UNIT III

Web Servers and Servlets

Web Servers:

Web servers are computers that deliver (*serves up*) Web pages. Every Web server has an IP

address and possibly a domain name. For example, if you enter the URL *http://www.mrcet.com/index.html* in your browser, this sends a request to the Web server whose domain name is mrcet*.com*. The server then fetches the page named *index.html* and sends it to your

Any computer can be turned into a Web server by installing server software and connecting the machine to the Internet. There are many Web server software applications, including public domain software and commercial packages.

Install TOMCAT web server and APACHE.
While installation, we assign port number 8080 to APACHE. Make sure that these ports are available i.e., no other process is using this port.

DESCRIPTION:

Set the JAVA_HOME Variable
You must set the JAVA_HOME environment variable to tell Tomcat where to find Java. Failing to properly set this variable prevents Tomcat from handling JSP pages. This variable should list the base JDK installation directory, not the bin subdirectory. On Windows XP, you could also go to the Start menu, select Control Panel, choose System, click on the Advanced tab, press the Environment Variables button at the bottom, and enter the JAVA_HOME variable and value directly as:
Name: JAVA_HOME
Value: C:\jdk

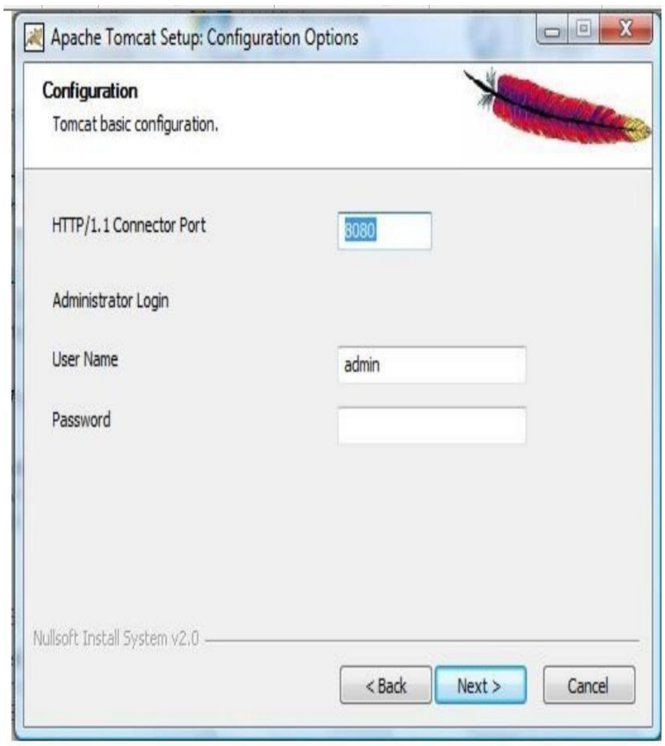
Set the CLASSPATH

Since servlets and JSP are not part of the Java 2 platform, standard edition, you have to identify the servlet classes to the compiler. The server already knows about the servlet classes, but the compiler (i.e., javac) you use for development probably doesn't. So, if you don't set your CLASSPATH, attempts to compile servlets, tag libraries, or other classes that use the servlet and JSP APIs will fail with error messages about unknown classes.

Name: JAVA_HOME
Value: install_dir/common/lib/servlet-api.jar

Turn on Servlet Reloading

The next step is to tell Tomcat to check the modification dates of the class files of requested servlets and reload ones that have changed since they were loaded into the server's memory. This slightly degrades performance in deployment situations, so is turned off by default. However, if you fail to turn it on for your development server, you'll have to restart the server every time you recompile a servlet that has already been loaded into the server's memory.


Apache Tomcat Setup: Configuration Options

Configuration

Tomcat basic configuration.

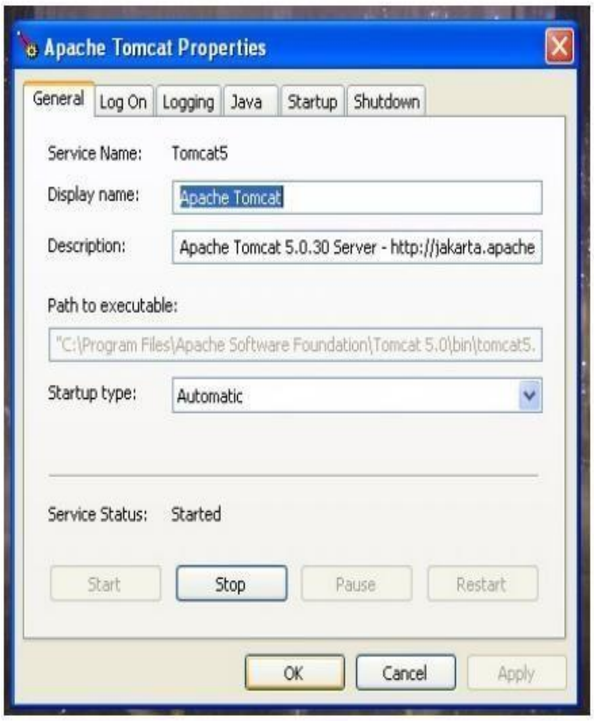
HTTP/1.1 Connector Port: 8080

Administrator Login

User Name: admin

Password:

Nullsoft Install System v2.0


Apache Tomcat Properties

Service Name: Tomcat5

Display name: Apache Tomcat

Description: Apache Tomcat 5.0.30 Server - http://jakarta.apache

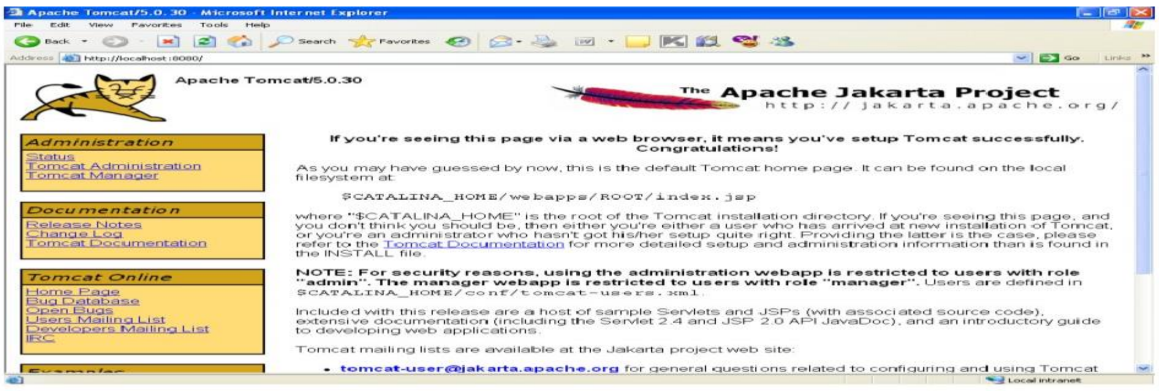
Path to executable: "C:\Program Files\Apache Software Foundation\Tomcat 5.0\bin\tomcat5.

Startup type: Automatic

Service Status: Started

RESULT: Thus TOMCAT web server was installed successfully.

