**Java Applications**

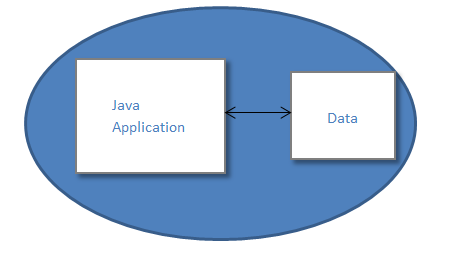
1. **Stand Alone application**

It is a single tier architecture.

In this application and Data resides in the same child process

Eg:

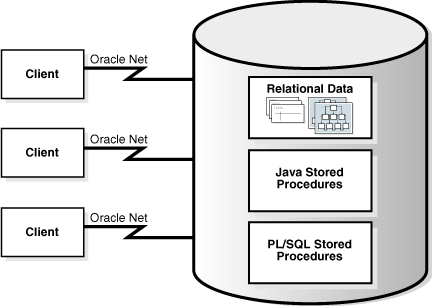
Java application working with files



1. **Client and Server Application:**

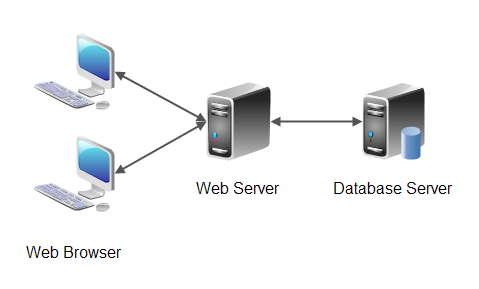
It is a two tier architecture

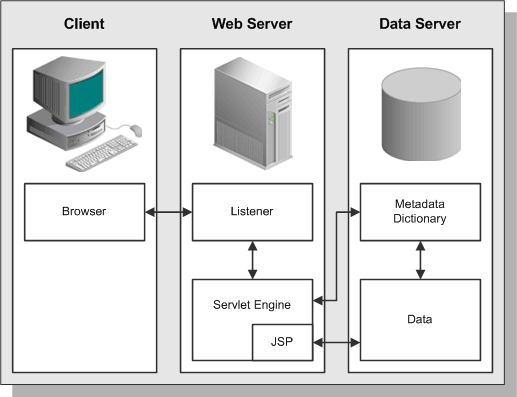
Here data and application resides in two different processes.



1. **Web Application**

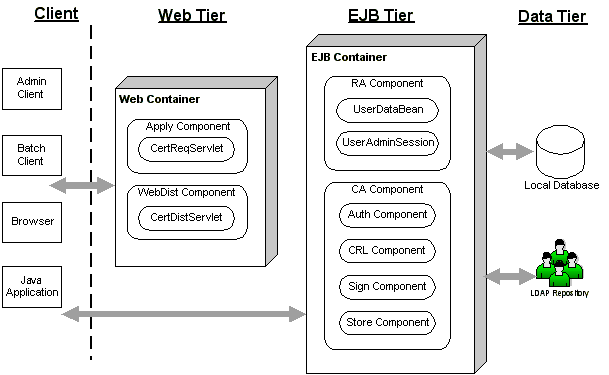
It is a three tier architecture





1. **Enterprise Application:**

It is a N-Tier Architecture:



**Client :**

Client can be a browser, Java application, TCP Client

**Server:**

Server is a device or a computer program that accepts and responds to the request made by other program, known as client. It is used to manage the network resources and for running the program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

**Web Server**

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

It is a computer where the web content can be stored. In general web server can be used to host the web sites but there also used some other web servers also such as FTP, email, storage, gaming etc.

Examples of Web Servers are:

Apache Tomcat

## Microsoft IIS ( Internet Information Services )

## Sun Java System Web Server

**Web Server Working**

It can respond to the client request in either of the following two possible ways:

* Generating response by using the script and communicating with database.
* Sending file to the client associated with the requested URL.

The block diagram representation of Web Server is shown below:



* If the requested web page at the server side is not found, then web server will sends the HTTP response: Error 404 Not found.
* When the web server searching the requested page if requested page is found then it will send to the client with an HTTP response.
* If the client requests some other resources then web server will contact to application server and data is store for constructing the HTTP response.

**Application Server**

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc. It is a component based product that lies in the middle-tier of a server centric architecture.

It provides the middleware services for state maintenance and security, along with persistence and data access. It is a type of server designed to install, operate and host associated services and applications for the IT services, end users and organizations.

**The block diagram representation of Application Server is shown below:**



The Example of Application Servers are:

1. **Weblogic:** Provided by Oracle. It more secured.
2. **Websphere:** Provided by IBM.
3. **JBoss:** Open-source server from JBoss community.
4. **Glassfish:** Provided by Sun Microsystem. Now acquired by Oracle.

**Database Servers:**

1. **Oracle**
2. **MYSQL**
3. **MS-SQL**
4. **DB2**
5. **PostGrey**
6. **Informics**

**What is web application?**

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

Server side Technologies:

These are of two types

1)Process based technology

2)Thread based Technology

1)Process Based technology :

Here separate child process is created for each and every request.

It is heavy weight technology as Process to Process communication is expensive

Very less Performance.

Eg:

CGI ( Common Gate Way Interface )

It is developed in PERL ( Practical Extraction and Report Language ), c

and c++

2) Thread Based Technology :

Here only one child process is created for any no of request.

Each and every request launched as thread in the same process and all thread are shred common child process.

It is a light weight process as only one process is created for any no of requests and Thread to Thread communication is not expensive.

Eg:

Servlets, JSP and ASP

**Servlets vs CGI vs ASP**

**CGI:**

**CGI/Perlis process based**. For every request a **separate process** is started

**Disadvantages of CGI**

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.
4. It is heavy weight process.

**ASP (Active Server Pages)**

The problems with **CGI/PERL** were studied by **Microsoft** and released **ASPas a replacement to CGI**.

The advantage of ASP is it is **thread based** (remember, CGI is process based).

One IIS (Internet Information Server) server software is enough to honor any number of client requests.

It was most wanted in those days. But it comes with many drawbacks (the drawbacks of almost every earlier Microsoft product; one product is designed to be dependable on other Microsoft product).

**Drawbacks of ASP**

1. **ASP works on Windows OS only**. But maximum servers on Web work, even as-on-today also, on **LINUX**. They placed a separate system for Windows and loaded IIS. Now the request comes for a LINUX server, the request is routed to Windows server. The response is routed back to client through Windows and LINUX. Routing is extra process overhead.
2. **ASP works on IIS server onl**y, developed by Microsoft only.
3. **Written in VBScript**, another Microsoft product.

**These drawbacks were fully overcome in Servlets**. Servlets come with lot of advantages and most demanded in the market as-on-today.

**Advantages of Servlets**

All the drawbacks of earlier technologies **CGI/PRRL** and **ASP** are overcome in Servlets.

1. **It is potable.** It is written in Java and Java is platform-independent, Servlets are also platform-independent,
2. **Works on any OS** (ASP is Windows dependent).
3. **Multithreaded** (ASP also supports threads). Threads are lightweight as all threads are spawned from the same process.
4. **Servlets are efficient.** Java is object-oriented whereas **PERL** and **VBScript** are object-based languages. All the drawbacks of object-based languages, like array and string bounds checking, unassigned variables filled with garbage values, are overcome in Servlets. Another Java feature, **Generics** is a big boon to Servlets.
5. **Abundant Web servers support.** Sun Microsystems supplied the **Servlet API** to all the vendors to develop Web servers that execute Servlets. Many came in the market with their own implementations like **Tomcat**, **Weblogic**, **WebSphere**, **Jetty** etc. The software companies has a choice to prefer their own fancy servers of less cost. Infact, **Tomcat is a free server** and most demanded (ASP works only on IIS).
6. **Inter Servlet communication.** One thread data can be shared by other thread. Even if one thread is killed, still the process will be working with other threads. Here, a thread represents a request. With CGI, one process cannot make use of other process data as when response is delivered the process stops.
7. **Servlets are convenient** with high-end utilities like HTTP headers, Cookie support and session management etc.
8. **Servlets are powerful.** Some capabilities exist with Servlets which are difficult in CGI to do. For example, communicating with the Web server. Servlets can communicate directly with the container where as in CGI difficult and requires extra API (which again server specific).
9. **Servlets are secure** as they are executed by JVM and Web container with sandbox security. CGI programs are not much secured as they are executed by OS shells.
10. **Servlets are inexpensive** for the reason the Web servers that execute Servlets are available **free-of-cost like Tomcat** of Apache with commercial quality.
11. **Servlets come with special features** (Java built-in) like exception handling and garbage collection.
12. **Writing dynamic web pages** is easier to Programmer and also faster to run by the container.
13. **Support for communication.** Container communication and Inter servlet communication are easier with built-in APIs.
14. **Support for Sessions.** Due to stateless and connectionless HTTP protocol, the Servlets come with its own API like HttpSession interface to support session tracking.
15. **Support for new frameworks like MVC Struts and Spring.** Lot of frameworks are existing in now-a-days market that makes Servlets simple to practice.

**Disadvantages of Servlets**

Disadvantages are very **trivial** to mention.

1. You are required to know an OOPs language Java indepth. Learning Java being object-oriented, will be cumbersome compared to object-based PERL and VBScript.
2. With complex business logic, sometimes Servlets become difficult to read and understand for others.
3. Servlets are language dependent and should be written in Java only whereas CGI is language independent; ofcourse, PERL is preferred mostly due to text manipulation regular expressions.
4. Business logic and present logic should be developed together. These logics can not be separated.

Later **JSP** was released by Sun Microsystems. JSP is still easier than Servlets. Infact, JSP is internally converted to a Servlet and executed.

With JSP we can separate the presentation logic from business logic.

**Advantage of Servlet**

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language..

**Servlets**

## What are Servlets?

## Java servlets are server side components

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP 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.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

Advantages:

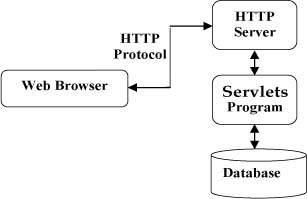
Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.

Servlets are platform-independent because they are written in Java.

Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.

## Servlets Architecture:

Following diagram shows the position of Servlets in a Web Application.



## 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

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.

**Tomcat server setup:**

1) Down load tomcat server

===========================

apache-tomcat-8.5.39

on google search for : apache mirrors

apache.mirrors.tds.net/

open tomcat folder:

goto folder -->[tomcat/tomcat-8/v8.5.39/bin/](http://apache.mirrors.tds.net/tomcat/tomcat-8/v8.5.39/bin/)

down load below software:

[apache-tomcat-8.5.39-windows-x64.zip](http://apache.mirrors.tds.net/tomcat/tomcat-8/v8.5.39/bin/apache-tomcat-8.5.39-windows-x64.zip)

2) Environment setup:

=================

after down loading extract it and set the environment as below

Setting the JAVA\_HOME, CATALINA\_HOME Environment Variable on Windows

One can do using command prompt:

set JAVA\_HOME=C:\ "top level directory of your java install"

set CATALINA\_HOME=C:\ "top level directory of your Tomcat install"

set PATH=%PATH%;%JAVA\_HOME%\bin;%CATALINA\_HOME%\bin

OR you can do the same:

Go to system properties

Go to environment variables and add a new variable with the name JAVA\_HOME and provide variable value as C:\ "top level directory of your java install"

Go to environment variables and add a new variable with the name CATALINA\_HOME and provide variable value as C:\ "top level directory of your Tomcat install"

In path variable add a new variable value as ;%CATALINA\_HOME%\bin;

eg:

Variable Name : JAVA\_HOME

Value : C:\Program Files\Java\jdk1.8.0\_121

================================================

Variable Name : CATALINA\_HOME

Value : C:\apache-tomcat-8.5.39 =======================================

Variable Name : PATH

Value : %PATH%;C:\app\nagaswarao\product\11.2.0\dbhome\_1\bin;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Skype\Phone\;%JAVA\_HOME%\bin;%CATALINA\_HOME%\bin

================================================

Variable Name : CLASSPATH

Value : %CLASSPATH%;C:\apache-tomcat-7.0.73\lib\servlet-api.jar

3)Start tomcat Server:

c:>startup

4) goto tomcat manager

<http://localhost:8080/manager/html>

tomcat userid and password:

go to folder “C:\apache-tomcat-8.0.41\conf” 🡪tomcat-users.xml

update with following

<role rolename="manager-gui"/>

<user username="admin" password="admin" roles="manager-gui"/>

5) Deploying .war file:

goto

War file to deploy --> select browse and browse for your .war and open

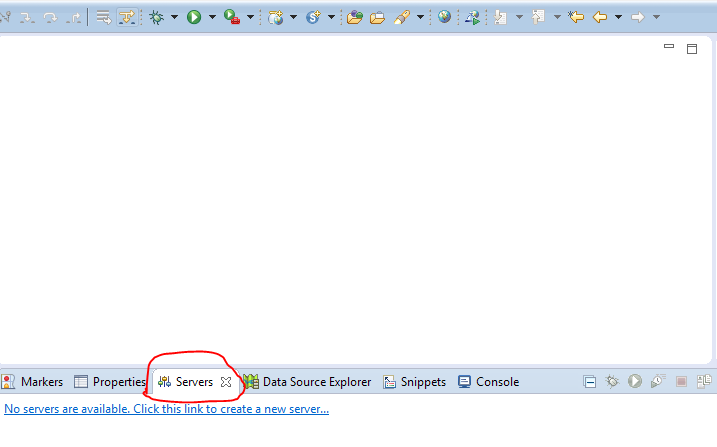
click on deploy.

**Developing Web Application using eclipse and tomcat server:**

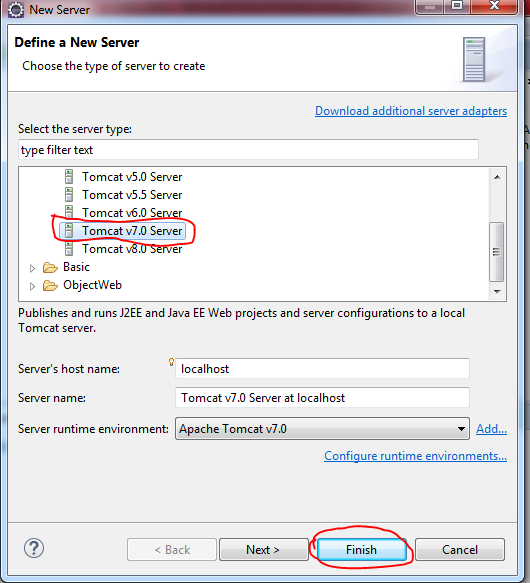
**Step1:**

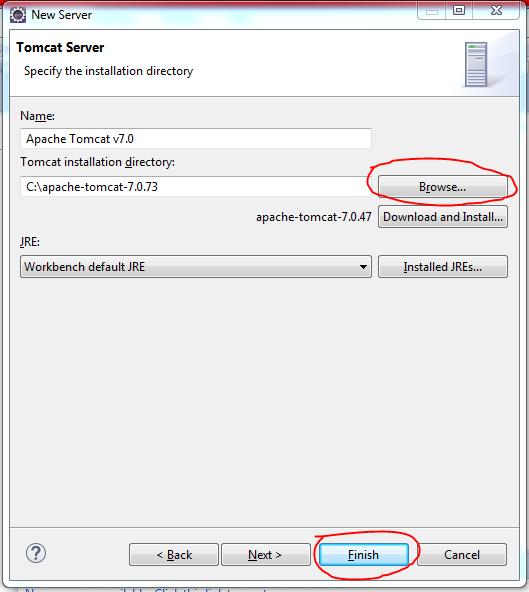
**Tomcat configuration for eclipse:**

**Click on “server” tab as below:**

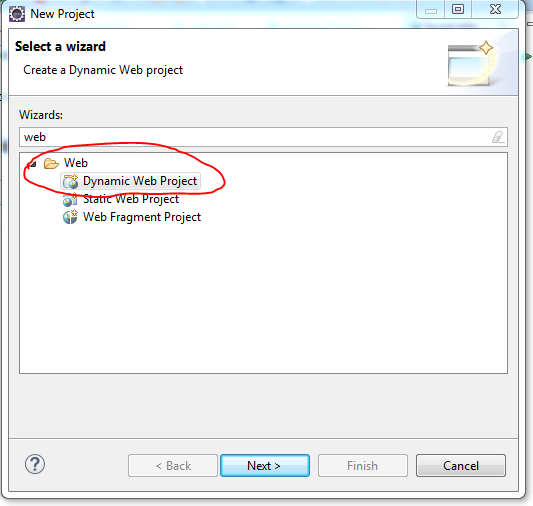


**Right click on blank space 🡪 new 🡪server**



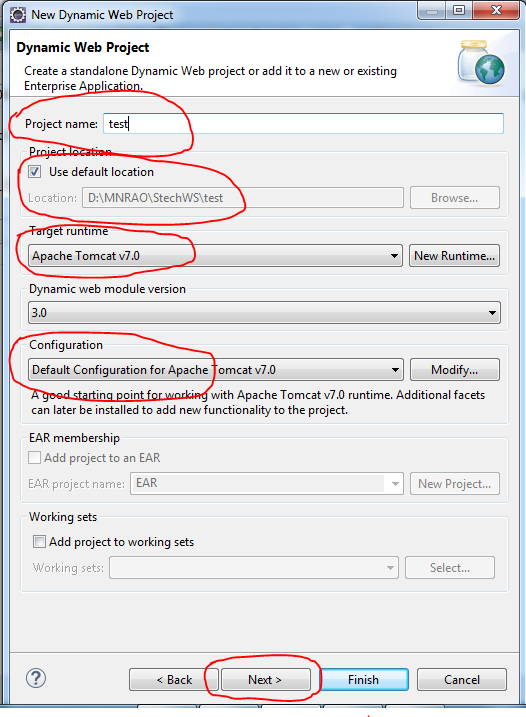


For creating a dynamic web project **click on File Menu -> New -> Project..-> Web -> dynamic web project -> write your project name e.g. test -> Finish**.

