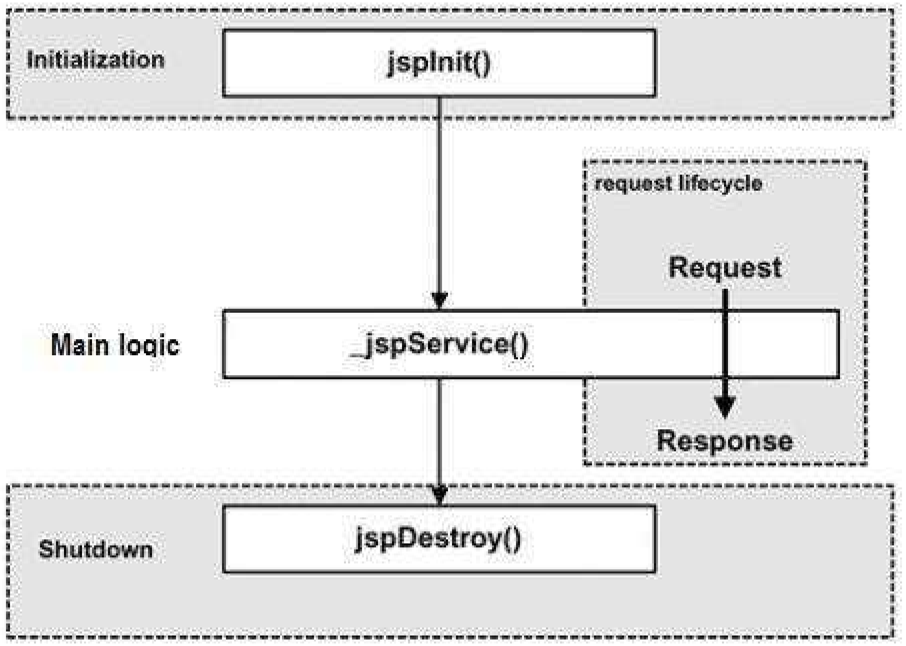
1.What Is JSP?

- Java based technology that simplifies the developing of dynamic web sites.
- JSP pages are HTML pages with embedded code that allows to access data from Java code running on the server.
- JSP provides separation of HTML presentation logic from the application logic.
- JSP technology provides a way to combine the worlds of HTML and Java servlet programming.
- JSP specs are built on the Java Servlet API.
- JSP supports two different styles for adding dynamic content to web pages:
- JSP pages can embed actual programming code (typically Java).
- JSP supports a set of HTML-like tags that interact with Java objects on the server (without the need for raw Java code to appear in the page).

Advantages of JSP

- JSP are translated and compiled into JAVA servlets but are easier to develop than JAVA servlets.
- JSP uses simplified scripting language based syntax for embedding HTML into JSP.
- JSP containers provide easy way for accessing standard objects and actions.
- JSP reaps all the benefits provided by JAVA servlets and web container environment, but they have an added advantage of being simpler and more natural program for web enabling enterprise developer.
- JSP use HTTP as default request / response communication paradigm and thus make JSP ideal as Web Enabling Technology.

2.JSP Life cycle:



Initialization:

- When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit()
{
// Initialization code...
}
```

- Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP service:

- This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

- The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows:

```
void _jspService(HttpServletRequest request, HttpServletResponse response)
{
// Service handling code...
}
```

- The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods ie. GET, POST, DELETE etc.

JSP destroy:

- The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
- The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.
- The jspDestroy() method has the following form:

```
public void jspDestroy()
{
// Your cleanup code goes here.
}
```

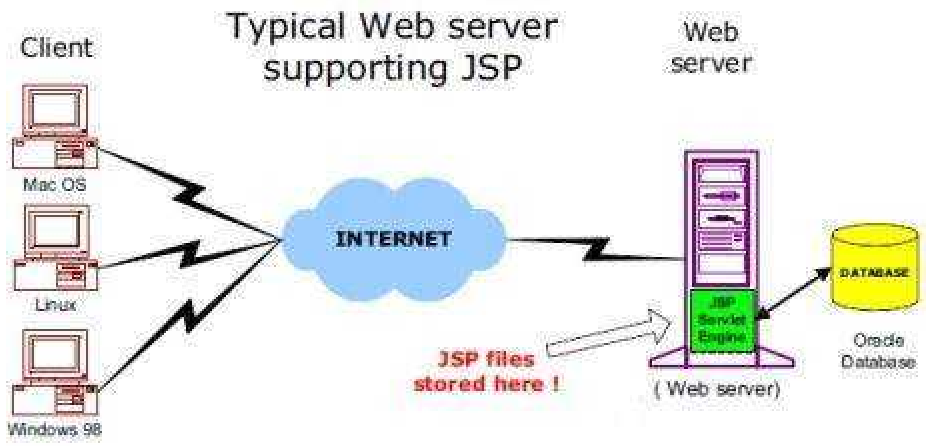
3.JSP Architecture:

The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of .html.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( )

statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.

- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page



4.JSP Tags(VTU question VIMP):

JSP tags define java code that is to be executed before the output of a JSP program is sent to the browser. There are five types of JSP tags:

- Comment Tag
- Declaration statement Tag
- Directive Tag
- Expression Tag
- Scriptlet Tag

Directive Tag:

A Directive tag opens with <%@ and closes with %>. There are three commonly used directives.Used to import java packages into JSP program

Example:

```
<%@ page import = "java.sql.*" %>
```

Comment Tag:

A comment tag opens with <%-- and closes with --%>, and is followed by a comment that usually describes the functionality of statements that follow the comment tag.

Example:

```
<%-- jsp comment tag --%>
```

Declaration Statement Tag:

A Declaration statement tag opens with <%! and is followed by a Java declaration statements that define variables, objects, and methods.

Example:

```
<%!
        int a=10;
        disp()  {   }
%>
```

Expression Tag:

- A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- The expression element can contain any expression that is valid according to the Java anguage Specification but you cannot use a semicolon to end an expression.

Syntax two forms:

<%= expr %>

example:

```
<%!      int a = 5, b = 10;
             <%= a+b %>
%>
```

Scriptlet Tag:

A scriptlet tag opens with <% and contains commonly used java control statements and loops. It closes with %>

Syntax two forms:

```
<%         control statements %>
```

Example:

```
<% for (int i = 0; i < 2; i++) {    %>
                  <p>Hello World!</p>
<%   }   %>
```

Program to display the grading system for the given java subject marks using control statements (VTU question VIMP):

```
<% !
int marks=65;
<% if(marks>=90)%>
<p>grade A</p>
<%else if(marks>=80 && marks<=89)%>
<p>Grade B</p>
<%else if(marks>=70 && marks<=79)%>
<p>Grade C</p>
<%else%>
<p>Fail</p>
%>
```

6. Request String:

- The browser generates a user **request string** whenever the submit button is selected.
- The HttpServletRequest parameter Request object has a request scope that is used to access the HTTP request data, and also provides a context to associate the request-specific data.
- Request object implements **javax.servlet.ServletRequest** interface.
- **Jsp provides the two ways of request string:**

  getParameter(String)

  getParameterNames()

Using request.getParameter()

getParameter() method requires an argument, which is the name of the field whose value you want to retrieve.

Program: Department has set the grade for java subject,accept the input from the user and display the grading on the browser. (VTU question VIMP)

above 90-grade A

80-89 grade B

70-79 grade C

below 70 Fail using jsp

A.html

```
<html>
<body>
<form action=A.jsp>
<input type=”textbox” name=”t1” value=” “>
<input type=”submit” mane=”submit”>
</form>
</body>
</html>
```

A.jsp

```
<%!
        String  t = request.getParameter("t1");
        int Marks=Integer.parseInt(t);
        <% if(marks>=90)%>
        <p>grade A</p>
        <%else if(marks>=80 && marks<=89)%>
        <p>Grade B</p>
        <%else if(marks>=70 && marks<=79)%>
        <p>Grade C</p>
        <%else%>
        <p>Fail</p>
 %>
```

<u>using getParameterNames():</u>

getParameterNames()-returns an enumeration of the parameter names.These are processed in loop

program:

```
<%@ import java.util.*; %>
<%!
Enumeration e=req.getParameterNames();
while(e.hasMoreElements())
{
Sting a=e.nextElement();
String msg=request.getParameter(a);
out.println(msg);
}
%>
```

A.html

```
<html>
<body>
<form action=A.jsp>
<input type=”textbox” name=”t1” value=” “>
<input type=”textbox” name=”t2” value=” “>
<input type=”submit” mane=”submit”>
</form>
</body>
</html>
```

6.cookies:

- A **cookie** is a small piece of information created by a JSP program that is stored in the client’s hard disk by the browser. Cookies are used to store various kind of information such as username, password, and user preferences, etc.
- Different methods in cookie class are:

    1.String getName()- Returns a name of cookie

    2.String getValue()-Returns a value of cookie

    3.**int getMaxAge()**-Returns a maximum age of cookie in millisecond

    **4. String getDomain(**)-Returns a domain

    5.boolean getSecure()-Returns true if cookie is secure otherwise false

    6.String getPath()-Returns a path of cookie

    7.**void setPath(Sting)**- set the path of cookie

    8.**void setDomain(String)-**set the domain of cookie

    9.**void setMaxAge(int)-**set the maximum age of cookie

    10.**void setSecure(Boolean)-**set the secure of cookie.

Creating cookie:

Cookie are created using cookie class constructor.

Content of cookies are added the browser using addCookies() method.

Reading cookies:

Reading the cookie information from the browser using getCookies() method.

Find the length of cookie class.

Retrive the information using different method belongs the cookie class

PROGRAM: To create and read the cookie for the given cookie name as “EMPID” and its value as”AN2356”.(VTU question VIMP)

JSP program to create a cookie

```
<%!

                Cookie c=new Cookie(“EMPID”,”AN2356”);
                response.addCookie(c);

%>
```

<u>JSP program to read a cookie</u>

```
<%!
                Cookie c[]=request.getCookies();
                for(i=0;i<c.length;i++)
                {
                String name=c[i].getName();
                String value=c[i].getValue();
                out.println(“name=”+name);
                out.println(“value=”+value);
                }
%>
```

7.Session object(session tracking or session uses)

- The HttpSession object associated to the request
- Session object has a session scope that is an instance of javax.servlet.http.HttpSession class. Perhaps it is the most commonly used object to manage the state contexts.
- This object persist information across multiple user connection.
- Created automatically by
- Different methods of HttpSession interface are as follows:

1.**object getAttribute(String)-**Returns the value associated with the name passed as argument.
2.**long getCreationTime()-**Returns the time when session created.
3.**String getID()-**Returns the session ID
4.**long getAccessedTIme()-**returns the time when client last made a request for this session.
5.**void setAttribute(String,object)-**Associates the values passed in the object name passed.

Program:

```
<%!
        HttpSession h=req.getSesssion(true);
        Date d=(Date) h.getAttribute("Date");
        out.println("last date and time"+d);
        Date d1=new Date();
        d1=h.setAttribute("date",d1);
        out.println("current date and time="+d1);
%>
```