Access the developed static web pages for books web site, using these servers by putting the web pages developed in week-1 and week-2 in the document root.


Apache Tomcat/5.0.30 - Microsoft Internet Explorer

Address: http://localhost:8080/

Apache Tomcat/5.0.30

The Apache Jakarta Project http://jakarta.apache.org/

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.jsp
```

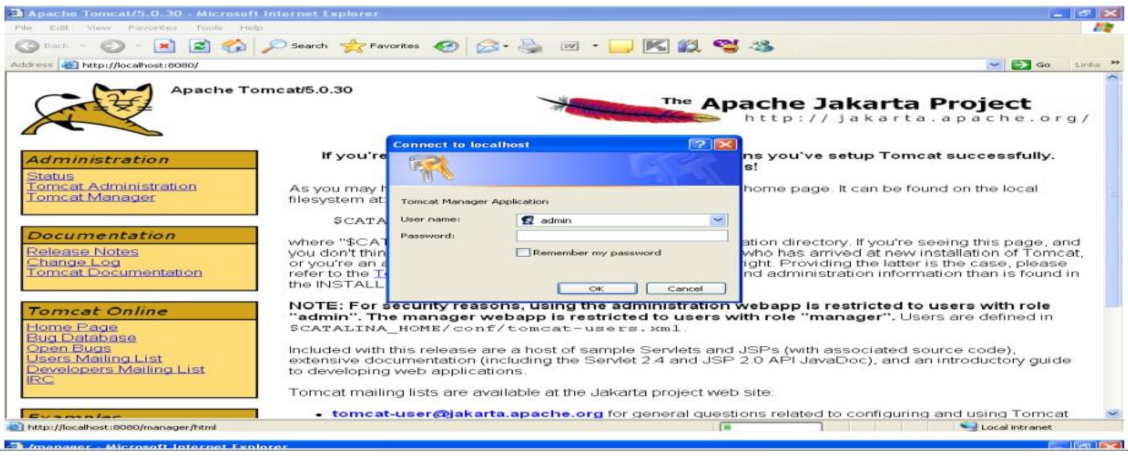
where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation (including the Servlet 2.4 and JSP 2.0 API JavaDoc), and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Jakarta project web site:

- **tomcat-user@jakarta.apache.org** for general questions related to configuring and using Tomcat


Apache Tomcat/5.0.30 - Microsoft Internet Explorer

Address: http://localhost:8080/

Connect to localhost

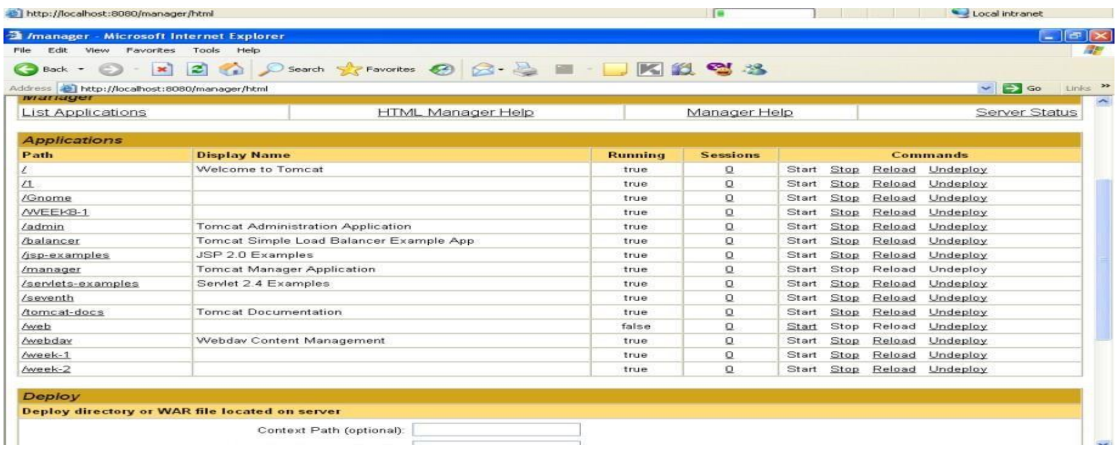
Tomcat Manager Application

User name: admin

Password:

☐ Remember my password

http://localhost:8080/manager/html



http://localhost:8080/manager/html — Local intranet

/manager - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:8080/manager/html

List Applications | HTML Manager Help | Manager Help | Server Status

Applications

Path	Display Name	Running	Sessions	Commands
/	Welcome to Tomcat	true	0	Start Stop Reload Undeploy
/1		true	0	Start Stop Reload Undeploy
/Gnome		true	0	Start Stop Reload Undeploy
/WEEK8-1		true	0	Start Stop Reload Undeploy
/admin	Tomcat Administration Application	true	0	Start Stop Reload Undeploy
/balancer	Tomcat Simple Load Balancer Example App	true	0	Start Stop Reload Undeploy
/jsp-examples	JSP 2.0 Examples	true	0	Start Stop Reload Undeploy
/manager	Tomcat Manager Application	true	0	Start Stop Reload Undeploy
/servlets-examples	Servlet 2.4 Examples	true	0	Start Stop Reload Undeploy
/seventh		true	0	Start Stop Reload Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy
/web		false	0	Start Stop Reload Undeploy
/webdav	Webdav Content Management	true	0	Start Stop Reload Undeploy
/week-1		true	0	Start Stop Reload Undeploy
/week-2		true	0	Start Stop Reload Undeploy

Deploy

Deploy directory or WAR file located on server

Context Path (optional):

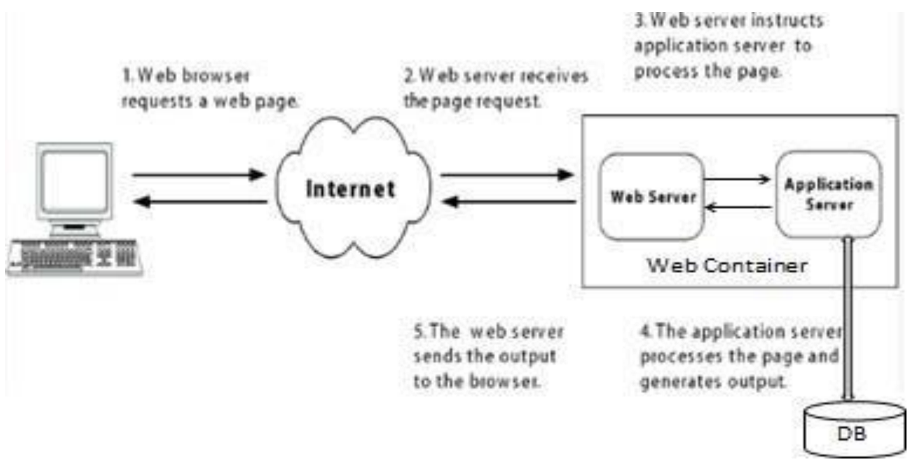
RESULT: These pages are accessed using the TOMCAT web server successfully.

INTRODUCTION TO SERVLETS

Servlets:

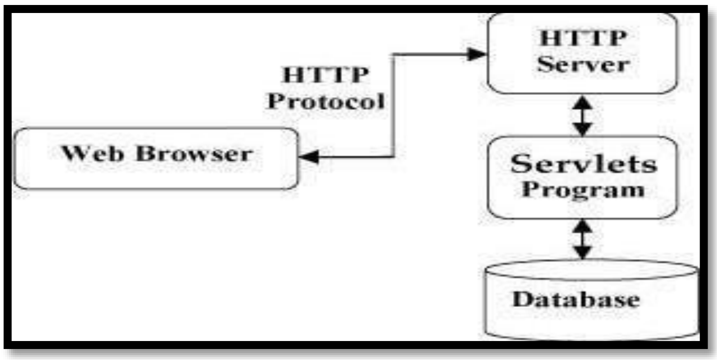
- Servlets are server side programs that run ona Web or Application server and act as a middle layer between a requests coming from a Web browser and databases or applications on the server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Servlets don‘t fork new process for each request, instead a new thread is created.
- Servlets are loaded and ready for each request.
- The same servlet can handle many requests simultaneously.

Web Container: It is web server that supports servlet execution. Individual Servlets are registered with a container. Tomcat is a popular servlet and JSP



1. Web browser requests a web page.
2. Web server receives the page request.
3. Web server instructs application server to process the page.
Internet
Web Server
Application Server
Web Container
4. The application server processes the page and generates output.
5. The web server sends the output to the browser.
DB

Servlet Architecture:

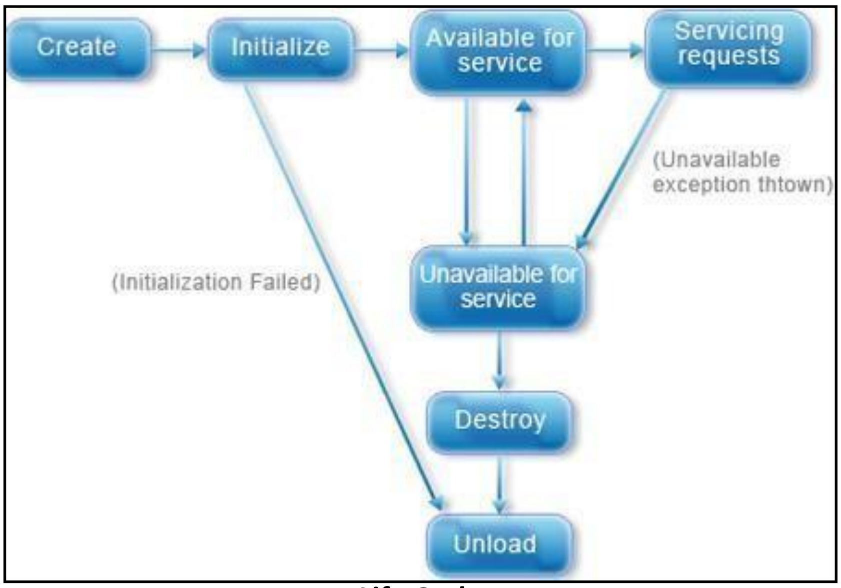


Servlets Tasks:

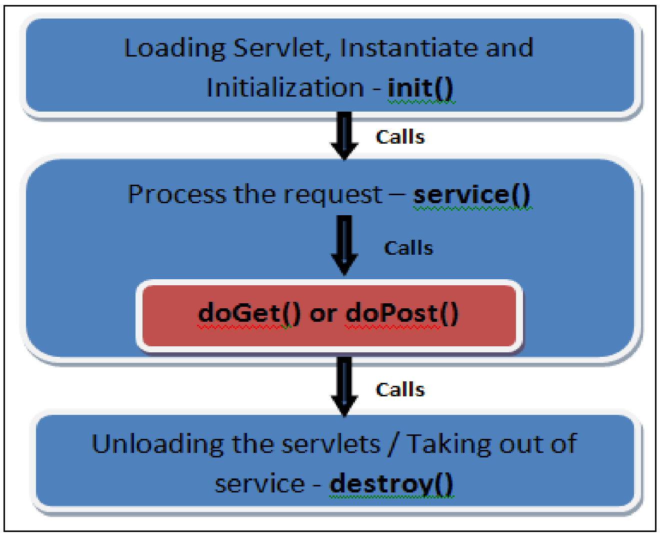
Servlets perform the following major tasks:

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Life Cycle of Servlet



Life Cycle


Loading Servlet, Instantiate and Initialization - init()
Calls
Process the request – service()
Calls
doGet() or doPost()
Calls
Unloading the servlets / Taking out of service - destroy()

Steps:

The sequence in which the Web container calls the life cycle methods of a servlet is:

1. The Web container loads the servlet class and creates one or more instances of the servlet class.
2. The Web container invokes init() method of the servlet instance during initialization of the servlet. The init() method is invoked only once in the servlet life cycle.
3. The Web container invokes theservice() method to allow a servlet to process a client request.
4. The service() method processes the request and returns the response back to the Web container.
5. The servlet then waits to receive and process subsequent requests as explained in steps 3 and 4.
6. The Web container calls the destroy() method before removing the servlet instance from the service. The destroy() method is also invoked only once in a servlet lifecycle.

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.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet.
- 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.