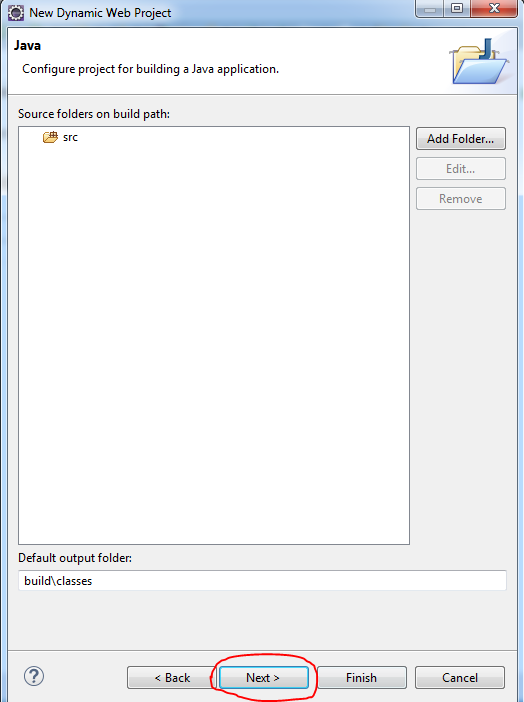


Click on “Next”

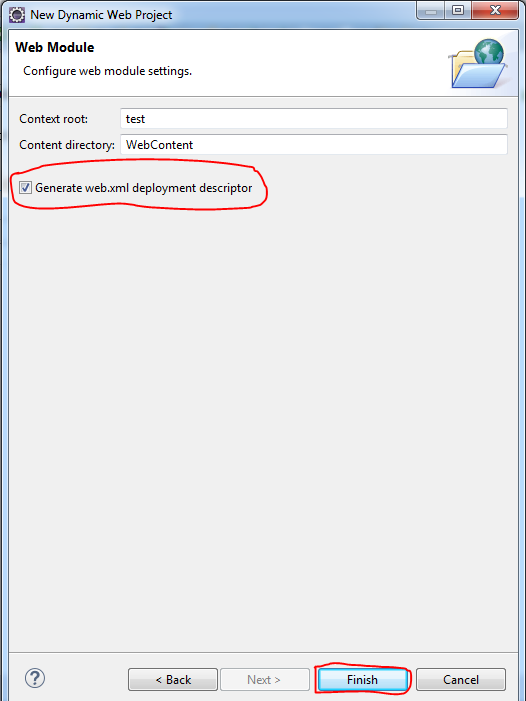
Fill the marked fields in the below:



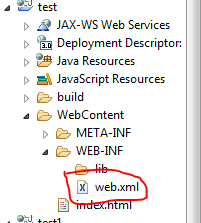
And click on Next in below :



Select “Generate web.xml deployment descriptor as mensioned below:

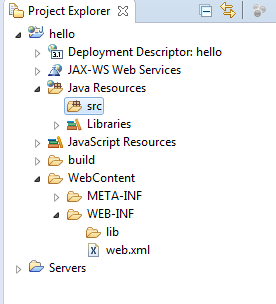


**Define application properties in web.xml:**



**Developing HelloServlet:**

**Go to src as below :**



**Create a package:**

**Goto src and create**

**com.durga.mnrao.test**

**Develop the servlet as below:**

**package** com.durga.mnrao.test;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.GenericServlet;

**import** javax.servlet.ServletException;

**import** javax.servlet.ServletRequest;

**import** javax.servlet.ServletResponse;

**public** **class** HelloServlet **extends** GenericServlet {

@Override

**public** **void** service(ServletRequest req, ServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/plain");

PrintWriter out = res.getWriter();

out.println("Hello java World");

}

}

Prepare web.xml file:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"* id=*"WebApp\_ID"* version=*"3.1"*>

<servlet>

<servlet-name>hello</servlet-name>

<servlet-class>com.durga.mnrao.test.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>hello</servlet-name>

<url-pattern>/myhello</url-pattern>

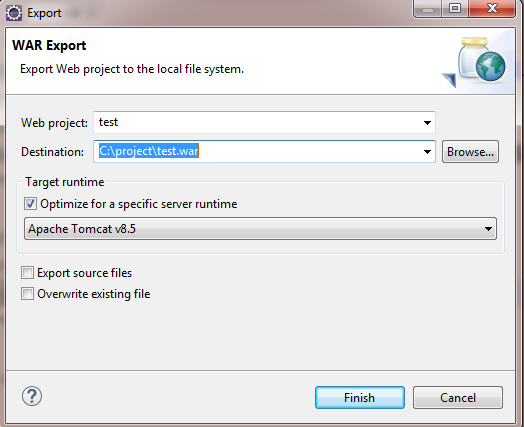
</servlet-mapping>

</web-app>

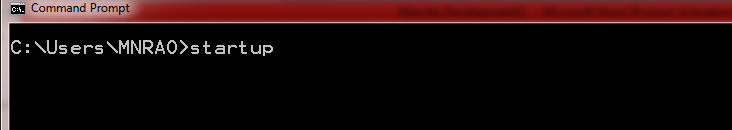
**Create .war file :**

**Rt.click on project 🡪 export 🡪war 🡪**

**Browse for location and finish**



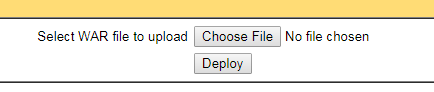
**Goto command line start tomcat server**



**Go to tomcat manager and deploy the .war file**

<http://localhost:8080/manager/html>

select war file to upload:



Send the request for the HelloServlet:

<http://localhost:8080/test/myhello>

here :

<http://localhost:8080> 🡪 tomcat URL

test 🡪 name of .war file

/myhello 🡪 url-pattern in web.xml

**Extends HttpServlet:**

**package** com.durga.mnrao.test;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** HelloServlet **extends** HttpServlet{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** **void** service(HttpServletRequest req, HttpServletResponse res) **throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter out = res.getWriter();

out.print("<h1> hello durga soft </h1>");

}

}

**Web.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"* id=*"WebApp\_ID"* version=*"3.1"*>

<servlet>

<servlet-name>test</servlet-name>

<servlet-class>com.durga.mnrao.test.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>test</servlet-name>

<url-pattern>/mytest</url-pattern>

</servlet-mapping>

</web-app>

**Working with servlets:**

## Servlets Packages:

1. **javax.servlet** and 2) **javax.servlet.http**

These classes implement the Java Servlet and JSP specifications, the versions are Java Servlet 2.5 and JSP 2.1

The three core elements of the Servlet API are the

1. javax.servlet.Servlet interface,
2. the javax.servlet.GenericServlet class,
3. javax.servlet.http. HttpServlet class

A Servlet can extend from

1. Servlet ( interface )
2. GenericServlet ( abstract class )
3. HttpServlet ( it is a fully concrete class, no abstract method but declared as **Abstract class** )

Servlet interface methods:

1. **public void init(ServletConfig config)**

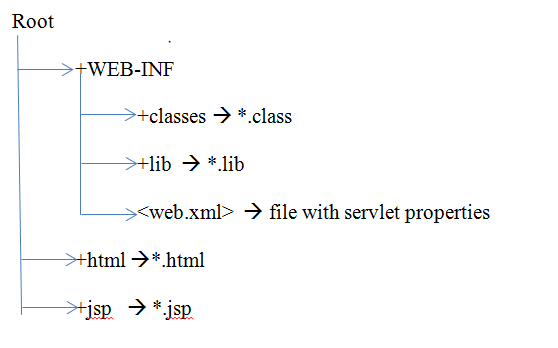
initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.

1. **public void service(ServletRequest request,ServletResponse response)**

provides response for the incoming request. It is invoked at each request by the web container.

1. **public void destroy()🡪**is invoked only once and indicates that servlet is being destroyed.
2. **public ServletConfig getServletConfig()🡪**returns the object of ServletConfig.
3. **public String getServletInfo()🡪**returns information about servlet such as writer, copyright, version etc.

**Below is the folder structure for web application:**



**Developing HelloServlet by extends of GenericServlet:**

import javax.servlet.\*;

import java.io.\*;

public class HelloServlet extends GenericServlet

{

public void service(ServletRequest req, ServletResponse resp)

throws ServletException, IOException

{

resp.setContentType("text/plain");

PrintWriter out = resp.getWriter();

// We won't use the ServletRequest object in this example

out.println("Hello.");

}

}

D:>testapp>notepad HelloServlet.java

Compile and get .class file for the above

D:>testapp>javac HelloServlet.java

1. Create a temp folder in a D: drive
2. Create WEB-INF under temp folder
3. Create classes folder under WEB-INF folder
4. Copy HelloServlet.class file into classes folder
5. Prepare web.xml as below:

<web-app>

<servlet>

<servlet-name>hello</servlet-name>

<servlet-class>HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>hello</servlet-name>

<url-pattern>/myhello</url-pattern>

</servlet-mapping>

</web-app>

1. Copy above web.xml file into WEB-INF folder
2. Generate .war file for the above

Go to cmd -> change to temp directory

D: > temp> jar –cvf hello.war \*

1. Start tomcat server and Deploy the .war file on to tomcat server:

C:>startup 🡪 to start tomcat server

Go to tomcat manager:

<http://localhost:8080/manager/html>

configuring userid and passwd:

go to tomcat home 🡪 conf🡪<tomcat-users>

C:\apache-tomcat-7.0.73\conf\tomcat-users.xml

<role rolename="manager-gui"/>

<user username="admin" password="admin" roles="manager-gui"/>

======================

go to “ war file to deploy “

click on browse button:

select .war file and open

and finally click on deply

1. Send request to servlet using below URL:

<http://localhost:8080/hello/myhello>

in the above :

http 🡪 browser protocol

localhost 🡪 is a location where tomcat is running

hello 🡪 is a name of .war file

myhello 🡪 URL pattern in web.xml file

====================================================================

**HttpServlet class:**

The HttpServlet class is an extension of GenericServlet that includes methods

for handling HTTP-specific data.

Here’s a simple HttpServlet:

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

public class HelloServletTest extends HttpServlet

{

public void service(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException

{

resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

out.println("<HTML>");

out.println("<HEAD><TITLE>This is second example?</TITLE></HEAD>");

out.println("<BODY><H1>Hello,World!</H1><H6>Again.</H6></BODY></HTML>");

}

}

HttpServlet is an abstract class, implemented by the provider of the servlet container.

HttpServlet also provides a number of methods,

such as doGet(), doPost(), and doPut(), to handle particular types of HTTP requests (GET, POST, and so on).

These methods are called by the default implementation of the service() method, which figures out what kind of request is being made and then invokes the appropriate method.

# Servlets - Life Cycle:

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

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

Now let us discuss the life cycle methods in details.

## 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. Such data base connection opening, tcp sockets opening etc.

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.

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:

publicvoid service(ServletRequest request,ServletResponse response)

throwsServletException,IOException

{

}

The service () method is called by the container and service method invokes doGet, 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 doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

publicvoid doGet(HttpServletRequest request,HttpServletResponse response)

throwsServletException,IOException

{

// Servlet code

}

## The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

publicvoid doPost(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.

The destroy method definition looks like this:

publicvoid destroy()

{

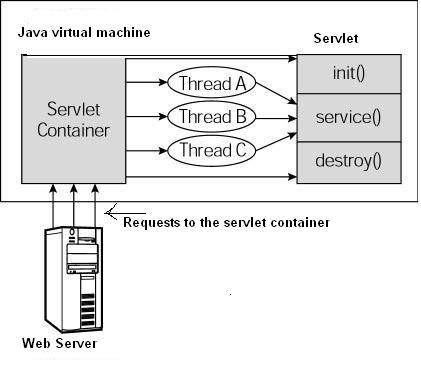
// Finalization code...

}

## Architecture Digram:

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.



Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface.

Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

## Sample Code for Hello World:

**Servlet program to implement Life Cycle methods of a servlet**

**i**mport javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

public class CounterServlet extends HttpServlet

{

private int counter;

public void init() throws ServletException

{

counter=0;

System.out.println("Init Called count : "+counter);

}

public void service(HttpServletRequest req, HttpServletResponse res ) throws ServletException, IOException

{

counter++;

res.setContentType("text/html");

PrintWriter out = res.getWriter();

out.println("<HTML>");

out.println("<HEAD>");

out.println("<TITLE>Counter</TITLE>");

out.println("</HEAD>");

out.println("<BODY>");

out.println("<H1>Service method "+counter+"</H1>");

out.println("</BODY>");

out.println("</HTML>");

# }

# public void destroy()

# {

# System.out.println("Destroy Called "+counter);

# }

# }

# Servlets - Form Data

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.gmail.com/login?uid=value1&pwd=value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box.

GET method is not recommended, if you have password or other sensitive information to pass to the server.

The GET method has size limtation: only 2048 characters can be in a request string.

Servlet handles this type of requests using **doGet()** method.

## POST method:

POST is more reliable method of passing information to a backend program.

This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after ( ? ) in the URL it sends it as a separate message.

This message comes to the backend program in the form of the standard input which you can parse and use for your processing.

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():**

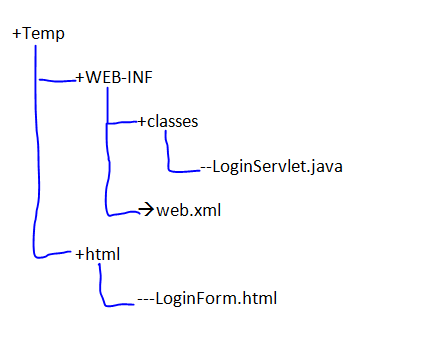
request.getParameter() 🡪 to get the value of a form parameter.

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

## GET Method Example Using URL:

**<http://localhost:8080/logonproj/mylogin?uid=mnrao&pwd=admin12345>**

Project Hierarchy:



LoginForm.html :

<html>

<body>

<form action="./mylogin" method="GET">

User Id: <input type="text" name="uid">

<br/>

Password: <input type="text" name="pwd" />

<input type="submit" value="Submit" />

</form>

</body>

</html>

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

public class LoginServlet extends HttpServlet

{

public void doGet(HttpServletRequest request, HttpServletResponse response ) throws ServletException, IOException

{

doPost(request,response);

}

public void doPost(HttpServletRequest request, HttpServletResponse response ) throws ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String userId=request.getParameter("uid");

String password=request.getParameter("pwd");

if(userId.equalsIgnoreCase("nrit") && password.equals("java"))

{

out.println("<H1>Welcome to User : "+userId+"</H1>");

}

else

{

out.println("<H1>Invalid Userid and password : "+userId+"</H1>");

}

}

}

Web.xml:

==========

<web-app>

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/mylogin</url-pattern>

</servlet-mapping>

</web-app>

Create .war file for the above project

Change to Temp dir

D:>cd temp

D:>temp>jar –cvf loginproj.war \*

Go to tomacat manager:

<http://localhost:8080/manager/html>

at bottom click on button “Browse” ---🡪 browse for loginproj.war select it and click on “open” and then deploy.

Open another tab in browser and send following request:

<http://localhost:8080/loginproj/html/LoginForm.html>

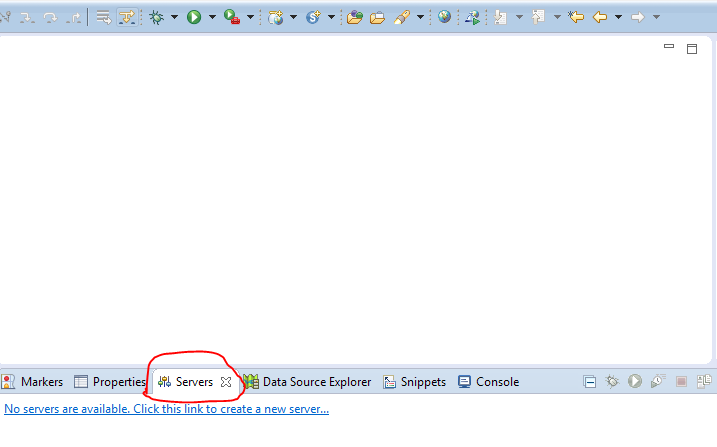
fill the form and submit

**Developing Web Application using eclipse and tomcat server:**

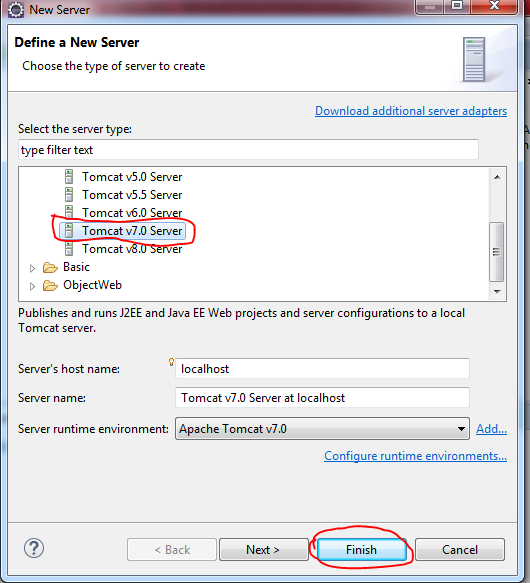
**Step1:**

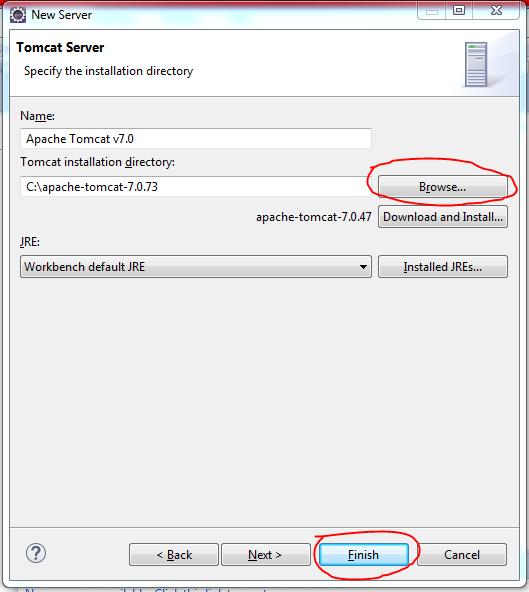
**Tomcat configuration for eclipse:**

**Click on “server” tab as below:**

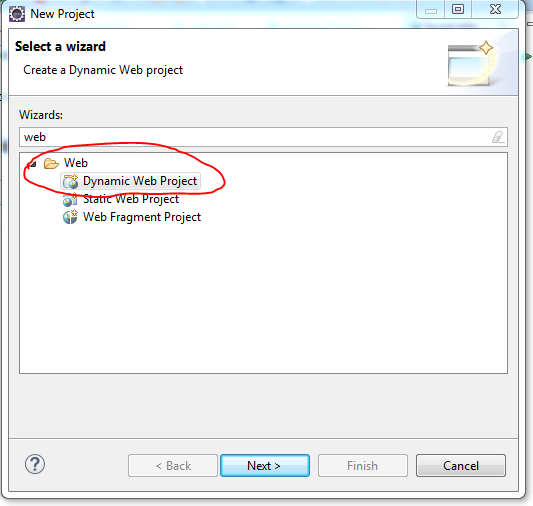


**Right click on blank space 🡪 new 🡪server**



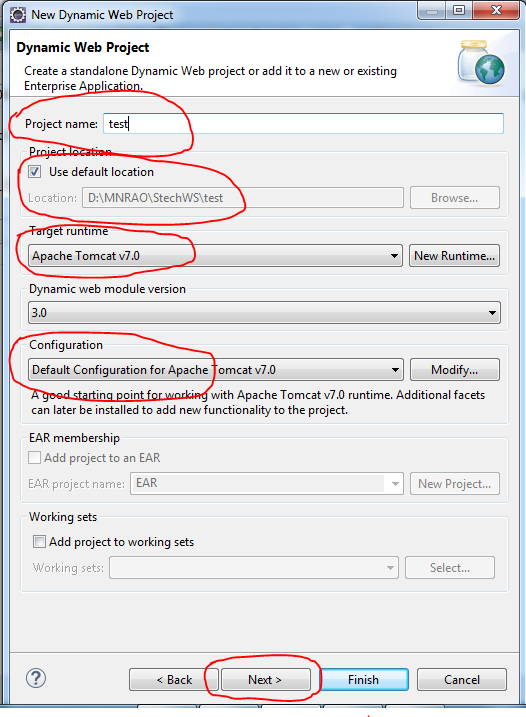


For creating a dynamic web project **click on File Menu -> New -> Project..-> Web -> dynamic web project -> write your project name e.g. test -> Finish**.

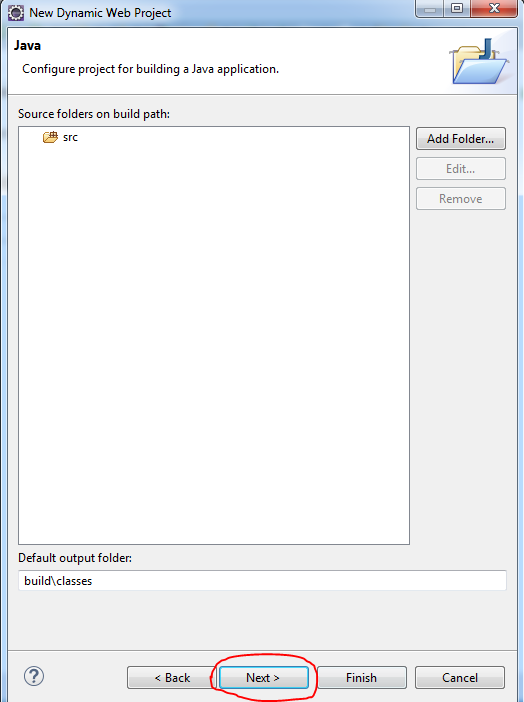


Click on “Next”

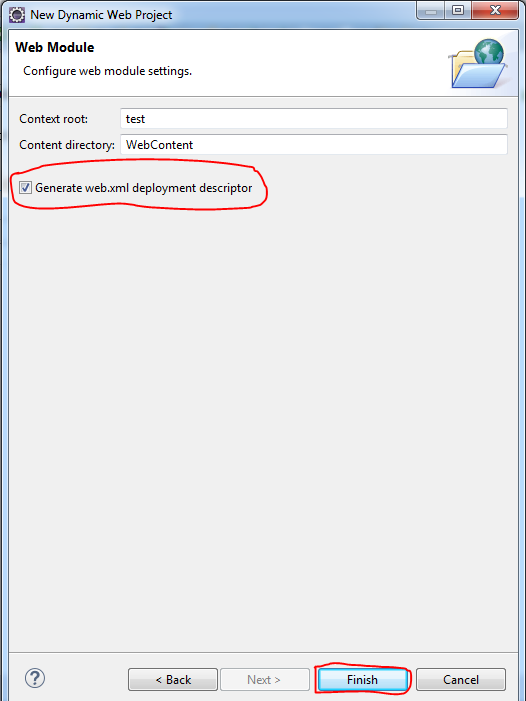
Fill the marked fields in the below:



And click on Next in below :



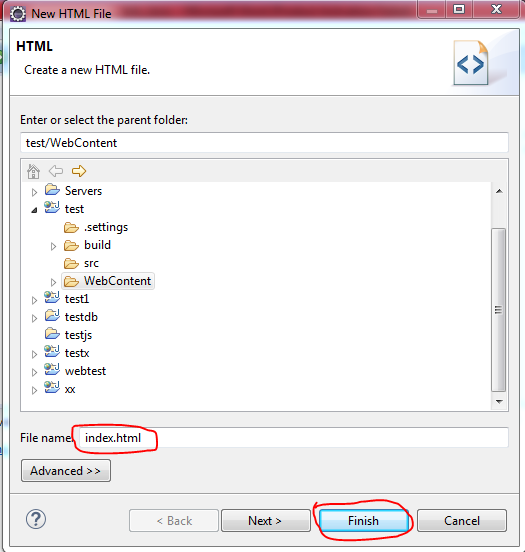
Select “Generate web.xml deployment descriptor as mensioned below:



Click on Finish:

Creating “index.html” file

Right click on project 🡪 new 🡪 HTML



Index.html coding :

<!DOCTYPEhtml>

<html>

<head>

<metacharset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

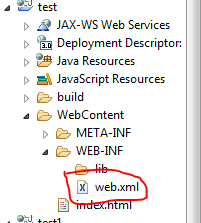
<body>

<h1>Hello World</h1>

</body>

</html>

**Define application properties in web.xml:**



For Eg:

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*id=*"WebApp\_ID"*version=*"3.1"*>

<display-name>sample</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

**How to run the above web application:**

Right click on the project🡪 Run As 🡪 Run on ServerSelect TomacatServer 🡪 and

**LoginServlet Example:**

Index.html coding :

<!DOCTYPEhtml>

<html>

<head>

<title>login form</title>

</head>

<body>

<formaction=*"mylogin"*method=*"post"*>

User ID:<inputtype=*"text"*name=*"uid"*/><br/>

Password:<inputtype=*"password"*name=*"pwd"*/><br/>

<inputtype=*"submit"*value=*"login"*/>

</form>

</body>

</html>

**Define application properties in web.xml:**

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*id=*"WebApp\_ID"*version=*"3.1"*>

<display-name>first</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>com.nrit.mnrao.log.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/mylogin</url-pattern>

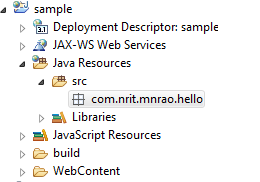
</servlet-mapping>

</web-app>

**Create package:**

Go to java resources and create a package :

Right click on src and create new package “com.nrit.mnrao.log”



**Create LoginServlet.java:**

Right click on package 🡪 new 🡪 Class

LoginServlet:

Write the code as below:

**package** com.nrit.mnrao.log;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass** LoginServlet **extends** HttpServlet{

**privatestaticfinallong*serialVersionUID*** = 1L;

@Override

**protectedvoid** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

resp.setContentType("text/html");

PrintWriter out = resp.getWriter();

String userId = req.getParameter("uid");

String password = req.getParameter("pwd");

**if**(userId.equalsIgnoreCase("nrit") &&password.equals("java"))

{

out.println("<h1> Welcome to User"+ userId+ "</h1>" );

}

**else**

{

out.println("<h1>Invalid user </h1>");

}

}

@Override

**protectedvoid** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

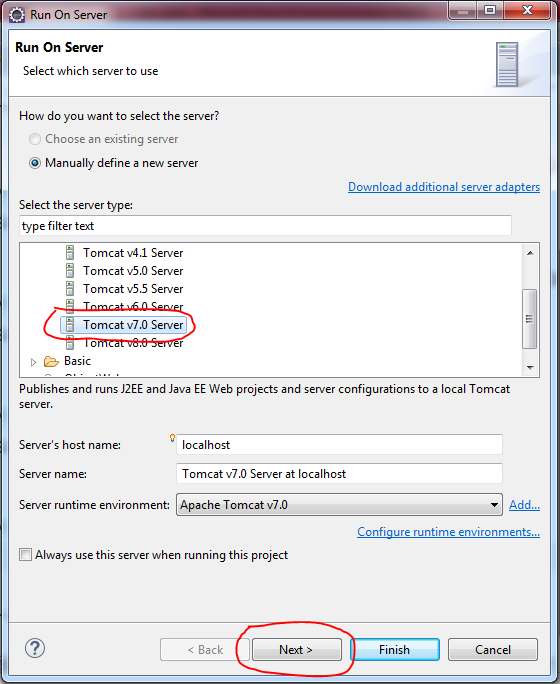
doGet(req, resp);

}

}

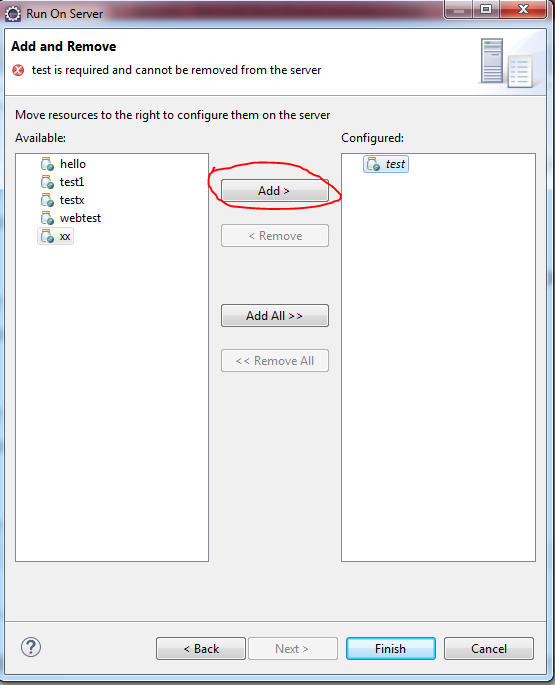
**How to run the above web application:**

Right click on the project🡪 Run As 🡪 Run on Server🡪Select TomacatServer 🡪 and



Click on “Next”

In below wizard select name of the project and click on “add” 🡪 it moves right side



Click on “Finish”

## Servlet vs GenericServlet vs HttpServlet

## Servlets are ****platform-independent**** server-side components, being written in Java. Before going for differences, first let us see how the three ****Servlet****, ****GenericServlet****, ****HttpServlet**** are related, their signatures and also at the end similarities.

## Following figure shows the hierarchy of Servlet vs GenericServlet vs HttpServlet and to know from where HttpServlet comes.

## Servlet vs GenericServlet vs HttpServlet

## Observe the hierarchy and understand the relationship between the three (involved in multilevel inheritance). With the observation, a conclusion can be arrived, to write a Servlet three ways exist.

## ****a)**** by implementing Servlet (it is interface) ****b)**** by extending GenericServlet (it is abstract class) ****c)**** by extending HttpServlet (it is abstract class)

## The disadvantage of the first way is, all the 5 abstract methods of the interface ****Servlet**** should be overridden eventhough Programmer is not interested in all

A smart approach is inheriting **GenericServlet**and overriding its only one abstract method **service()**. It is enough to the programmer to override only this method. It is a **callback** method (called implicitly)

extending **HttpServlet( like inheriting from adapter classes)** and need not to override any methods as HttpServlet contains no abstract methods. Eventhough the HttpServlet does not contain any abstract methods, it is declared as abstract class by the Designers to not to allow the Programmer to create an object directly because a Servlet object is created by the system (here system is Servlet Container).

**1. Servlet interface**

It is the super interface for the remaining two – GenericServlet and HttpServlet. It contains 5 abstract methods and all inherited by GenericServlet and HttpServlet. Programmers implement Servlet interface who would like to develop their own container.

**2. GenericServlet**

It is the immediate subclass of **Servlet interface**. In this class, only one abstract method **service()** exist. Other 4 abstract methods of Servlet interface are given implementation (given body). Anyone who extends this **GenericServlet** should override **service()** method. It was used by the Programmers when the Web was not standardized to **HTTP** protocol. It is protocol independent; it can be used with any protocol, say, SMTP, FTP, CGI including HTTP etc.

Signature:

**public abstract class GenericServlet extends java.lang.Object implements Servlet, ServletConfig, java.io.Serializable**

**3. HttpServlet**

When HTTP protocol was developed by W3C people to suit more Web requirements, the Servlet designers introduced **HttpServlet** to suit more for HTTP protocol. HttpServlet is protocol dependent and used specific to HTTP protocol only.