```
publicvoidinit()throwsServletException{
// 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.

```
publicvoid service(ServletRequest request,
ServletResponse response)
throwsServletException,IOException{
}
```

The doGet() Method

- The doGet() method processes client request, which is sent by the client, using the HTTP GET method.
- To handle client requests that are received using GET method, we need to override the doGet() method in the servlet class.
- In the doGet() method, we can retrieve the client information of the HttpServletRequest object. We can use the HttpServletResponse object to send the response back to the client.

```
publicvoiddoGet(HttpServletRequest request,
HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

The doPost() Method:

- The doPost() method handles requests in a servlet, which is sent by the client, using the HTTP POST method.
- For example, if a client is entering registration data in an HTML form, the data can be sent using the POST method.
- Unlike the GET method, the POST request sends the data as part of the HTTP request body. As a result, the data sent does not appear as a part of URL.
- To handle requests in a servlet that is sent using the POST method, we need to override the doPost() method. In the doPost() method, we can process the request and send the response back to the client.

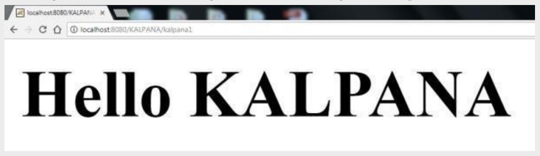
```
publicvoiddoPost(HttpServletRequest request,
HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

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.
-

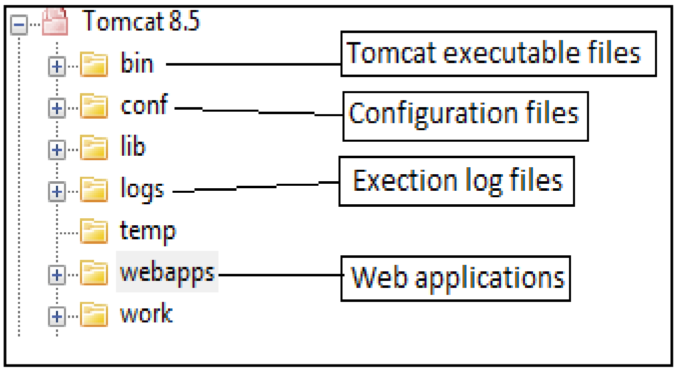
```
publicvoid destroy()
{
  // Finalization code...
  }
```