The immediate super class of **HttpServlet** is **GenericServlet**. HttpServlet overrides the **service()** method of GenericServlet. HttpServlet is abstract class but without any abstract methods.

With HttpServlet extension, **service()** method can be replaced by **doGet()** or **doPost()** with the same parameters of service() method.

Signature:

**public abstract class HttpServlet extends GenericServlet implements java.io.Serializable**

**Difference between GenericServlet and HttpServlet**

|  |  |  |
| --- | --- | --- |
| S.No | **GenericServlet** | **HttpServlet** |
| 1 | Can be used with any protocol (means, can handle any protocol). Protocol independent. | Should be used with HTTP protocol only (can handle HTTP specific protocols) . Protocol dependent. |
| 2 | All methods are concrete except service() method. service() method is abstract method. | All methods are concrete including service() |
| 3 | service() should be overridden being abstract in super interface. | service() method need not be overridden. |
| 4 | It is a must to use service() method as it is a callback method. | Being service() is non-abstract, it can be replaced by doGet() or doPost() methods. |
| 5 | Extends Object and implements interfaces Servlet, ServletConfig and Serializable. | Extends GenericServlet and implements interface Serializable |
| 6 | Direct subclass of Servet interface. | Direct subclass of GenericServlet. |
| 7 | Defined in javax.servlet package. | Defined in javax.servlet.http package. |
| 8 | All the classes and interfaces belonging to javax.servlet package are protocol independent. | All the classes and interfaces present in javax.servlet.http package are protocol dependent (specific to HTTP). |
| 9 | GenericServlet supports only service() method does not contain doGet() and doPost() methods. | HttpServlet support also doGet(), doPost(), doHead() methods (HTTP 1.0) plus doPut(), doOptions(), doDelete(), doTrace() methods (HTTP 1.1). |
| 10 | Use Service method. | Use doPost, doGet method instead of service method. |
| 11 | Not used now-a-days. | Used always. |

**Similarities :**

1. One common feature is both the classes are abstract classes.  
2. Used with Servlets only.

## Passing Checkbox Data to Servlet Program

CheckBoxForm.html:

<html>

<body>

<form action="./myCheckBox" method="POST" target="\_blank">

<input type="checkbox" name="maths" checked="checked" /> Maths

<br/>

<input type="checkbox" name="physics" /> Physics

<br/>

<input type="checkbox" name="chemistry" checked="checked " /> Chemistry

<br/>

<input type="submit" value="Select Subject" />

</form>

</body>

</html>

Web.xml:

===========

<web-app>

<servlet>

<servlet-name>cb</servlet-name>

<servlet-class>CheckBoxServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>cb</servlet-name>

<url-pattern>/myCheckBox</url-pattern>

</servlet-mapping>

</web-app>

**CheckBoxServlet.java:**

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class CheckBoxServlet extends HttpServlet

{

// Method to handle GET method request.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String mathsFlag=request.getParameter("maths");

String physicsFlag=request.getParameter("physics");

String chemistryFlag=request.getParameter("chemistry");

String result = "Math :"+mathsFlag+"\n"+"Physics : "+physicsFlag+"\n"+"Chem :"+chemistryFlag;

out.println("<h>"+result+"</h>");

}

// Method to handle POST method request.

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

**HTTP (Hyper Text Transfer Protocol)**

The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.

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

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

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

There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

* **HTTP is media independent:** It refers to any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.
* **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sends the client disconnects from server and waits for the response.
* **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

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

## Servlet HTTP5

## Differences GET vs POST Which to prefer?

## *The developer should specify to the Browser in what style the data is to be sent to the server. The most commonly used styles and available are GET and POST. These styles are to be written in METHOD attribute of <FORM> tag as follows. Let us see their differences and when to use them in GET vs POST.*

<form method="GET" action="http://localhost:8080/myproject/test">

**In GET style, the data is appended to the URL (written in action attribute in the above statement) and sent to the server and in POST style, the client data is sent separately as message body.**

**Eg:**

**Browser Request:**

**http://locahost:8080/myproject/html/Login.html**



Observe the URL in browser prompt.

Output screen when submit button is clicked.

<http://localhost:8888/myproject/login?uid=MNRAO&pwd=MNRAO12345>

hence this is not secured approach

|  |  |  |
| --- | --- | --- |
| Features | GET | POST |
| Sending of data | Client data is appended to URL and sent | Client data is sent implicitly |
| Storing in Browser History | As data is appended, the client data is stored in browser history | As data is sent implicitly, the client data is not stored in browser history |
| Bookmark | The URL with client data can be bookmarked. Thereby, later without filling the HTML form, the same data can be sent to server | Not possible to bookmark |
| Encoding or encrypte | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. For binary data, multipart enctype to be used |
| Limitation of data sent | Limited to 2048 characters (browser dependent) | Unlimited data |
| Hacking easiness | Easy to hack the data as data is stored in browser history | Difficult to hack |
| Type of data sent | Only ASCII data can be sent | Any type of data can be sent including binary data |
| Data secrecy | Data is not secret as other people can see the data in browser history | Data is secret as not stored in history |
| When to be used | Prefer when data sent is not secret. Do not use for passwords etc. | Prefer for critical and sensitive data like passwords etc. |
| Cache | Can be caught | Cannot be caught |
| Default | Relatively faster as data is appended to URL | A separate message body is to be created |

**Which is to be preferred – GET or POST?**

It depends on your application need. If client data includes only **ASCII** characters, no secrecy and limited to **2KB** length (depends on the browser), then prefer **GET**, else **POST**

## Other HTTP Request Method

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

## Reading All Form Parameters:

**getParameterNames()** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.

ReadParams.html:

<html>

<body>

<form action="./myparams" method="POST" target="\_blank">

<input type="checkbox" name="maths" checked="checked" /> Maths

<input type="checkbox" name="physics" /> Physics

<input type="checkbox" name="chemistry" checked="checked" /> Chem

<input type="submit" value="Select Subject" />

</form>

</body>

</html>

Enumeration paramNames = request.getParameterNames();

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.\*;

// Extend HttpServlet class

public class ReadAllParams extends HttpServlet {

// Method to handle GET method request.

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

Enumeration paramNames = request.getParameterNames();

while(paramNames.hasMoreElements())

{

String paramName = (String)paramNames.nextElement();

out.print("<tr><td>" + paramName + "</td>\n<td>");

String[] paramValues = request.getParameterValues(paramName);

// Read single valued data

if (paramValues.length == 1)

{

String paramValue = paramValues[0];

if (paramValue.length() == 0)

out.println("<i>No Value</i>");

else

out.println(paramValue);

}

else

{

// Read multiple valued data

out.println("<ul>");

for(int i=0; i < paramValues.length; i++)

{

out.println("<li>" + paramValues[i]);

}

out.println("</ul>");

}

}

out.println("</tr>\n</table>\n</body></html>");

}

// Method to handle POST method request.

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

}

Web.xml:

=======

<web-app>

<servlet>

<servlet-name>mp</servlet-name>

<servlet-class>ReadAllParams</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>mp</servlet-name>

<url-pattern>/myparams</url-pattern>

</servlet-mapping>

</web-app>

## Methods to read HTTP Header:

These methods are available with *HttpServletRequest* object.

**1) String getMethod()**

Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.

String method = request.getMethod();

**2) String getParameter(String name)**

Returns the value of a request parameter as a String, or null if the parameter does not exist.

String userId = request.**getParameter** (“uid”);

**3) String[]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.

String [] values = request**.**getParameterValues(“ckboxGroup”);

**4) Enumeration getParameterNames()**

Returns an Enumeration of String objects containing the names of the parameters contained in this request.

Enumerationnames = request.getParameterNames();

**while**(names.hasMoreElements())

{

String name = (String) names.nextElement();

out.println("param name : "+name);

String value = request.getParameter (name);

}

**5) String getQueryString()**

Returns the query string that is contained in the request URL after the path.

String queryString **=** request.getQueryString();

Eg:

<http://localhost:8888/myproject/login?uid=MNRAO&pwd=MNRAO12345>

**queryString**value is following.

uid=MNRAO&pwd=MNRAO12345

**6) int getContentLength()**

Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.

int length **=** request**.**getContentLength();

**7) String getContentType()**

Returns the MIME (Multipurpose Internet Mail Extensions) type of the body of the request ( non ASCII Chars ) , or null if the type is not known.

String type = request. getContentType();

**8) String getRemoteAddr()**

Returns the Internet Protocol (IP) address of the client,that sent the request.

String address = request .getRemoteAddr();

**9) String getRemoteHost()**

Returns the fully qualified name of the client,that sent the request.

String hostName = request.getRemoteHost();

**10) String getRemoteUser()**

Returns the login of the user making this request, if the user has been authenticated,or null if the user has not been authenticated.

String userName = request.getRemoteUser ();

**11) String getRequestURI()**

Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.

String url = request.getRequestURI();

Eg:

<http://localhost:8888/myproject/login?uid=MNRAO&pwd=MNRAO12345>

url 🡪<http://localhost:8888/myproject/login>

uri 🡪http://localhost:8888/myproject

**12) String getServletPath()**

Returns the part of this request's URL that calls the JSP.

String path = request.getServletPath();

**13) String getPathInfo()**

Returns any extra path information associated with the URL the client sent when it made this request ( pathname inside the claees folder)

String path = request.getPathInfo();

**14) int getServerPort()**

Returns the port number on which this request was received ( 8080 )

int port = request .getServerPort();

**15) String getProtocol()**

Returns the name and version of the protocol the request ( default http )

String protocol = request .getProtocol();

**16) Object getAttribute(String name)**

Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.

Used in Servelet communications.

Object attribValue = request.getAttribute(“uid”);

**17) String getHeader(String name)**

Returns the value of the specified request header as a String.

String header = request .getHeader(“abc”);

**18) int getIntHeader(String name)**

Returns the value of the specified request header as an int.

int header = request .getIntHeader(“abc”);

**19) Enumeration getAttributeNames()**

Returns an Enumeration containing the names of the attributes available to this request.

Enumeration names = request .getAttributeNames();

**while**(names.hasMoreElements())

{

String name = (String) names.nextElement();

out.println("param name : "+name);

String value = request.getAttribute(name);

}

**20) Enumeration getHeaderNames()**

Returns an enumeration of all the header names this request contains.

Enumeration names = request .getHeaderNames ();

**while**(names.hasMoreElements())

{

String name = (String) names.nextElement();

out.println("param name : "+name);

String value = request.getHeader (name);

}

**21) HttpSession getSession()**

Returns the current session associated with this request, or if the request does not have a session, creates one.

HttpSession session = request.getSession();

**22) HttpSession getSession(boolean create)**

Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

HttpSession session = request .getSession(true);

**23) String getRequestedSessionId()**

Returns the session ID specified by the client.

HttpSession sessionId = request .getRequestedSessionId ();

**24) String getAuthType()**

Returns the name of the authentication scheme used to protect the servlet, for example, "BASIC" or "SSL," or null if the JSP was not protected.

String authType = request.getAuthType();

**25) boolean isSecure()**

Returns a boolean indicating whether this request was made using a secure channel, such as HTTPS.

boolean secure = request .isSecure();

**26) Cookie[] getCookies()**

Returns an array containing all of the Cookie objects the client sent with this request.

Cookie[] cockies = request. getCookies();

**27) Locale getLocale()**

Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.

Locale language = request .getLocale();

**28) ServletInputStream getInputStream()**

Retrieves the body of the request as binary data using a ServletInputStream.

ServletInputStream is= request.getInputStream();

**29) String getCharacterEncoding()**

Returns the name of the character encoding used in the body of this request.

String encodeChar = request .getCharacterEncoding()

**30) String getContextPath()**

Returns the portion of the request URI that indicates the context of the request.

Example to test above methods:

String conextPath= request .getContextPath();

Request 🡪<http://localhost:8888/myproject/login?uid=MNRAO&pwd=MNRAO12345>

url 🡪<http://localhost:8888/myproject/login>

uri 🡪http://localhost:8888/myproject

conextPath🡪myproject/login

# Content Type

Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser.

MIME is an internet standard that is used for extending the limited capabilities of email by allowing the insertion of sounds, images and text in a message.

The features provided by MIME to the email services are as given below:

* It supports the non-ASCII characters
* It supports the multiple attachments in a single message
* It supports the attachment which contains executable audio, images and video files etc.
* It supports the unlimited message length.

# List of Content Types

There are many content types. The commonly used content types are given below:

* text/html
* text/plain
* application/msword
* application/vnd.ms-excel
* application/jar
* application/pdf
* application/octet-stream
* application/x-zip
* images/jpeg
* images/png
* images/gif
* audio/mp3
* video/mp4
* video/quicktime etc.

LoginForm.html :

=================

<html>

<body>

<form action="./mylogin" method="GET">

User Id: <input type="text" name="uid">

<br/>

Password: <input type="text" name="pwd" />

<input type="submit" value="Submit" />

</form>

</body>

</html>

**import** java.io.\*;

**import** java.util.Enumeration;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

// Extend LoginServlet class

**publicclass**LoginServlet**extends** HttpServlet {

**publicvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

// Set response content type

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<HTML>" );

out.println("<h1>");

String method = request.getMethod();

out.println("METHOD :"+method);

String userId = request.getParameter("uid");

out.println("User Id : "+userId);

String passwd = request.getParameter("pwd");

out.println("Password :"+passwd);

Enumerationnames = request.getParameterNames();

**while**(names.hasMoreElements())

{

String name = (String) names.nextElement();

out.println("param name : "+name);

}

String queryString = request.getQueryString();

out.println("QUERY String :"+queryString);

**int**len = request.getContentLength();

out.println("Content Length :"+len);

String contentType = request.getContentType();

out.println("Content Type :"+contentType);

String remoteAddr = request.getRemoteAddr();

out.println("Remote Address :"+remoteAddr);

String remoteHost = request.getRemoteHost();

out.println("Remote Host :"+remoteHost);

**int**remotePort = request.getRemotePort();

out.println("Remote Port :"+remotePort);

String remoteUser = request.getRemoteUser();

out.println("Remote User :"+remotePort);

String requestURI = request.getRequestURI();

out.println("Request URI :"+remotePort);

StringBuffer requestURL = request.getRequestURL();

String temp = requestURL.toString();

out.println("Request URL :"+temp);

String servletPath = request.getServletPath();

out.println("Servlet Path :"+servletPath);

String pathInfo = request.getPathInfo();

out.println("Servlet Path Info :"+pathInfo);

String serverName = request.getServerName();

out.println("Server Name :"+serverName);

**int**serverPort = request.getServerPort();

out.println("Server Port :"+serverPort);

String protocol = request.getProtocol();

out.println("Protocol :"+protocol);

out.println("</h1>");

out.println("</HTML>" );

}

}

O/p:

METHOD :GET

User Id : nrit

Password :java

param name : uid

param name : pwd

QUERY String :uid=nrit&pwd=java

Content Length :-1

Content Type :null

Remote Address :127.0.0.1

Remote Host :127.0.0.1

Remote Port :50487

Remote User :50487

Request URI :50487

Request URL :http://localhost:8080/reqmethods/mylogin

Servlet Path :/mylogin

Servlet Path Info :null

Server Name :localhost

Server Port :8080

Protocol :HTTP/1.1

**Servlets with Database:**

**Register Form.**

**register.html:**

<!DOCTYPEhtml>

<html>

<head>

<title>Register</title>

</head>

<body>

<formaction=*"./register"*method=*"post"*>

userid<inputtype=*"text"*name=*"uid"*/><br/>

password <inputtype=*"password"*name=*"pwd"*/><br/>

Retype Password <inputtype=*"password"*name=*"rpwd"*/><br/>

First Name <inputtype=*"text"*name=*"firstname"*><br/>

Last Name <inputtype=*"text"*name=*"lastname"*><br/>

Birth Date <inputtype=*"text"*name=*"birthdate"*><br/>

Gender <inputtype=*"text"*name=*"gender"*><br/>

<inputtype=*"submit"*value=*"Submit"*/>

</form>

</body>

</html>

**Web.xml:**

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*id=*"WebApp\_ID"*version=*"3.1"*>

<display-name>user</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>register</servlet-name>

<servlet-class>com.nrit.user.register.RegisterServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>register </servlet-name>

<url-pattern>/register </url-pattern>

</servlet-mapping>

</web-app>

RegisterServlet.java

**package** com.nrit.user.register;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass**RegisterServlet**extends** HttpServlet {

/\*\*

\*

\*/

**privatestaticfinallong*serialVersionUID*** = 1L;

Connection con = **null**;

PreparedStatement pst = **null**;

@Override

**publicvoid** init() **throws** ServletException {

**try** {

Class.*forName*("oracle.jdbc.driver.OracleDriver");

con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");

con.setAutoCommit(**false**);

pst = con.prepareStatement("insert into track\_user values(?,?,?,?,?,?)");

} **catch** (Exception e) {

e.printStackTrace();

}

}

@Override

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**try**

{

String userId = request.getParameter("uid");

String password = request.getParameter("pwd");

String retypePassword = request.getParameter("rpwd");

String firstName = request.getParameter("firstname");

String lastName = request.getParameter("lastname");

String date = request.getParameter("birthdate");

String gender = request.getParameter("gender");

**if**(!password.equalsIgnoreCase(retypePassword))

{

out.println("<h1>retype not matched</h1>");

**return**;

}

pst.setString(1, userId);

pst.setString(2, password);

pst.setString(3, firstName);

pst.setString(4, lastName);

pst.setString(5, date);

pst.setString(6, gender);

pst.executeUpdate();

out.println("<h1>Successfully Register</h1>");

con.commit();

}

**catch**(Exception e)

{

e.printStackTrace();

out.println("<h1>Server busy</h1>");

**try** {

con.rollback();

} **catch** (SQLException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}

}

@Override

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

// **TODO** Auto-generated method stub

doGet(request, response);

}

@Override

**publicvoid** destroy() {

**try**

{

con.close();

}

**catch**(Exception e)

{

e.printStackTrace();

}

}

}

**Login Check:**

**login.html:**

<!DOCTYPEhtml>

<html>

<head>

<title>Login</title>

</head>

<body>

<formaction=*"./login"*method=*"post"*>

userid<inputtype=*"text"*name=*"uid"*/><br/>

password <inputtype=*"password"*name=*"pwd"*/><br/>

<inputtype=*"submit"*value=*"Submit"*/>

</form>

</body>

</html>

**Web.xml:**

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*id=*"WebApp\_ID"*version=*"3.1"*>

<display-name>userlogin</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>com.nrit.user.login.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login </servlet-name>

<url-pattern>/login </url-pattern>

</servlet-mapping>

</web-app>

LoginServlet.java

**package** com.nrit.user.login;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import**java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass** LoginServlet **extends** HttpServlet {

/\*\*

\*

\*/

**privatestaticfinallong*serialVersionUID*** = 1L;

Connection con = **null**;

PreparedStatementpst = **null**;

@Override

**publicvoid** init() **throws** ServletException {

**try** {

Class.*forName*("oracle.jdbc.driver.OracleDriver");

con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger");

con.setAutoCommit(**false**);

pst = con.prepareStatement("select count(\*) from track\_user where user\_id = ? and user\_passwd = ?");

} **catch** (Exception e) {

e.printStackTrace();

}

}

@Override

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**try** {

String userId = request.getParameter("uid");

String password = request.getParameter("pwd");

pst.setString(1, userId);

pst.setString(2, password);

ResultSet rs = pst.executeQuery();

**if** (rs.next()) {

**int**count = rs.getInt(1);

**if** (count == 1) {

out.println("<h1>Valid User </h1>");

} **else** {

out.println("<h1>In Valid User </h1>");

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

@Override

**protectedvoid** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

doGet(req, resp);

}

@Override

**publicvoid** destroy() {

**try** {

con.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**ServletConfig Interface**

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

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.

Main purpose is to read external configuration parameters, which are changed dynamically.

External configuration parameters can be passed through the web.xml file.

**Advantage of ServletConfig**

The main advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

**Methods of ServletConfig interface**

1. **public String getInitParameter(String name):**

Returns the parameter value for the specified parameter name.

1. **public Enumeration getInitParameterNames():**

Returns an enumeration of all the initialization parameter names.

1. **public String getServletName():**

Returns the name of the servlet.

1. **public ServletContext getServletContext():**

Returns an object of ServletContext.

**Obtaining object of ServletConfig**

**getServletConfig() 🡪 is a method** of Servlet interface, which returns the object of ServletConfig.

Signature :

Public ServletConfig getServletConfig();

eg:

ServletConfig config=getServletConfig();

**Passing initialization parameter to a servlet through web.xml**

<web-app>

<servlet>

     .....

    <init-param>

      <param-name>parametername</param-name>

      <param-value>parametervalue</param-value>

     </init-param>

     ......

   </servlet>

</web-app>

**Note : These parameters are servlet specific**

**Example of ServletConfig to get initialization parameter**

Web.xml:

================

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*id=*"WebApp\_ID"*version=*"3.1"*>

<display-name>user</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>register</servlet-name>

<servlet-class>com.nrit.user.register.RegisterServlet</servlet-class>

<init-param>

<param-name>driver</param-name>

<param-value>oracle.jdbc.driver.OracleDriver</param-value>

</init-param>

<init-param>

<param-name>dbhost</param-name>

<param-value>localhost</param-value>

</init-param>

<init-param>

<param-name>port</param-name>

<param-value>1521</param-value>

</init-param>

<init-param>

<param-name>sid</param-name>

<param-value>orcl</param-value>

</init-param>

<init-param>

<param-name>uid</param-name>

<param-value>scott</param-value>

</init-param>

<init-param>

<param-name>pwd</param-name>

<param-value>tiger</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>register </servlet-name>

<url-pattern>/register </url-pattern>

</servlet-mapping>

</web-app>

**Servlet to read configuration parameters and connect to database.**

**package** com.nrit.user.register;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass** RegisterServlet **extends** HttpServlet {

/\*\*

\*

\*/

**privatestaticfinallong*serialVersionUID*** = 1L;

Connection con = **null**;

PreparedStatement pst = **null**;

@Override

**publicvoid** init() **throws** ServletException {

**try** {

ServletConfig config = getServletConfig();

String jdbcDriver = config.getInitParameter("driver");

String dbHost = config.getInitParameter("dbhost");

String dbPortNum = config.getInitParameter("port");

String dbsid = config.getInitParameter("sid");

String userId = config.getInitParameter("uid");

String password = config.getInitParameter("pwd");

Class.*forName*(jdbcDriver);

con = DriverManager.*getConnection*("jdbc:oracle:thin:@"+dbHost+":"+dbPortNum+":"+dbsid, userId, password);

con.setAutoCommit(**false**);

pst = con.prepareStatement("insert into track\_user values(?,?,?,?,?,?)");

} **catch** (Exception e) {

e.printStackTrace();

}

}

@Override

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**try**

{

String userId = request.getParameter("uid");

String password = request.getParameter("pwd");

String retypePassword = request.getParameter("rpwd");

String firstName = request.getParameter("firstname");

String lastName = request.getParameter("lastname");

String date = request.getParameter("birthdate");

String gender = request.getParameter("gender");

**if**(!password.equalsIgnoreCase(retypePassword))

{

out.println("<h1>retype not matched</h1>");

**return**;

}

pst.setString(1, userId);

pst.setString(2, password);

pst.setString(3, firstName);

pst.setString(4, lastName);

pst.setString(5, date);

pst.setString(6, gender);

pst.executeUpdate();

out.println("<h1>Successfully Register</h1>");

con.commit();

}

**catch**(Exception e)

{

e.printStackTrace();

out.println("<h1>Server busy</h1>");

**try** {

con.rollback();

} **catch** (SQLException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}

}

@Override

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

// **TODO** Auto-generated method stub

doGet(request, response);

}

}

**ServletContext Interface**

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get context 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.

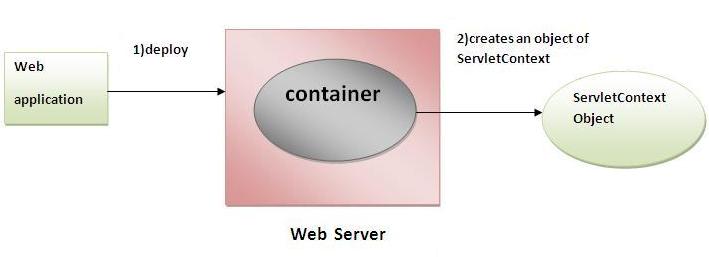
**Advantage of ServletContext**

**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.

**Usage of ServletContext Interface**

There can be a lot of usage of ServletContext object. Some of them are as follows:

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 of ServletContext interface**

* 1. **public String getInitParameter(String name):**

Returns the parameter value for the specified parameter name.

* 1. **public Enumeration getInitParameterNames():**

Returns the names of the context's initialization parameters.

* 1. **public void setAttribute(String name,Object object):**

sets the given object in the application scope.

* 1. **public Object getAttribute(String name):**

Returns the attribute for the specified name.

* 1. **public Enumeration getInitParameterNames():**

Returns the names of the context's initialization parameters as an Enumeration of String objects.

* 1. **public void removeAttribute(String name):**

Removes the attribute with the given name from the servlet context.

**Obtaining the object of ServletContext interface:**

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

Signature:

Public ServletContext getServletContext() ;

1. ServletContext  object from ServletConfig object.

ServletConfig config=getServletConfig();

ServletContext applicationContext= config.getServletContext();

1. //Another convenient way to get the ServletContext object

ServletContext applicationContext =getServletContext();

**Passing initial parameters through web.xml**

<web-app>

  ......

  <context-param>

     <param-name>parametername</param-name>

    <param-value>parametervalue</param-value>

  </context-param>

 ......

</web-app>

**Example to share context initialization parameters by LoginServlet and RegisterServlet:**

**Index.html**

<!DOCTYPEhtml>

<html>

<head>

<title>Register</title>

</head>

<body>

<formaction=*"./register"*method=*"post"*>

userid<inputtype=*"text"*name=*"uid"*/><br/>

password <inputtype=*"password"*name=*"pwd"*/><br/>

Retype Password <inputtype=*"password"*name=*"rpwd"*/><br/>

First Name <inputtype=*"text"*name=*"firstname"*><br/>

Last Name <inputtype=*"text"*name=*"lastname"*><br/>

Birth Date <inputtype=*"text"*name=*"birthdate"*><br/>

Gender <inputtype=*"text"*name=*"gender"*><br/>

<inputtype=*"submit"*value=*"Submit"*/>

</form>

</body>

</html>

**Login.html**

<!DOCTYPEhtml>

<html>

<head>

<title>Login</title>

</head>

<body>

<formaction=*"./login"*method=*"post"*>

userid<inputtype=*"text"*name=*"uid"*/><br/>

password <inputtype=*"password"*name=*"pwd"*/><br/>

<inputtype=*"submit"*value=*"Submit"*/>

</form>

</body>

</html>

**Web.xml:**

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appxmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*id=*"WebApp\_ID"*version=*"3.1"*>

<display-name>user</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<context-param>

<param-name>driver</param-name>

<param-value>oracle.jdbc.driver.OracleDriver</param-value>

</context-param>

<context-param>

<param-name>dbhost</param-name>

<param-value>localhost</param-value>

</context-param>

<context-param>

<param-name>port</param-name>

<param-value>1521</param-value>

</context-param>

<context-param>

<param-name>sid</param-name>

<param-value>orcl</param-value>

</context-param>

<context-param>

<param-name>uid</param-name>

<param-value>scott</param-value>

</context-param>

<context-param>

<param-name>pwd</param-name>

<param-value>tiger</param-value>

</context-param>

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>com.nrit.user.register.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login </servlet-name>

<url-pattern>/login </url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>register</servlet-name>

<servlet-class>com.nrit.user.register.RegisterServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>register </servlet-name>

<url-pattern>/register </url-pattern>

</servlet-mapping>

</web-app>

**RegisterServlet.java:**

**package** com.nrit.user.register;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.SQLException;

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass**RegisterServlet**extends** HttpServlet {

/\*\*

\*

\*/

**privatestaticfinallong*serialVersionUID*** = 1L;

Connection con = **null**;

PreparedStatement pst = **null**;

@Override

**publicvoid** init() **throws** ServletException {

**try** {

ServletContext conext = getServletContext();

String jdbcDriver = conext.getInitParameter("driver");

String dbHost = conext.getInitParameter("dbhost");

String dbPortNum = conext.getInitParameter("port");

String dbsid = conext.getInitParameter("sid");

String userId = conext.getInitParameter("uid");

String password = conext.getInitParameter("pwd");

Class.*forName*(jdbcDriver);

con = DriverManager.*getConnection*("jdbc:oracle:thin:@"+dbHost+":"+dbPortNum+":"+dbsid, userId, password);

con.setAutoCommit(**false**);

pst = con.prepareStatement("insert into track\_user values(?,?,?,?,?,?)");

} **catch** (Exception e) {

e.printStackTrace();

}

}

@Override

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**try**

{

String userId = request.getParameter("uid");

String password = request.getParameter("pwd");

String retypePassword = request.getParameter("rpwd");

String firstName = request.getParameter("firstname");

String lastName = request.getParameter("lastname");

String date = request.getParameter("birthdate");

String gender = request.getParameter("gender");

**if**(!password.equalsIgnoreCase(retypePassword))

{

out.println("<h1>retype not matched</h1>");

**return**;

}

pst.setString(1, userId);

pst.setString(2, password);

pst.setString(3, firstName);

pst.setString(4, lastName);

pst.setString(5, date);

pst.setString(6, gender);

pst.executeUpdate();

out.println("<h1>Successfully Register</h1>");

con.commit();

}

**catch**(Exception e)

{

e.printStackTrace();

out.println("<h1>Server busy</h1>");

**try** {

con.rollback();

} **catch** (SQLException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}

}

@Override

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

// **TODO** Auto-generated method stub

doGet(request, response);

}

}

**LoginServlet.java**

**package** com.nrit.user.register;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass** LoginServlet **extends** HttpServlet {

/\*\*

\*

\*/

**privatestaticfinallong*serialVersionUID*** = 1L;

Connection con = **null**;

PreparedStatement pst = **null**;

@Override

**publicvoid** init() **throws** ServletException {

**try** {

ServletContext conext = getServletContext();

String jdbcDriver = conext.getInitParameter("driver");

String dbHost = conext.getInitParameter("dbhost");

String dbPortNum = conext.getInitParameter("port");

String dbsid = conext.getInitParameter("sid");

String userId = conext.getInitParameter("uid");

String password = conext.getInitParameter("pwd");

Class.*forName*(jdbcDriver);

con = DriverManager.*getConnection*("jdbc:oracle:thin:@"+dbHost+":"+dbPortNum+":"+dbsid, userId, password);

con.setAutoCommit(**false**);

pst = con.prepareStatement("select count(\*) from track\_user where user\_id = ? and user\_passwd = ?");

} **catch** (Exception e) {

e.printStackTrace();

}

}

@Override

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**try** {

String userId = request.getParameter("uid");

String password = request.getParameter("pwd");

pst.setString(1, userId);

pst.setString(2, password);

ResultSet rs = pst.executeQuery();

**if** (rs.next()) {

**int**count = rs.getInt(1);

**if** (count == 1) {

out.println("<h1>Valid User </h1>");

} **else** {

out.println("<h1>In Valid User </h1>");

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

@Override

**protectedvoid** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

doGet(req, resp);

}

@Override

**publicvoid** destroy() {

**try** {

con.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**Servlet to read all context initial parameters**

**import** java.io.\*;

**import** java.util.Enumeration;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**ConextInitAllParamsServlet**extends** HttpServlet

{

**publicvoid** doGet(HttpServletRequest req,HttpServletResponse res)

**throws** ServletException,IOException

{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

ServletContext context=getServletContext();

Enumeration<String>e=context.getInitParameterNames();

String str="";

**while**(e.hasMoreElements())

{

str=e.nextElement();

out.print("<br> "+context.getInitParameter(str));

}

}

}

**Servlet Collaboration:**

The Servlet collaboration is all about sharing information among the servlets. Collaborating servlets is to pass the common information that is to be shared directly by one servlet to another through various invocations of the methods. To perform these operations, each servlet need to know the other servlet with which it is collaborated.

Here are several ways to communicate with one another:

* Using RequestDispatchers *include()* and *forward()* method;
* Using HttpServletResponse *sendRedirect()* method;
* Using ServletContext *setAttribute()* and *getAttribute()* methods;
* Using Java's system-wide *Properties* list;
* Using singleton class object.