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

import javax.servlet.http.*;



public class HelloWorld extends HttpServlet {
private String message;

public void init() throws ServletException    {

      // Do required initialization
message = "Hello KALPANA";

  }

public void doGet(HttpServletRequestrequest,HttpServletResponse response)
throwsServletException, IOException {

      // Set response content type

response.setContentType("text/html");
```


Hello KALPANA

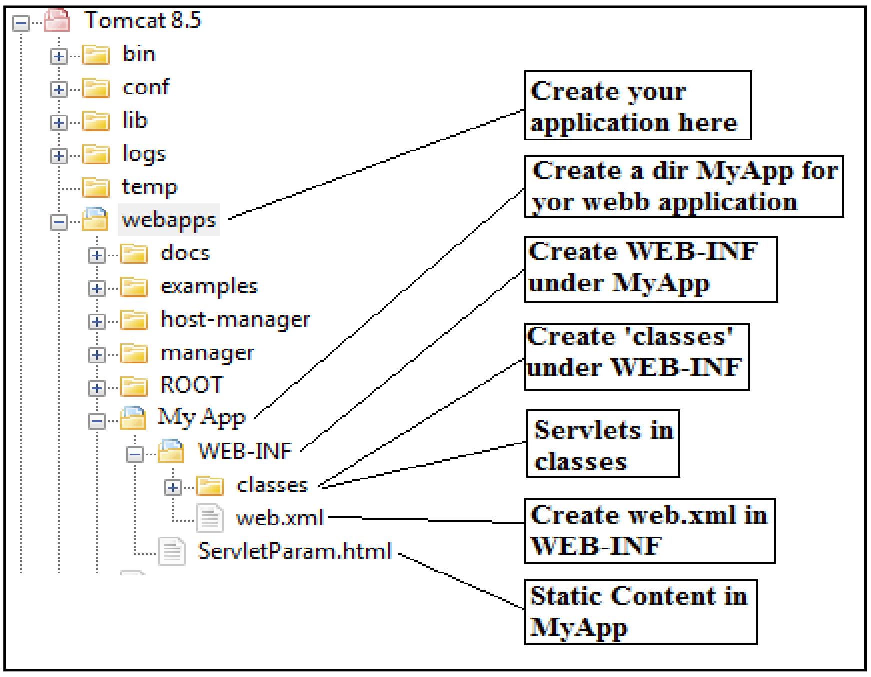
Deploying a Servlet- Steps:

1. Download and install the Java Software Development kit(SDK).
2. Download a server(Tomcat).
3. Configure the server
   - After installation Tomcat folder will contain –Start Tomcat‖ and –Stop Tomcat‖ shortcuts.
   - The JAVA_HOME environment variable should be set so that Tomcat can find JDK JAVA_HOME = c:\jdk1.5
4. Setup deployment environment



Tomcat Directory Structure

- To set up a new application, add a directory under the **webapps**directory and create a subdirectory called **WEB-INF.**
- WEB-INF needs to contain **web.xml (**servlet configuration file**)**
- After WEB-INF dir is created, create a subdirectory **classes**under it.Java classes will go under this directory.



5. Creating ServletDemoServlet

There are three different ways to create a servlet.

   a. By implementing **Servlet** interface
   b. By extending **GenericServlet** class
   c. By extending **HttpServlet** class

6. Compile Servlet and save the class file in **classes** folder.

7. Create a Deployment Descriptor

- The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked.
- The web container uses the Parser to get the information from the web.xml file.
- Add a servlet entry and a servlet- mapping entry for each servlet for Tomcat to run. Add entries after <web-app> tag inside web.xml

```
<web-app>
        <servlet>

        <servlet-name>Demo</servlet-name>

        <servlet-class>DemoServlet</servlet-class>

        </servlet>
        <servlet-mapping>

        <servlet-name>Demo</servlet-name>

        <url-pattern>/welcome</url-pattern>

        </servlet-mapping>
</web-app>
```

8. Start Tomcat server

9. Open browser and type http://localhost/MyApp/DemoServlet

Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

1. **javax.servlet**
2. **javax.servlet.http**

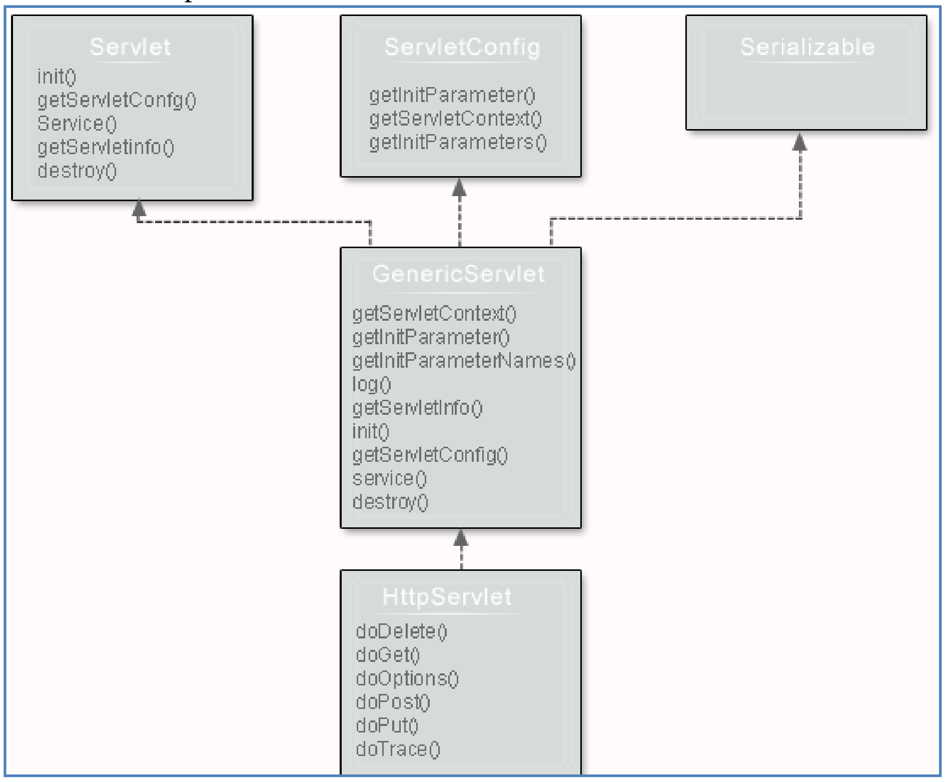
1. javax.servlet

Interfaces

1. Servlet – Declares life cycle methods for a servlet.
2. ServletConfig – To get initialization parameters
3. ServletContext- To log events and accessinformation
4. ServletRequest- To read data from a client request
5. ServletResponse – To write data from client response

Classes

1. GenericServlet – Implements Servlet and ServletConfig
2. ServletInputStream – Provides an input stream for reading clientrequests.
3. ServletOutputStream - Provides an output stream forwriting responses to a client.
4. ServletException – Indicates servlet error occurred.
5. UnavailableException - Indicates servlet is unavailable

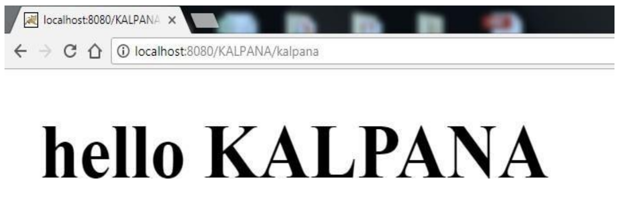


Servlet Interface

- 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.

Method	Description
public void init(ServletConfigconfig)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequestrequest,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfiggetServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

```
import java.io.*;
import javax.servlet.*;
public class First implements Servlet{
        ServletConfig config=null;
        public void init(ServletConfig config){
                this.config=config;
                System.out.println("servlet is initialized");
        }
        public void service(ServletRequest req,ServletResponse res)
                throws IOException,ServletException{
                res.setContentType("text/html");
                PrintWriter out=res.getWriter();
                out.print("<html><body>");
                out.print("<b>hello KALPANA</b>");
                out.print("</body></html>");
        }
        public void destroy(){
                System.out.println("servlet is destroyed");
        }
        public ServletConfig getServletConfig(){
                return config;
        }
        public String getServletInfo(){
                return "copyright 2007-1010";
        }
}
```


localhost:8080/KALPANA/kalpana

hello KALPANA

ServletConfig interface

- When the **Web Container** initializes a servlet, it creates a **ServletConfig** object for the servlet.
- ServletConfig object is used to pass information to a servlet during initialization by getting configuration information from **we b.xml**(Deployment Descriptor).

Methods

- getInitParameter(String name): returns a String value initialized parameter
- getInitParameterNames(): returns the names of the servlet's initialization parameters as an Enumeration of String objects
- getServletContext(): returns a reference to the ServletContext
- getServletName(): returns the name of the servlet instance

ServletContext Interface

- For every **Web application** a **ServletContext** object is created by the web container.
- ServletContext object is used to get configuration information from **Deployment Descriptor**(web.xml) which will be available to any servlet.

Methods :

- getAttribute(String name) - returns the container attribute with the given name
- getInitParameter(String name) - returns parameter value for the specified parameter name
- getInitParameterNames() - returns the names of the context's initialization parameters as an Enumeration ofString objects
- setAttribute(String name,Objectobj) - set an object with the given attribute name in the application scope
- removeAttribute(String name) - removes the attribute with the specified name fromthe application context

Servlet RequestInterface

- True job of a Servlet is to handle client request.
- Servlet API provides two important interfaces **javax.servlet.ServletRequest** to encapsulate client request.
- Implementation of these interfaces provides important information about client request to a servlet.

Methods

- getAttribute(String name), removeAttribute(String name), setAttribute(String name, Object o), getAttributeName() – used to store and retrieve an attribute fromrequest.
- getParameter(String name) - returns value of parameter by name
- getParameterNames() - returns an enumeration of all parameter names
- getParameterValues(String name) - returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist

Servlet ResponseInterface

- Servlet API provides ServletResponseto assist in sending response to client.

Methods
- getWriter()- returns a PrintWriter object that can send character text to the client.
- setContentType(String type)- sets the content type of the response being sent to the client before sending the respond.

<u>GenericServlet</u> class

- 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.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods

- **public void init(ServletConfigconfig)** is used to initialize the servlet.
- **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.
- **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
- **publicServletConfiggetServletConfig()** returns the object of ServletConfig.
- **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
- **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
- **publicServletContextgetServletContext()** returns the object of ServletContext.
- **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
- **public Enumeration getInitParameterNames()** returns all the parameters defined in the **we b.xml** file.
- **public String getServletName()** returns the name of the servlet object.
- **public void log(String msg)** writes the given message in the servlet log file.
- **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

<u>ServletInputStream Class</u>

- It provides stream to read binary data such as image etc. from the request object. It is an abstract class.
- The **getInputStream()** method of **ServletRequest** interface returns the instance of ServletInputStream class
- **intreadLine(byte[] b, int off, intlen)** it reads the input stream.

ServletOutputStream Class

- It provides a stream to write binary data into the response. It is an abstract class.
- The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class.
- ServletOutputStream class provides print() and println() methods that are overloaded.

ServletException and UnavailableException

- ServletException is a general exception that the servlet container will catch and log. The cause can be anything.

- The exception contains a root cause exception.
- Defines an exception that a servlet or filter throws to indicate that it is permanently or temporarily unavailable.
- When a servlet or filter is permanently unavailable, something is wrong with it, and it cannot handle requests until some action is taken. For example, a servlet might be configured incorrectly, or a filter's state may be corrupted.

2. javax.servlet.http

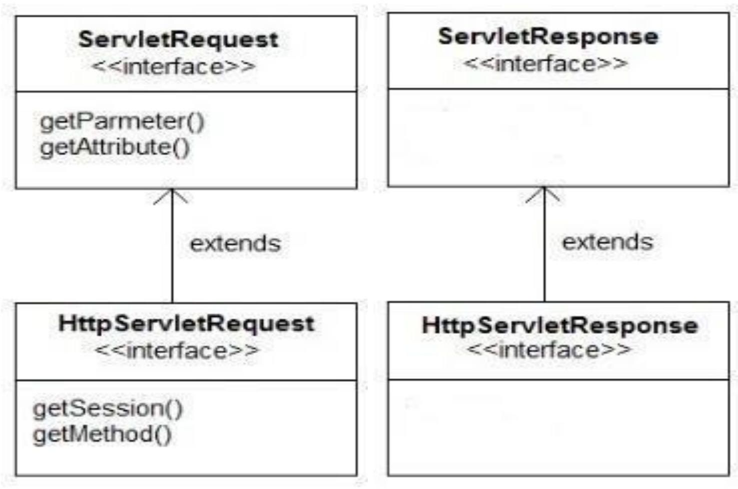
Interfaces

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession

Classes

1. HttpServlet
2. Cookie

HTTPServletRequest and HTTPServletResponse



- ***HTTPServletRequest***Extends the ServletRequest interface to provide request information for HTTP servlets.
- The servlet container creates an HttpServletRequest object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).
- It Contains all the client's request information.
- The HttpServletRequest breaks a request down into parsed elements, such as request URI, query arguments and headers. Various get methods allow you to access different parts of the request.

1. **requestURI** – URL sent bybrowser

2. **Parameters** -

   The `HttpServletRequest` provides methods for accessing parameters of a request. The methods `getParameter()`, `getParameterValues()`and `getParameterNames()` are offered as ways to access the arguments.

3. **Attributes** –
   The request object defines a method called `getAttribute()`. The servlet interface provides this as a way to include extra information about the request that is not covered by any of the other `HttpServletRequest` methods.

4. **ServletInputStream** –
   The `ServletInputStream` is an `InputStream` that allows your servlets to read all of the request's input following the headers.

- ***HTTPServletResponse***Extends the ServletResponse interface and can perform thesetasks

1. **Set Response Codes** –
   The response code for a request is a numeric value that represents the status of the response. For example, 200 represents a successful response, 404 represents a file not found.

2. **Set Headers** –
   Headers for the response can be set by calling setHeader, specifying the name and value of the header to be set.

3. **Send Redirects** –
   The sendRedirect method is used to issue a redirect to the browser, causing the browser to issue a request to the specified URL. The URL passed to sendRedirect must be an absolute URL—it must include protocol, machine, full path, and so on.

4. **Set ServletOutputStream** –
   The ServletOutputStream is obtained by calling getOutputStream on the HttpServletResponse. It is a subclass of OutputStream that contains a number of convenient print and println methods.Data written to the ServletOutputStream goes straight back to the browser.

HTTPSession

- **HttpSession** object is used to store entire session with a specific client.
- We can store, retrieve and remove attribute from **HttpSession** object.
- Any servlet can have access to **HttpSession** object throughout the getSession() method of the **HttpServletRequest** object.

HTTPServlet

- HttpServlet is extends from GenericServlet and does not override init, destroy and other methods.
- It implements the service () method which is abstract method in GenericServlet.
- A subclass of HttpServlet must override at least one method, usually one of these:
  - doGet(), if the servlet supports HTTP GETrequests
  - doPost(), for HTTP POST requests
  - doPut(), for HTTP PUT requests
  - doDelete(), for HTTP DELETE requests
  - Init() and destroy(), to manage resources that are held for the life of the servlet
  - getServletInfo(), which the servlet uses to provide information about itself

Cookie

- A **cookie** is a small piece of information that is persisted between the multiple client requests.
- **javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.
- **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
- **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

Reading Servlet Parameters(or) Handling HTTPRequest and HTTPResponse

- The parameters are the way in which a client or user can send information to the Http Server.
- The **HTTPServletRequest** interface includes methods that allow you to read the names and values of parameters that are included in a client request.
- The **HttpServletResponse** Interface provides functionality for sending response to client.
- The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET method:

- The GET method sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the ? character as follows:

```
http://www.test.com/hello?key1=value1&key2=value2
```

- The GET method is the defualt method to pass information from browser to web server.
- Never use the GET method if you have password or other sensitive information to pass to the server.
- The GET method has size limtation: only 1024 characters can be in a request string.
- This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable.
- Servlet handles this type of requests using **doGet()** method.

POST method:

- A generally more reliable method of passing information to a backend program is the POST method.
- This message comes to the backend program in the form of the standard input which you can parse and use for yourprocessing.
- Servlet handles this type of requests using **doPost()** method.

Reading Form Data using Servlet:

Servlets handles form data parsing automatically using the following methods depending on the situation:

- **getParameter():** You call request.getParameter() method to get the value of a form parameter.

- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

Sending Data to Client:
Obtain a PrintWriter object HTTPServletResponse that can send character text to the client.
```
PrintWriterpw = response.getWriter();
pw.println(–Hello world‖);
```

POST method example
Let us consider **HelloForm.java**

```
import java.io.*;
import java.util.*;
import javax.servlet.http.*;
public class HelloForm extends HTTPServlet {
    public void doPost(HttpServletRequest request,HttpServletResponse response)throws
    IOException,ServletException{
PrintWriter pw = response.getWriter();
pw.print("<html><body>");
pw.print("Name: "+request.getParameter("first_name")+
                        – –+request.getParameter("last_name"));
pw.print("</body></html>");
pw.close();
}
}
```

- Compile HelloForm.java as follows: $javac HelloForm.java
- Compilation would produce **HelloForm.class** file.
- Next you would have to copy this class file in
  <Tomcat- installation-directory>/webapps/ROOT/WEB-INF/classes
- Create following entries in **we b.xml** file located in
  <Tomcat- installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloForm</servlet-name>
<servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>HelloForm</servlet-name>
<url-pattern>/HelloForm</url-pattern>
</servlet- mapping>
```

- Now create a HTML page Hello.htmland put it in

<Tomcat- installation-directory>/webapps/ROOT directory
```
html>
<body>
<formaction="HelloForm"method="GET">
    First Name: <inputtype="text"name="first_name"><br/>
    Last Name: <inputtype="text"name="last_name"/></br>
<inputtype="submit"value="Submit"/>
</form>
</body>
</html>
```

- When you access ***http://localhost:8080/Hello.html***, then output of the aboveform.

First Name: Kalpana

Last Name: Mrcet

Submit

- Start Tomcat Server and open browser.
- Now enter firstname and lastname, Click Submit
- It will generate result

Name: KalpanaMrcet

Reading Initialization Parameters

1. Using Servlet Config:

- An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xmlfile.
- If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Methods

- getInitParameter(String name): returns a String value initialized parameter
- getInitParameterNames(): returns the names of the servlet's initialization parameters as an Enumeration of String objects
- getServletContext(): returns a reference to the ServletContext
- getServletName(): returns the name of the servlet instance

Syntax to provide the initialization parameter for a servlet

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
<web-app>
    <servlet>
     ......
     <init-param>
       <param-name>email</param-name>
       <param-value>kalpana@gamil.com</param-value>
     </init-param>
     ......
    </servlet>
</web-app>
```

Retrieve ServletConfig

```
ServletConfigsc = getServletConfig();
out.println(sc.getInitParameter("email"));
```

Ex: we b.xml

```
<web-app>
        servlet>
        <servlet-name>TestInitParam</servlet-name>
        <servlet-class>TestInitParam</servlet-class>
        <init-param>
        <param-name>email</param-name>
        <param-value>kalpana@gmail.com</param-value>
        </init-param>
        </servlet>
        <servlet-mapping>
        <servlet-name>TestInitParam</servlet-name>
        <url-pattern>/TestInitParam</url-pattern>
        </servlet- mapping>
</web-app>
```

TestInitParam.java

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

import javax.servlet.http.*;
public class TestInitParam extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
   response.setContentType("text/html;charset=UTF-8");
   PrintWriter out = response.getWriter();
   ServletConfigsc=getServletConfig();
                          out.print("<html><body>");
                          out.print("<b>"+sc.getInitParameter("email")+"</b>");
                          out.print("</body></html>");
                          out.close();
   }
}
```

- It will generate result

```
kalpana@gmail.com
```

2. **Using ServletContext**

- An object of ServletContext is created by the web container at time of deploying the project.
- This object can be used to get configuration information from web.xml file.
- There is only one ServletContext object per web application.
- If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

Advantage

- **Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet.
- We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Uses

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

Methods:

- getAttribute(String name) - returns the container attribute with the given name
- getInitParameter(String name) - returns parameter value for the specified parameter name
- getInitParameterNames() - returns the names of the context's initialization parameters as an Enumeration of String objects
- setAttribute(String name,Objectobj) - set an object with the given attribute name in the application scope
- removeAttribute(String name) - removes the attribute with the specified name from the application context