1. **Using RequestDispatcher**
2. **Using SendRedirect**

**Servlet RequestDispatcher :**

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

1. **public void forward(ServletRequest request,ServletResponse response) throws**

**ServletException, java.io.IOException**

Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

1. **public void include(ServletRequest request,ServletResponse response)throws**

**ServletException,java.io.IOException**

Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

## RequestDispatcher forward()

how to pass data between two servlets (one-to-one) and for this Servlet API comes with **javax.servlet.RequestDispatcher**  interface.

**When to use RequestDispatcher interface?**

1. To **include** the response (output) of one Servlet into another (that is, client gets the response of both Servlets).
2. To **forward** the client request to another Servlet to honour (that is, client calls a Servlet but response to client is given by another Servlet).

**How to obtain an object of RequestDispatcher interface?**

There are two ways

1. using **ServletRequest** object and **ServletContext** object.

**getRequestDispatcher(String path)** method returns an object of **RequestDispatcher**

this method defined in both interfaces of **ServletRequest** and **ServletContext**.

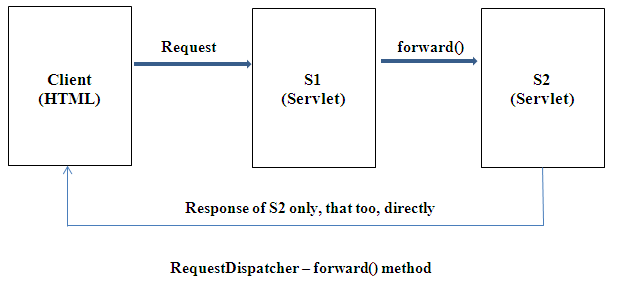
**getRequestDispatcher(String path)** method from **ServletRequest interface**

**RequestDispatcher getRequestDispatcher(String path):**

Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.

A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

**RequestDispatcher forward() Method**



**When to use RequestDispatcher forward() method?**

Client calls a Servlet for some information. But the Servlet cannot honour the request because it is incapable. But it knows that another Servlet exists which can do the job of the client. Then how the first Servlet (called by the client) can send (forward) the request to another Servlet. Here, **forward()** method of **RequestDispatcher** is used.

That is, client calls one **FirstServlet** but response goes from another **SecondServlet**. but the client is not aware of all this transactions happening on the server.

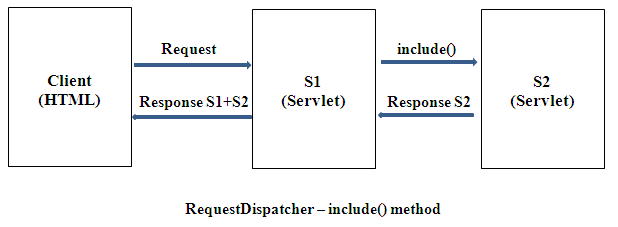
Here, response of**SecondServlet** goes to client but never of **FirstServlet**.

Where as , in [include()](http://way2java.com/servlets/requestdispatcher-include-example/) method both **FirstServlet** and **SecondServlet**goes.

**RequestDispatcher include() Method**

**When to use RequestDispatcher include() method?**

To place (include) the response content (ouput) of one servlet into another servlet’s response. That is here, client gets the response of both servlets. Or simply to say, to put the output of one servlet into another.



Observe the figure. Client request (or calls) the servlet **FirstServlet** .

**FirstServlet**  sends client data to **SecondServlet** with **include()** method.

Response of **SecondServlet**  comes to **FirstServlet** .

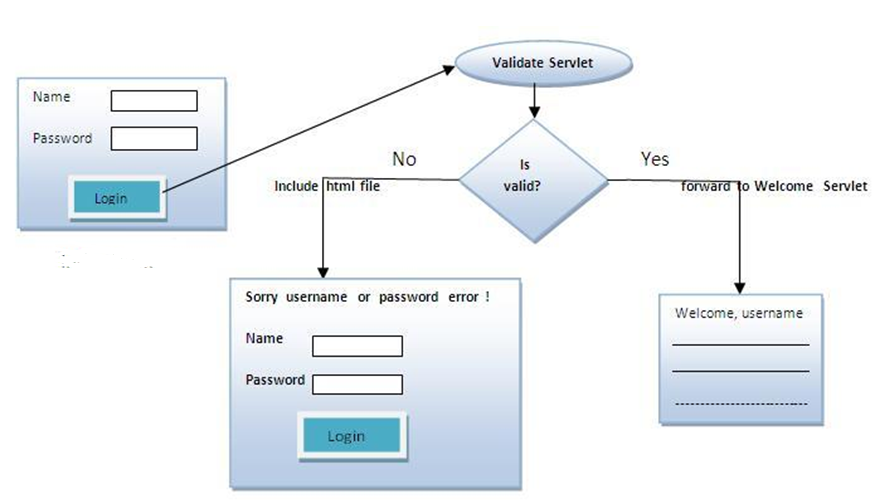
**FirstServlet** sends to client its response (of **FirstServlet** ) and also of **SecondServlet**

**Example of RequestDispatcher interface:**

In this example, we are validating the password entered by the user. If password is valid, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter.

In this example, we have created following files:

LoginForm.html:



**LoginForm.html:**

<formaction=*"./mylogin"*method=*"post"*>

Name:<inputtype=*"text"*name=*"uid"*/><br/>

Password:<inputtype=*"password"*name=*"pwd"*/><br/>

<inputtype=*"submit"*value=*"login"*/>

</form>

LoginServlet.java

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**LoginServlet**extends** HttpServlet

{

**publicvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String userName=request.getParameter("uid");

String userPassword=request.getParameter("pwd");

**if**(userName.equals("MNRAO")&&userPassword.equals("java"))

{

RequestDispatcher welcomeServletRD

=request.getRequestDispatcher("mywcservlet");

welcomeServletRD.forward(request, response);

}

**else**

{

out.print("Sorry UserName or Password Error!");

RequestDispatcher

loginForm=request.getRequestDispatcher("/LoginForm.html");

loginForm.include(request, response);

}

}

}

**WelcomeServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**WelcomeServlet**extends** HttpServlet

{

**publicvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String userName=request.getParameter("uid");

out.print("Welcome "+userName);

}

}

Web.xml:

<web-app>

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>LoginServlet</servlet-class>

</servlet>

<servlet>

<servlet-name>wcs</servlet-name>

<servlet-class>WelcomeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/mylogin</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>wcs</servlet-name>

<url-pattern>/mywcservlet</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>LoginForm.html</welcome-file>

</welcome-file-list>

</web-app>

## Difference between include() and forward()methods

|  |  |  |
| --- | --- | --- |
| **Property** | **include() Method** | **forward() Method** |
| **Result** | Includes another file in our current file. | Will forward the client request to the forwarding page. |
| **Merge of response** | Response of **FirstServlet** and **SecondServlet** are merged and sent to client (as if a single response). This way, the Programmer can achieve "server side includes". | No merge of response. Only **SecondServlet** response will go to the client. |
| **Retaining execution control** | Shifted temporarily from **FirstServlet** to **SecondServlet**. It works like a general simple method call. | Shifted permanently from **FirstServlet** to **SecondServlet.Control never return back to FirstServlet.** |
| **Control coming back** | Execution control comes back to **FirstServlet** after executing **SecondServlet** for further processing of **FirstServlet** after include() statement from where the execution control shifted. | Once shifted, the control never returns to **FirstServlet**. It is permanent shifting. |
| **Response placement** | Response of **SecondServlet** is placed in **FirstServlet**. | Response of **SecondServlet** is not placed in **FirstServlet**. |
| **Client receives** | Response of **FirstServlet** and **SecondServlet** is received by client. | Only response of **SecondServlet** is received by client. |
| **Control returned** | After executing **SecondServlet**, control returned to **FirstServlet**. | After executing **SecondServlet**, control returned to client. |
| **Extra activity** | Once control is returned to **FirstServlet** from **SecondServlet**, any activity can be done on the server like calling another servlet with another RequestDispatcher object. | Once control returned to client, no activity can be done on **FirstServlet** or **SecondServlet**. |
| **Usage** | Used by Programmer when the output of both servlets **FirstServlet**  and **SecondServlet** is required. | Used only **SecondServlet** response is required. |
| **Speed of delivery to client** | Comparatively slower. | Faster. |
| **Access** | **SecondServlet**  has access to the request and response objects of **FirstServlet**  , but limitations exist. **SecondServlet**  cannot set headers and also cannot call any other method like setCookie etc. affecting the response headers. That is, **SecondServlet**  cannot attempt to change the HTTP headers or response status code etc. and performing any activities like this is simply ignored. | Here also **SecondServlet** cannot alter as response is delivered on **FirstServlet** URL. |
| **out.println** | Output of **FirstServlet**  out.println() statements go to client. | Output of **FirstServlet**  out.println() statements never go to client. |
| **Client response** | Client receives the response from the same servlet which he requested. | Client actually receives the response from a different servlet (not known to client). |
| When to Use | Used when static information is to be included. | Used when dynamic information is to be included |

**Servlet SendRedirect :**

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url to make another request. So, it can work inside and outside the server.

Signature :

public void sendRedirect(String URL)throws IOException;

eg:

response.sendRedirect("http://www.google.com");

**example for sendRedirect method :**

in this, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, that is why we can our request to anywhere. We can send our request within and outside the server.

SearchForm.html:

<html>

<head><title>sendRedirect example</title></head>

<body>

<formaction=*"./MySearcher"*>

<inputtype=*"text"*name=*"name"*>

<inputtype=*"submit"*value=*"search"*>

</form>

</body>

</html>

**SearchServlet.java:**

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass**SearchServlet**extends** HttpServlet

{

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

String searchText=request.getParameter("name");

response.sendRedirect("https://www.google.co.in/#q="+searchText);

}

}

<web-app>

<servlet>

<servlet-name>search</servlet-name>

<servlet-class>SearchServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>search</servlet-name>

<url-pattern>/MySearcher</url-pattern>

</servlet-mapping>

</web-app>

RedirectServlet.java:

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass** RedirectServlet **extends** HttpServlet

{

**publicvoid** doGet(HttpServletRequest request,HttpServletResponse response)

**throws** ServletException,IOException

{

response.setContentType("text/html");

PrintWriter out=response.getWriter();

response.sendRedirect("http://www.google.com");

out.close();

}

}

###### **Difference between forward () and sendRedirect ()**

|  |  |  |
| --- | --- | --- |
| **Property** | **forward()** | **sendRedirect()** |
| Defined interface | Defined in RequestDispatcher | Defined in HttpServletResponse |
| Signature | void forward(ServletRequest request, ServletResponse response) | void sendRedirect(String url) |
| Client awareness | Client is not aware of that he is getting response from a different Servlet as the URL will not change in client’s browser. | Client can know easily as the URL (from where he is getting response) changes in the client browser’s prompt. |
| Execution control | Execution control changes to another Servlet on the same server without client being informed that altogether a different Servlet is going to process his request. | Control changes to client |
| Where is what? | Forward is done on server side without client’s knowledge. | Browser issues a new request on the URL that is redirected (sent as parameter) by the server and client can easily aware of. |
| Where happens | Everything happens on server side within the Web container and client is not involved. | sendRedirect() causes the Web container to return to the client’s browser. Client inturn can redirect to different servers or domains. |
| Speed | Faster as forward runs on server-side entirely and no extra network trip (to client) is required. | Due to extra round trip between browser-server-browser (running on client as well as on server side), it is slower. |
| Content | forward() sends the same request to another resource of the same Web application. | Calls another page with a different request URL but not on the same request. |
| Usage | forward() reuses the current request object | Redirects create a new request object; consequently looses the original request with all its parameters and attributes. |
| Transfer of parameters | Original request and response objects transfer data coming from client along with additional information set with setAttribute() method (if any) to another resource request and response objects. | Redirect action sends header back to the client. Browser uses the URL contained in the header to call a new resource. As client initiates a new request, the original request and response objects are lost and fresh ones are to be created. |
| Transfer control | Internally, the Servlet container transfers control of client request to another Servlet (or JSP). | This method sends the HTTP response to client browser to allow the client to send another request with a different URL. Usage of this method is equivalent to opening a new browser window and typing the URL. |
| What is sent? | Server sends the response (information required) to the client. | With this method, server sends a URL to the client. |
| Visual difference | Client cannot see the address of new resource that honours the client request in the address bar of the browser. | Client can see the new redirected address in address bar. |
| Examples | Calling another resource to process the data like validation of Login data. | Calling advertisements on the Web page or payment gateways. |
| Task separation | With this method, the responsibility of handling the client request can be distributed between many Servlets (or JSPs). | Used to transfer control altogether to a different domain. Also used to write separation of tasks. |
| Back and Forward buttons | As everything happens on server with forward, nothing is stored on browser history. So, Back and Forward buttons will not work. | As client makes new request and updated in browser history, back and forward buttons work. |
| URL | Use only relative URLs with forward(). | Use absolute URLs. |
| MVC to hide | Useful in MVC design pattern to hide JSP/Servlet from direct access. | Once redirected to client, server looses control. |
| Which one to prefer? | If you would like to forward the client request to a new resource on the same server for further process, prefer forward() where data of the original resource can be passed to the new resource. | If you would like to transfer the control to a new server or domain where client treats as a new task, prefer sendRedirect(). If the data of the original resource (which client requested) is needed in the new resource, store them in Session object and reuse. |

**Some more points to notice of forward vs sendRedirect.**

1. Session is not lost in both cases.
2. The above differences are applicable to Servlets and JSPs. In Servlets, these methods are used in service() and in JSP used in scriptlets.
3. In frameworks like Struts, the Controller can decide, at the end of request processing, which one to use of either forward or redirect operation.
4. The Controller also can decide with forward() method, to what resource the forward should be made, depending on different conditions of client request requirements.

**Which one is preferred?**

Just depends on the scenario.

If you would like to forward the client request to a new resource on the same server for further process, prefer forward() where data of the original resource can be passed to the new resource.

**Servlet Container**

It provides the runtime environment for JavaEE (j2ee) applications. The client/user can request only a static WebPages from the server. If the user wants to read the web pages as per input then the servlet container is used in java.

The servlet container is used in java for dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.



**Servlet Container States**

The servlet container is the part of web server which can be run in a separate process. We can classify the servlet container states in three types:

* **Standalone:** It is typical Java-based servers in which the servlet container and the web servers are the integral part of a single program. For example:- Tomcat running by itself
* **In-process:** It is separated from the web server, because a different program is runs within the address space of the main server as a plug-in. For example:- Tomcat running inside the JBoss.
* **Out-of-process:** The web server and servlet container are different programs which are run in a different process. For performing the communications between them, web server uses the plug-in provided by the servlet container.

**The Servlet Container performs many operations that are given below:**

* Life Cycle Management
* Multithreaded support
* Object Pooling
* Security etc.

**Servlet Attributes:**

**Scope of the Servlet attributes:**

1. request scope
2. session scope
3. application scope

The servlet programmer can pass informations from one servlet to another using attributes

**Attribute specific methods of ServletRequest, HttpSession and ServletContext interface**

There are following 4 attribute specific methods.

1. **public void setAttribute(String name,Object object):**

sets the given object in the application scope.

1. **public Object getAttribute(String name):**

Returns the attribute for the specified name.

1. **public Enumeration getInitParameterNames():**

Returns the names of the context's initialization parameters as an Enumeration of String objects.

1. **public void removeAttribute(String name):**

Removes the attribute with the given name from the servlet context.

**Example :**

**ServletContext to set and get attribute**

1. **Servlet to store attributes into context**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**ContextAttribTestServlet1**extends** HttpServlet

{

**publicvoid** doGet(HttpServletRequest req,HttpServletResponse res) **throws** IOException

{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

**try**

{

ServletContext context=getServletContext();

context.setAttribute("name","MNRAO");

out.println("Welcome to first servlet");

out.close();

}

**catch**(Exception e)

{

out.println(e);

}

}

}

1. **Servlet to read attributes from context**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**ContextAttribTestServlet2**extends** HttpServlet

{

**publicvoid** doGet(HttpServletRequest req,HttpServletResponse res) **throws** IOException

{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

**try**

{

ServletContext context=getServletContext();

String myName=(String)context.getAttribute("name");

out.println("Welcome to "+myName);

out.close();

}

**catch**(Exception e)

{

out.println(e);

}

}

}

**Web.xml:**

<web-app>

<servlet>

<servlet-name>ca1</servlet-name>

<servlet-class>ContextAttribTestServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ca1</servlet-name>

<url-pattern>/contextattrib1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>ca2</servlet-name>

<servlet-class>ContextAttribTestServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ca2</servlet-name>

<url-pattern>/contextattrib2</url-pattern>

</servlet-mapping>

</web-app>

**Difference between ServletConfig and ServletContext**

|  |
| --- |
| The servletconfig object refers to the single servlet whereas servletcontext object refers to the whole web application.  **Servlets Session Tracking** |

A session refers to the entire interaction between a client and a server from the time of the client’s first request, which generally begins the session, to the time the session is terminated.