Retrieve ServletContext

```
ServletContextapp = getServletContext();
                              OR

ServletContextapp = getServletConfig().getServletContext();
```

Ex: we b.xml

```
  <web-app>
      <context-param>
      <param-name>driverName</param-name>
      <param-value>sun.jdbc.JdbcOdbcDriver</param-value>
      </context-param>


  <servlet>
  <servlet-name>TestServletContext</servlet- name>
<servlet-class>TestServletContext</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>TestServletContext</servlet- name>
  <url-pattern>/TestServletContext</url-pattern>
</servlet- mapping>
</web-app>
```

TestServletContext.java

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

import javax.servlet.http.*;

public class TestServletContext extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
ServletContextsc = getServletContext();
out.println(sc.getInitParameter("driverName"));
   }
}
```

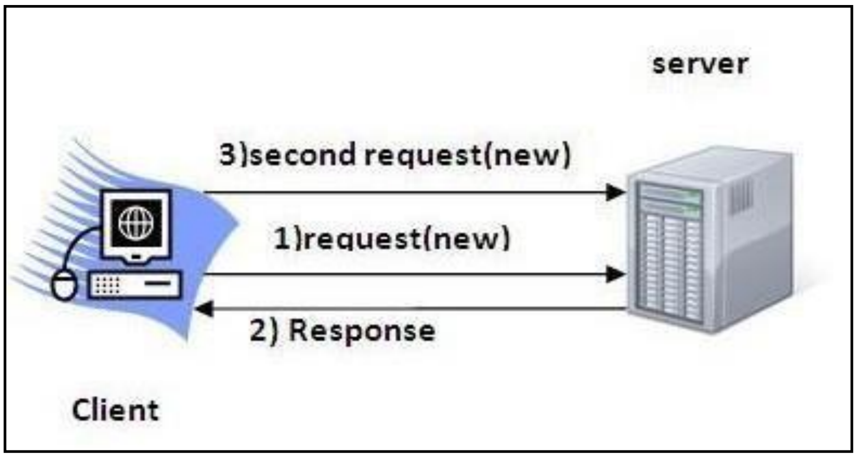
- It will generate result

```
sun.jdbc.JdbcOdbcDriver
```

Context Init parameters	Servlet Init parameter
Available to all servlets and JSPs that are part of web	Available to only servlet for which the <init-param> was configured
Context Init parameters are initialized within the <web-app> not within a specific <servlet> elements	Initialized within the <servlet> for each specific servlet.
ServletContext object is used to get Context Init parameters	ServletConfig object is used to get Servlet Init parameters
Only one ServletContext object for entire web app	Each servlet has its own ServletConfig object

Session Tracking

- Session simply means a particular interval of time.
- Session Tracking is a way to maintain state (data) of an user.
- Http protocol is a stateless, each request is considered as the new request, 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 an user to recognize to particular user.



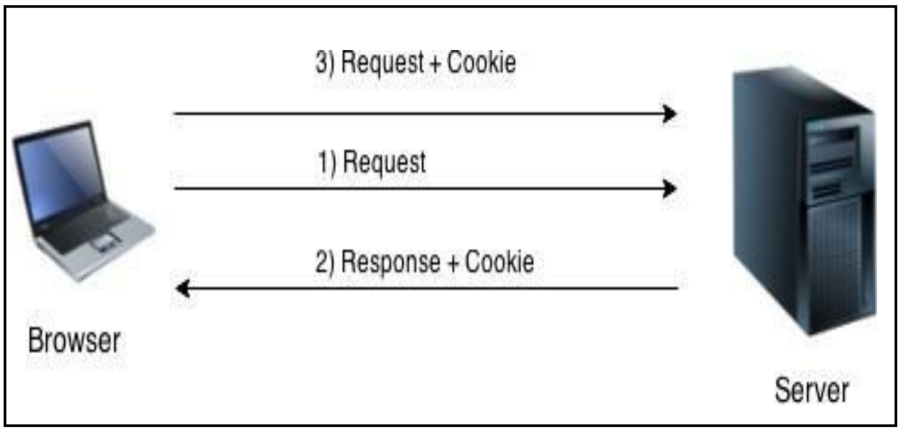
- We use session tracking to recognize the user It is used to recognize the particular user.
- Session Tracking Techniques
  - Cookies
  - Hidden Form Field
  - URL Rewriting
  - HttpSession

Cookies

- Cookies are text files stored on the client computer and they are kept for various information tracking purpose
- There are three steps involved in identifying returning users:
  - Server script sends a set of cookies to the browser inresponse header.
  - Browser stores this information on local machine for future use.
  - When next time browser sends any request to web server then it sends those cookies information to the server in request header and server uses that information to identify the user.
- Cookies are created using **Cookie** class present in Servlet API.
- For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

  a. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.

  b. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.



Disadvantage of Cookies

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

Methods

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.

Create Cookie

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

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
```

Get 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
```

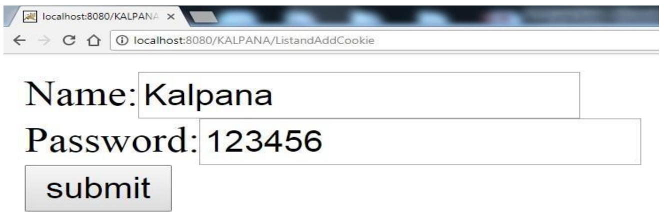
Sending the Cookie into the HTTP response headers:

We use **response.addCookie** to add cookies in the HTTP response header as follows:

```
response.addCookie(cookie);
```

Ex: List and AddCookie.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ListandAddCookieextends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Cookie cookie = null;
        out.println("<html><body>"+
        "<form method='get' action='/mrcet/CookieLab'>"+
        "Name:<input type='text' name='user' /><br/>"+
                "Password:<input type='text' name='pass' ><br/>"+
                "<input type='submit' value='submit'>"+
                "</form>");

                String name = request.getParameter("user");
                String pass = request.getParameter("pass");

        if(!pass.equals("") || !name.equals(""))        {
                Cookie ck = new Cookie(name,pass);
                        response.addCookie(ck);
        }

                Cookie[] cookies = request.getCookies();
                if( cookies != null ){
                        out.println("<h2> Found Cookies Name and Value</h2>");
                        for (inti = 0; i<cookies.length; i++){
                        cookie = cookies[i];
                                out.print("Cookie Name : " + cookie.getName() + ", ");
                        out.print("Cookie Value: " + cookie.getValue()+" <br/>");
                        }
                }
                out.println("</body></html>");
    }
}
```

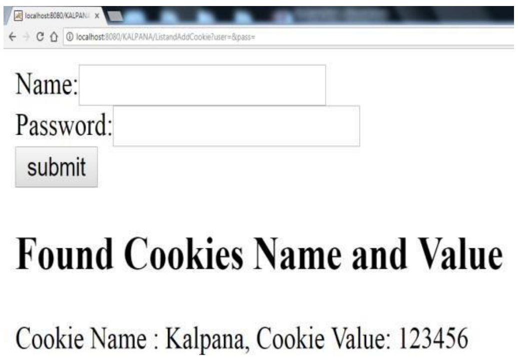


localhost:8080/KALPANA/ListandAddCookie

Name: Kalpana
Password: 123456
submit

we b.xml

```
<web-app>
        <servlet>
        <servlet-name>ListandAddCookie</servlet-name>
        <servlet-class>ListandAddCookie</servlet-class>
        </servlet>
        <servlet-mapping>
        <servlet-name>ListandAddCookie</servlet-name>
        <url-pattern>/ListandAddCookie</url-pattern>
        </servlet- mapping>
</web-app>
```



localhost:8080/KALPANA/ListandAddCookie?user=&pass=

Name:
Password:
submit

Found Cookies Name and Value

Cookie Name : Kalpana, Cookie Value: 123456

Session

- HttpSession Interface provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
- Web container creates a session id for each user. The container uses this id to identify the particular user.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- The session persists for a specified time period, across more than one connection or page request from the user.