The session could be terminated by the client’s request, or the server could automatically close it after a certain period of time.

**What is Session Tracking and Why ?**

**Session Tracking** is a way to maintain state (data) of an user.

It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

**HTTP** is used by client to request the server for some information over the **Web**. The client establishes a connection with the server before sending a request. Over this connection only, the server returns the response to client. When the response is delivered, the connection between client and server is **destroyed**. By chance, if the same client would like to contact the same server again, the client should establish altogether a **new connection**. To put in a simple way, if the client would like to request the same server 100 times, the client should establish a new connection 100 times as the server closes the connection for each request when the response is delivered.

**What is Stateless Protocol?**

Once response is delivered, the server simply forgets the client. If the client wants the same data, it must again establish a new connection, send request and receive response and the same earlier process is to be repeated. For this reason, **HTTP protocol is treated as stateless protocol**.

each request is treated as a separate and independent transaction that is not related at all with any previous requests.

HTTP Protocol, which is once the most advantageous being connectionless and stateless, now became a minus point over the time. With this nature of HTTP Protocol, it is not possible to develop **e-commerce** applications where on a single connection, lot of data should be transferred between Client and Server multiple times. To overcome this in Servlets and JSP, there comes session, session management, session tracking etc. in Servlets.

**What is a Session in Servlets?**

Session begins when the client **logs in** to a Web site and ends when the user **logs out**.

**Session** is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response.

The **interactive time** between client and server on a single connection is known as a session.

(Or)

The **period of time** between connection establishment and connection closing between client and server is known as a session.

(Or)

Session starts when the connection is **established** at both ends (client and server) and terminates when the connection (communication) is **ended** (closed by server).

A connection is **well maintained** by the Servlet container while the client and server are conversing back and forth in a session (with some duration).

**What is Session Tracking or Session Management?**

It is used to recognize the particular user.

A session includes a **lot of interactions**, where data will be exchanged, between client and server, of course on a single connection. Once the server accepts the client connection, the client and server talk together and keep with them lot of data exchanged that includes commits, questions, answers and alike. **Keeping the data of session intact (preserve) so that the data can be reused later is known as session tracking or session management.**  
A session can temporarily store information related to the activities of the user while logged in. A servlet should be capable to store temporary information pertaining to the activities of the user in a session.

**how to maintain session with HTTP protocol nature of stateless and connectionless?**

There are four ways to maintain session tracking. They are

1. Using Hidden fields
2. Using URL Rewriting
3. Using HttpSession interface
4. Using Cookies

The first two ways are almost discarded by the programming world as they increase lot of **network traffic** and **processing load** of each request on the Web server.

**Session Tracking by using HttpSession interface:**

Container creates a session id for each user. The container uses this id to identify the particular user

Servlet API provides Session management through HttpSession interface. We can get session from HttpServletRequest object using following methods.

HttpSession allows us to set objects as attributes that can be retrieved in future requests.

1. **HttpSession getSession()** –

This method always returns a HttpSession object. It returns the session object attached with the request, if the request has no session attached, then it creates a new session and return it.

1. **HttpSession getSession(boolean flag)** –

This method returns HttpSession object if request has session else it returns null.

**Methods of HttpSession interface**

1. **public String getId():**

Returns a string containing the unique identifier value.

1. **public long getCreationTime():**

Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

1. **public long getLastAccessedTime():**

Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

1. **int getMaxInactiveInterval():**

Returns maximum inactive interval time in seconds as an int value. Inactive interval indicates the period which the client does not do any activity on the site like adding or deleting etc. That is, this much time in seconds, the server does not close the session even if the client does not interact with the site.

1. **void setMaxInactiveInterval(int seconds):**

The inactive interval can be set by the Programmer with this method.

1. **public void invalidate():**

Invalidates this session then unbinds any objects bound to it.

1. **void setAttribute(String name, Object value):**

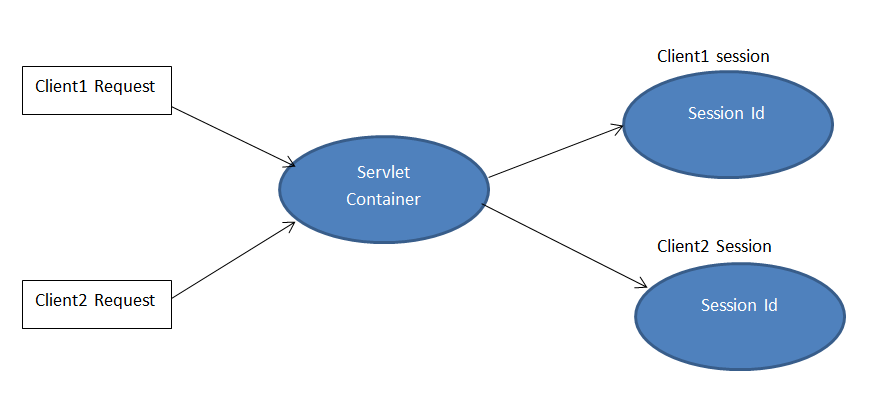
Binds an object to this session, using the name str. If an object of the same name is already bound to the session, the object is replaced. str is treated as the key and value as obj. The setAttribute() method takes always key/value pairs where key is always a string and value can be an object of any Java class. With this method, the Programmer can store session data with the session object.

1. **Object getAttribute(String name):**

Returns the object attached to the session on the name str which is set earlier with setAttribute() method.

1. **void removeAttribute(String name):**

Removes the object attached with session object with the name str. If the session does not have an object bound with the specified name, this method does nothing.



# Servlet HttpSession Login and Logout Example:

We can bind the objects on HttpSession instance and get the objects by using setAttribute and getAttribute methods.

Real world login and logout application without using database code

**Index.html:**

<!DOCTYPEhtml>

<html>

<head>

<metacharset=*"ISO-8859-1"*>

<title>Servlet Login Example</title>

</head>

<body>

<h1>Login App using HttpSession</h1>

<ahref=*"Login.html"*>Login</a>|

<ahref=*"logout"*>Logout</a>|

<ahref=*"profile"*>Profile</a>

</body>

</html>

**WebLink.html:**

<!DOCTYPEhtml>

<ahref=*"Login.html"*>Login</a> |

<ahref=*"logout"*>Logout</a> |

<ahref=*"profile"*>Profile</a>

<hr>

**Login.html:**

<formaction=*"login"*method=*"post"*>

Name:<inputtype=*"text"*name=*"uid"*><br>

Password:<inputtype=*"password"*name=*"pwd"*><br>

<inputtype=*"submit"*value=*"login"*>

</form>

**Web.xml:**

**<web-app>**

**<servlet>**

**<servlet-name>login</servlet-name>**

**<servlet-class>LoginServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>login</servlet-name>**

**<url-pattern>/login</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>logout</servlet-name>**

**<servlet-class>LogoutServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>logout</servlet-name>**

**<url-pattern>/logout</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>profile</servlet-name>**

**<servlet-class>ProfileServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>profile</servlet-name>**

**<url-pattern>/profile</url-pattern>**

**</servlet-mapping>**

**<welcome-file-list>**

**<welcome-file>index.html</welcome-file>**

**</welcome-file-list>**

**</web-app>**

**LoginServlet.java:**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.RequestDispatcher;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

**publicclass**LoginServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out=response.getWriter();

RequestDispatcher linkRequestDispatcher = request.getRequestDispatcher("WebLink.html");

linkRequestDispatcher.include(request, response);

String userName=request.getParameter("uid");

String userPassword=request.getParameter("pwd");

**if**(userName.equals("MNRAO")&&userPassword.equals("java"))

{

out.print("Welcome, "+userName);

HttpSession session=request.getSession();

session.setAttribute("uid",userName);

}

**else**

{

out.print("Sorry, username or password error!");

RequestDispatcher loginRequestDispatcher = request.getRequestDispatcher("Login.html");

loginRequestDispatcher.include(request, response);

}

out.close();

}

}

**LogoutServlet.java:**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.RequestDispatcher;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

**publicclass**LogoutServlet**extends** HttpServlet

{

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out=response.getWriter();

RequestDispatcher loginRequestDispatcher=request.getRequestDispatcher("WebLink.html");

loginRequestDispatcher.include(request, response);

HttpSession session=request.getSession();

session.invalidate();

out.print("You are successfully logged out!");

out.close();

}

}

ProfileServlet.java:

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.RequestDispatcher;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

**publicclass**ProfileServlet**extends** HttpServlet

{

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out=response.getWriter();

RequestDispatcher linkRequestDispatcher=request.getRequestDispatcher("WebLink.html");

linkRequestDispatcher.include(request, response);

HttpSession session=request.getSession(**false**);

**if**(session!=**null**)

{

String userName=(String)session.getAttribute("uid");

out.print("Hello, "+userName+" Welcome to Profile");

}

**else**

{

out.print("Please login first");

RequestDispatcher loginRequestDispatcher=request.getRequestDispatcher("Login.html");

loginRequestDispatcher.include(request, response);

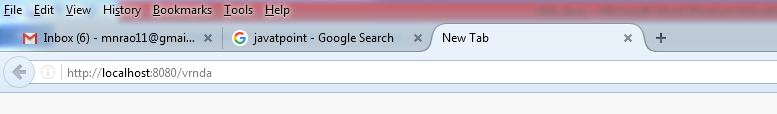
}

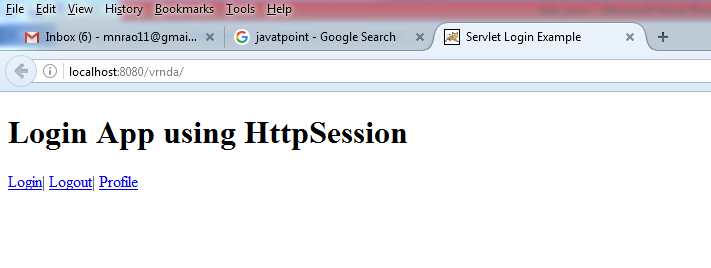
out.close();

}

}

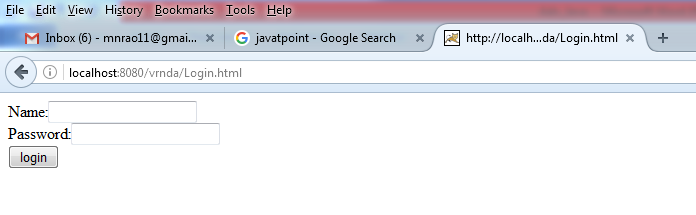
http://localhost:8080/vrnda:



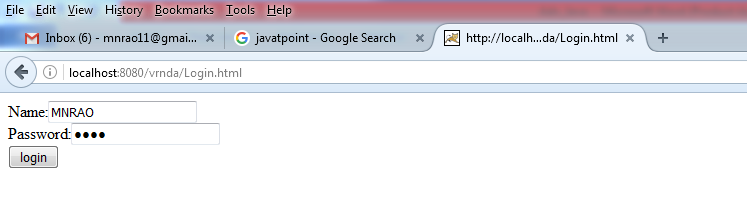


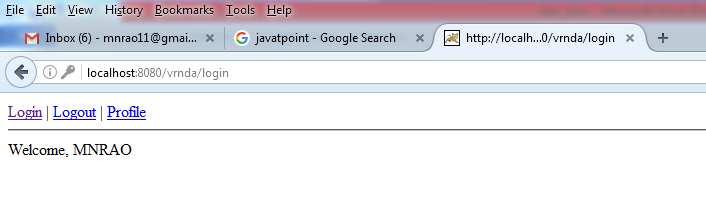
Here :

Click on “Login”

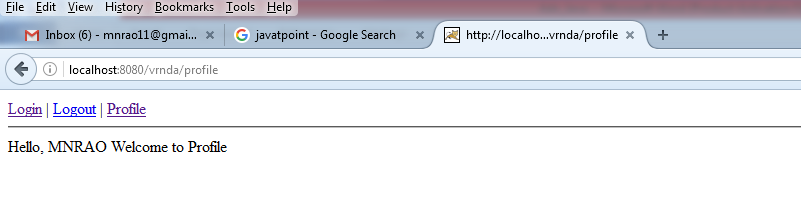


File the form and submit click on “login”

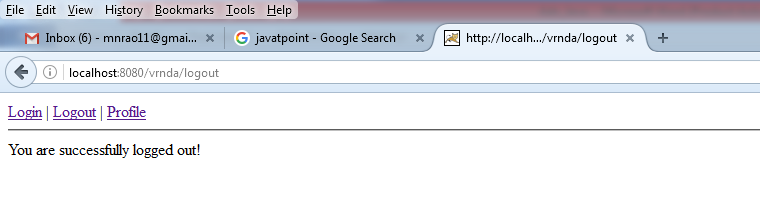




Click on “Profile”



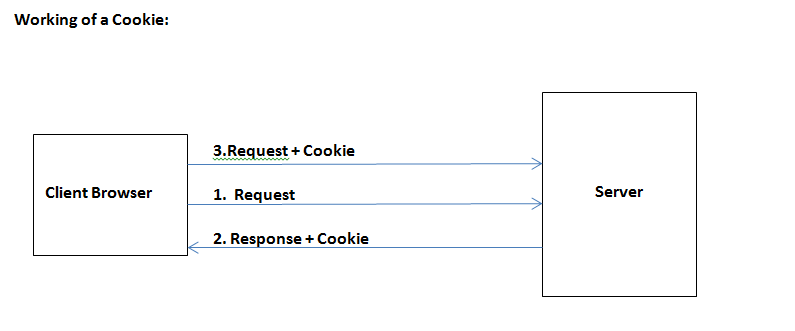
Click on Logout:



**Cookies in Servlet:**

A **cookie** is a small piece of information that is persisted between the multiple requests of the client.

**Servlet Cookie** stores small strings of data. Cookies stay all through the session. Programmer can store the **session data** with Cookie object. It is another way of tracking session data.



Servlet Cookies are **created on the serve**r (by Servlet container) and sent to the client’s browser for temporary storage. The cookies can be sent to and fro between client and server in their conversation. Client is not aware of the entire process of Cookie management as everything is done by the server and client is kept completely transparent.

The **Cookie object** created on the server is attached to response object and sent to client. The Cookie carries implicitly the client information like **name**, **path**, **host** and **connection type** etc. and the combination of this information is used by server to know cookie is coming from which client.

Cookies are used mainly for tracking different types of client’s information. That is, the Web server can store data on the client browser in the form of Cookie objects.

**Uses of a Servlet Cookie are**

1. To store User name and Password
2. To write Shopping cart in e-Commerce application
3. Advertisement on client browser
4. Customization of Web sites etc.

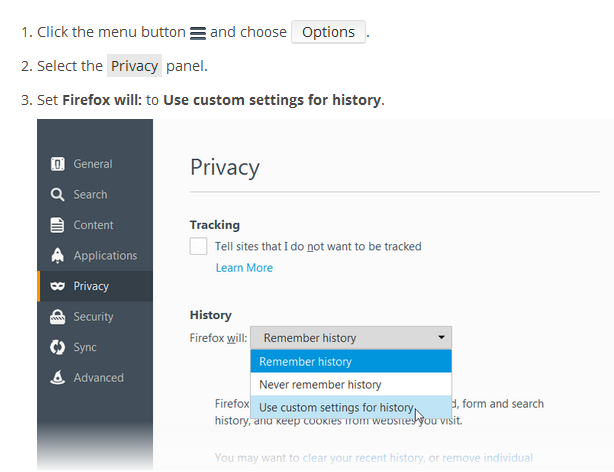
**Advantage of Cookies**

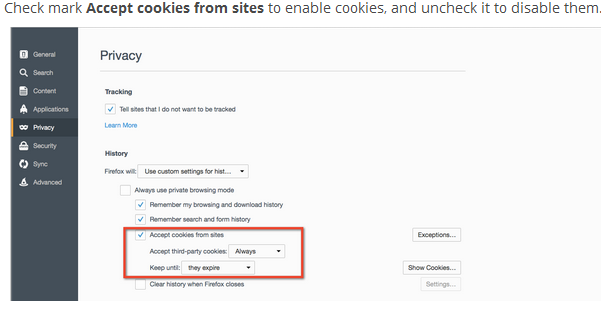
1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

**Disadvantage of Cookies**

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

How to disable and enable cookies in Mozilla firefox browser :





Choose how long cookies are allowed to be stored:

* Keep until:  
  **they expire**: Each cookie will be removed when it reaches its expiration date, which is set by the site that sent the cookie.  
  **I close Firefox**: The cookies that are stored on your computer will be removed when Firefox is closed.