Get the HttpSession object

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

1. **publicHttpSessiongetSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **publicHttpSessiongetSession(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.

Destroy Session

session.**invalidate**();

Set/Get data in session

session**.setAttribute(name,value);**

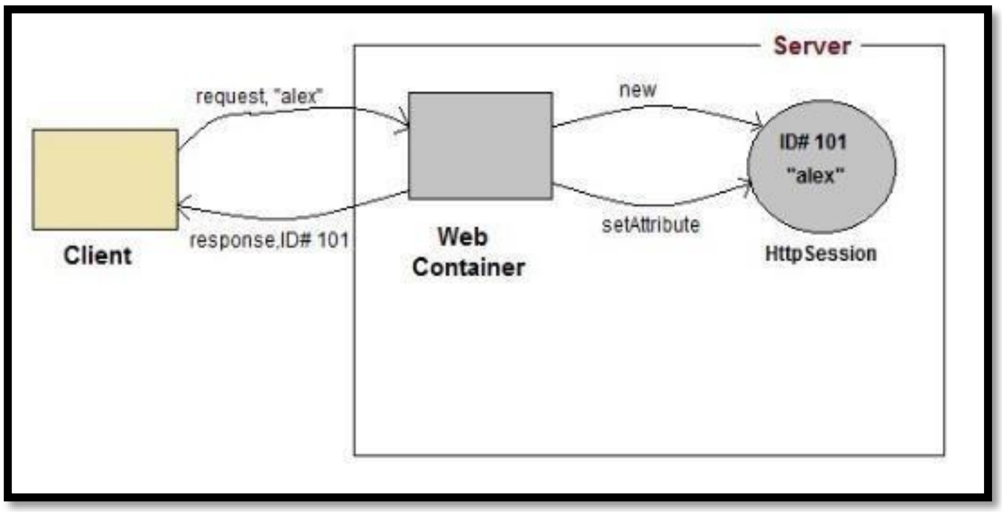
session**.getAttribute(name);**

Methods

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session wascreated, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent arequest 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 toit.

Steps

- On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by webcontainer.
- The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
- The **Web Container** uses this ID, finds the matching session with the ID andassociates the session with the request.



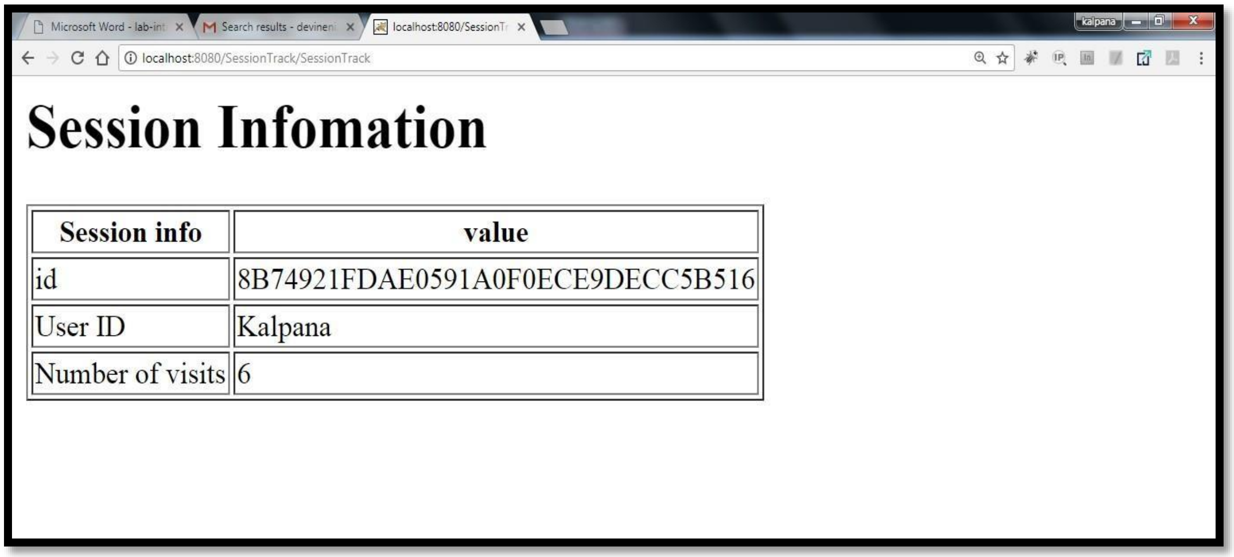
Ex: SessionTrack.java

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

public class SessionTrack extends HttpServlet {

public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    // Create a session object if it is already not created.
HttpSession session = request.getSession(true);

    String title = "Welcome to my website";
String userID = "";
    Integer visitCount = new Integer(0);

if (session.isNew())
{
userID = "Kalpana";
session.setAttribute("UserId", "Kalpana");
    }
else {
visitCount = (Integer)session.getAttribute("visitCount");
visitCount = visitCount + 1;
userID = (String)session.getAttribute("UserId");
    }
session.setAttribute("visitCount", visitCount);

response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<html>" +
           "<body>" +
           "<h1>Session Infomation</h1>" +
           "<table border='1'>" +
           "<tr><th>Session info</th><th>value</th></tr>" +
           "<tr><td>id</td><td>" + session.getId() + "</td></tr>" +
           "<tr><td>User ID</td<td>" + userID + –</td></tr>" +
           "<tr><td>Number of visits</td><td>" + visitCount + "</td></tr>" +
           "</table></body></html>");
  }
}
```

we b.xml

```
<web-app>
        <servlet>
        <servlet-name>SessionTrack</servlet-name>
        <servlet-class>SessionTrack</servlet-class>
        </servlet>
        <servlet-mapping>
        <servlet-name>SessionTrack</servlet-name>
        <url-pattern>/SessionTrack</url-pattern>
        </servlet- mapping>
</web-app>
```

Output:


Microsoft Word - lab-int
Search results - devineni
localhost:8080/SessionTr
localhost:8080/SessionTrack/SessionTrack

Session Infomation

Session info	value
id	8B74921FDAE0591A0F0ECE9DECC5B516
User ID	Kalpana
Number of visits	6