There are two types of cookies and they are differentiated on their life span.

1. **Session Cookies ( Non – Persistent ) :**

Session cookies get expired and deleted from browser when the browser closes, thereby, they do not carry any **expiration time** constraint parameter. They are stored temporarily (just for the usage of session) in the **RAM memory**.

1. **Persistent Cookies:**

As the name indicates, they **persist long time** on the client browser. The time of expiration can be set programmatically for each Cookie. These Cookies are **stored on hard disk**. The Cookie gets deleted automatically when the time or age expires.

**How to create Cookie:**

**Cookie** is a class from **javax.servlet.http** package to created and handle the cookies :

**Public class Cookie extends java.lang.Object implements java.lang.Cloneable**

**Constructor:**

**public Cookie(String name, String value):**

Constructs a cookie object with the **name and value**.

Cookie **uidCookie**= new Cookie("UID", "MNRAO");

**Methods of Cookie class:**

1. **public String getName() :** Returns the name of the cookie. it is UID.
2. **String getValue():** Returns the value of the cookie. it is MNRAO.
3. **void setValue(String newValue):** In the previous Cookie, the value set is MNRAO. If the user would like to modify to new values, this method is useful.
4. **public void setComment(String purpose)**
5. **public String getComment()**
6. **public void setMaxAge(int expiryTime) 🡪 expiryTime in seconds**
7. **public int getMaxAge():**
8. **public void setDomain(String domainName):**

The domainName indicates the domains (servers) in which the cookie should be made available. For example, the parameter ("www.gmail.com") is visible to server [www.gmail.com](http://www.gmail.com).

1. **public String getDomain()**
2. **public void setPath(String uri) :**

path for the cookie to which the client should return the cookie. General practice is, the cookie is returned to the same server which sent them.

The cookie is visible to all subpaths or subdirectories on the server. For example, the path "/finance" makes cookie visible to finance directory and also all its subdirectories.

1. **getPath():** Returns the path on the server to which the browser returns the cookie. For example, /finance.
2. **void setSecure(boolean flag):**

Dictates how the browser should return the cookie to server. If true, it must be secure protocol like HTTPS (HyperText Transfer Protocol Secured) or SSL (Secured Socket Layer). If this is not set, the default is false indicating general protocol like HHTP.

1. **public boolean getSecure():**

Returns a boolean value of true if the browser is using a secure protocol to return the cookie to the server and false indicates browser is using any protocol like HTTP.

1. **public int getVersion(): Returns the version of the protocol this cookie is set to work. Returns 1 if RFC 2109 version** is set or 0 when uses Netscape specifications.
2. **public void setVersion(int version):** Sets the version of the cookie protocol. Set with 1 if RFC 2109 is used and 0 when Netscape cookie specification is used.
3. **public Object clone():** Returns a clone copy of the Cookie when the original clone() method of Object class is overridden.

**Creating a Cookie and adding to response Object:**

* 1. Cookie uidCookie = new Cookie("uid", "MNRAO");
  2. **response**.addCookie(uidCookie);

**Deleting a Cookie:**

* 1. uidCookie.setMaxAge(0);//changing the maximum age to 0 seconds
  2. **response**.addCookie(uidCookie);

**Reading cookies from the client:**

Cookie cookies[]=request.getCookies();

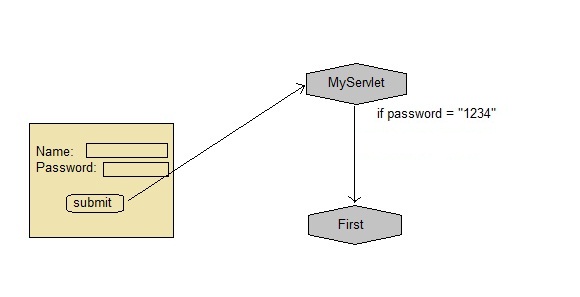
for(int i=0;i<cookies.length;i++)

{

out.print("<br>"+cookies[i].getName()+" "+cookies[i].getValue);

}

**Example for Session Tracking using Cookie:**



Index.html:

<formaction=*"validate"*method=*"post"*>

Name:<inputtype=*"text"*name=*"uid"*/><br/>

Password:<inputtype=*"text"*name=*"pwd"*><br/>

<inputtype=*"submit"*value=*"submit"*>

</form>

Web.xml:

<web-app>

<servlet>

<servlet-name>validate</servlet-name>

<servlet-class>ValidateServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>validate</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>first</servlet-name>

<servlet-class>FirstServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>first</servlet-name>

<url-pattern>/first</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>Index.html</welcome-file>

</welcome-file-list>

</web-app>

ValidateServlet**.java:**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**ValidateServlet **extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

String name = request.getParameter("uid");

String pass = request.getParameter("pwd");

**if**(name.equals("MNRAO") &&pass.equals("1234"))

{

Cookie ck = **new** Cookie("username",name);

response.addCookie(ck);

response.sendRedirect("first");

}

}

}

FirstServlet.java:

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**FirstServlet**extends** HttpServlet

{

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

Cookie[] cks = request.getCookies();

out.println("Welcome "+cks[0].getValue());

}

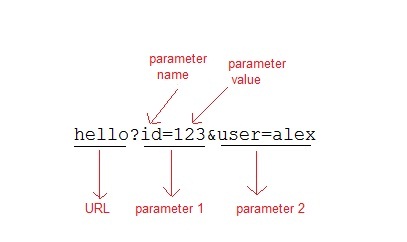
}

**Using URL Rewriting for Session Management:**

If the client has disabled cookies in the browser then session management using cookie wont work. In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work.

In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal(=) sign.

For Example:



When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL. The **Web Container** will fetch the extra part of the requested URL and use it for session management.

The getParameter() method is used to get the parameter value at the server side.

**Example for using URL rewriting:**

**Index.html:**

<formaction=*"validate"*method=*"post"*>

Name:<inputtype=*"text"*name=*"uid"*/><br/>

Password:<inputtype=*"text"*name=*"pwd"*/><br/>

<inputtype=*"submit"*value=*"submit"/*>

</form>

**Web.xml:**

<web-app>

<servlet>

<servlet-name>validate</servlet-name>

<servlet-class>ValidateServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>validate</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>first</servlet-name>

<servlet-class>FirstServle</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>first</servlet-name>

<url-pattern>/first</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>Index.html</welcome-file>

</welcome-file-list>

</web-app>

**ValidateServlet:**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**ValidateServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

String name = request.getParameter("uid");

String pass = request.getParameter("pwd");

**if**(name.equals("MNRAO") &&pass.equals("1234"))

{

response.sendRedirect("first?user\_id="+name+"");

}

}

}

**FirstServlet.java:**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**FirstServlet**extends** HttpServlet {

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String user = request.getParameter("user\_uid");

out.println("Welcome "+user);

}

}

**Using Hidden Form Field for Session Management**

Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieved from another servlet.

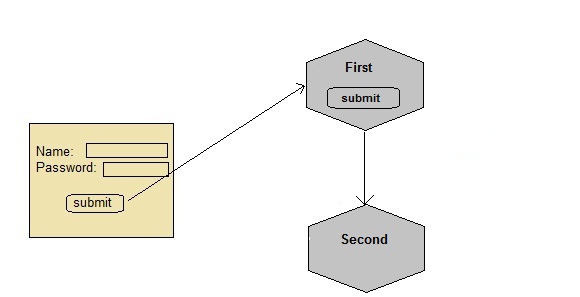
*Advantages :*

* Does not have to depend on browser whether the cookie is disabled or not.
* Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement

*Disadvantage :*

* Extra form submission is required on every page. This is a big overhead.

#### Exmple for usage of Hidden Form Field for Session



**Index.htmL:**

<formmethod=*"post"*action=*"validate"*>

Name:<inputtype=*"text"*name=*"uid"*/><br/>

Password:<inputtype=*"hidden"*name=*"pwd"*value=”abc”><br/>

<inputtype=*"submit"*value=*"submit"*>

</form>

**Web.xml:**

<web-app>

<servlet>

<servlet-name>first</servlet-name>

<servlet-class>FirstServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>first</servlet-name>

<url-pattern>/first</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>second</servlet-name>

<servlet-class>SecondServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>second</servlet-name>

<url-pattern>/second</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

FirstServlet.java:

**publicclass**FirstServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

//getting value submitted in form from HTML file

String user = request.getParameter("uid");

//creating a new hidden form field

out.println("<form action='second'>");

out.println("<input type='hidden' name='user\_id' value='"+user+"'>");

out.println("<input type='submit' value='submit' >");

out.println("</form>");

}

}

**SecondServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**SecondServlet**extends** HttpServlet

{

**protectedvoid** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

//getting parameter from the hidden field

String user = request.getParameter("user\_id");

out.println("Welcome "+user);

}

}

**Event and Listener in Servlet:**

Events are basically occurrence of something.

Changing the state of an object is known as an event.

We can perform some important tasks at the occurrence of these exceptions, such as counting total and current logged-in users, creating database connection object etc.

Event classes and Listener interfaces are from the javax.servlet and javax.servlet.http packages.

## Event classes

1. ServletContextEvent
2. ServletContextAttributeEvent
3. HttpSessionEvent
4. HttpSessionBindingEvent
5. ServletRequestEvent
6. ServletRequestAttributeEvent

## Listener interfaces

1. ServletContextListener
2. ServletContextAttributeListener
3. HttpSessionListener
4. HttpSessionBindingListener
5. HttpSessionAttributeListener
6. HttpSessionActivationListener
7. ServletRequestListener
8. ServletRequestAttributeListener

## Constructor of ServletContextEvent class

public ServletContextEvent(ServletContext sc)

## Method of ServletContextEvent class:

## **public ServletContext getServletContext()**: returns the instance of ServletContext.

## Methods of ServletContextListener interface:

1. **public void contextInitialized(ServletContextEvent e)**: is invoked when application is deployed on the server.
2. **public void contextDestroyed(ServletContextEvent e)**: is invoked when application is undeployed from the server.

**Example of ServletContextEvent and ServletContextListener**

## AppContextListener.java:

**import** javax.servlet.\*;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.text.DateFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**publicclass** AppContextListener **implements** ServletContextListener

{

**publicvoid** contextInitialized(ServletContextEvent event)

{

**try**

{

ServletContext context = event.getServletContext();

String tomcatHome = System.*getenv*("CATALINA\_HOME");

String logFilePath = tomcatHome;

System.***out***.println("Log location : "+logFilePath);

File outputFile = **new** File(logFilePath+"/Success\_Log.txt");

FileWriter fout = **new** FileWriter(outputFile,**true**);

String contextPath = context.getContextPath();

DateFormat df = **new** SimpleDateFormat("dd/MM/yy HH:mm:ss");

Date dateobj = **new** Date();

String currentTime = df.format(dateobj);

fout.write("Deployed the Project "+contextPath+" at Time : "+currentTime);

fout.close();

}

**catch**(Exception e)

{

e.printStackTrace();

}

}

**publicvoid** contextDestroyed(ServletContextEvent event)

{

**try**

{

ServletContext context = event.getServletContext();

String tomcatHome = System.*getenv*("CATALINA\_HOME");

String logFilePath = tomcatHome;

System.***out***.println("Log location : "+logFilePath);

File outputFile = **new** File(logFilePath+"/Success\_Log.txt");

FileWriter fout = **new** FileWriter(outputFile,**true**);

String contextPath = context.getContextPath();

DateFormat df = **new** SimpleDateFormat("dd/MM/yy HH:mm:ss");

Date dateobj = **new** Date();

String currentTime = df.format(dateobj);

fout.write("Un Deployed the Project "+contextPath+" at Time : "+currentTime);

fout.close();

}

**catch**(Exception e)

{

e.printStackTrace();

}

}

}

## Web.xml:

<web-app>

<listener>

<listener-class>AppContextListener</listener-class>

</listener>

<servlet>

<servlet-name>validate</servlet-name>

<servlet-class>MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>validate</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>First</servlet-name>

<servlet-class>First</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>First</servlet-name>

<url-pattern>/First</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>Index.html</welcome-file>

</welcome-file-list>

## </web-app>

**ServletContextAttributeEvent:**

Implementation of ServletContextAttributeListener interface, receive notifications of changes to the attribute list on the servlet context of a web application.

**Methods of ServletContextAttributeListener:**

* 1. public void attributeAdded(ServletContextAttributeEvent e)

notification that a new attribute was added to the context.

* 1. Public void attributeRemoved(ServletContextAttributeEvent e)

notification that an existing attribute was removed from the context.

* 1. Public void attributeReplaced(ServletContextAttributeEvent e)

notification that an attribute was replaced on the context.

**Example:**

**import** javax.servlet.\*;

**publicclass** AppContextAttributeListener **implements** ServletContextAttributeListener

{

@Override

**publicvoid** attributeAdded(ServletContextAttributeEvent event)

{

// **TODO** Auto-generated method stub

String attributeName = event.getName();

Object attributeValue = event.getValue();

System.***out***.println("attribute added");

System.***out***.println(" Name : "+attributeName);

System.***out***.println(" Value : "+attributeValue);

}

@Override

**publicvoid** attributeRemoved(ServletContextAttributeEvent event)

{

String attributeName = event.getName();

Object attributeValue = event.getValue();

System.***out***.println("attribute removed");

System.***out***.println(" Name : "+attributeName);

System.***out***.println(" Value : "+attributeValue);

}

@Override

**publicvoid** attributeReplaced(ServletContextAttributeEvent event)

{

String attributeName = event.getName();

Object attributeValue = event.getValue();

System.***out***.println("attribute vale Replaced");

System.***out***.println(" Name : "+attributeName);

System.***out***.println(" Value : "+attributeValue);

}

}

## Web.xml:

<web-app>

<listener>

<listener-class>AppContextListener</listener-class>

</listener>

<listener>

<listener-class>AppContextAttributeListener</listener-class>

</listener>

<servlet>

<servlet-name>validate</servlet-name>

<servlet-class>MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>validate</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>First</servlet-name>

<servlet-class>First</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>First</servlet-name>

<url-pattern>/First</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>Index.html</welcome-file>

</welcome-file-list>

## </web-app>

**HttpSessionListener:**

* 1. **publicvoid** sessionCreated(HttpSessionEvent sessionEvent)
  2. **publicvoid** sessionDestroyed(HttpSessionEvent sessionEvent)

**import** javax.servlet.http.HttpSessionEvent;

**import** javax.servlet.http.HttpSessionListener;

**publicclass** AppClentSessionListener **implements** HttpSessionListener

{

@Override

**publicvoid** sessionCreated(HttpSessionEvent sessionEvent)

{

System.***out***.println("Session Created:: ID="+sessionEvent.getSession().getId());

}

@Override

**publicvoid** sessionDestroyed(HttpSessionEvent sessionEvent)

{

System.***out***.println("Session Destroyed: ID="+sessionEvent.getSession().getId());

}

}

## Web.xml:

<web-app>

<listener>

<listener-class>AppContextListener</listener-class>

</listener>

<listener>

<listener-class>AppContextAttributeListener </listener-class>

</listener>

<servlet>

<listener>

<listener-class> AppClentSessionListener</listener-class>

</listener>

<servlet-name>validate</servlet-name>

<servlet-class>MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>validate</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>First</servlet-name>

<servlet-class>First</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>First</servlet-name>

<url-pattern>/First</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>Index.html</welcome-file>

</welcome-file-list>

</web-app>

HttpSessionAttributeListener:

**import** javax.servlet.http.HttpSessionAttributeListener;

**import** javax.servlet.http.HttpSessionBindingEvent;

**publicclass** AppClientSessionAtributeListener **implements**

HttpSessionAttributeListener

{

@Override

**publicvoid** attributeAdded(HttpSessionBindingEvent sessionAttributeEvent)

{

String sessionId = sessionAttributeEvent.getSession().getId();

System.***out***.println("Attribute added to session Id : "+sessionId);

String name = sessionAttributeEvent.getName();

String value = (String) sessionAttributeEvent.getValue();

System.***out***.println("name :"+name);

System.***out***.println("value :"+value);

}

@Override

**publicvoid** attributeRemoved(HttpSessionBindingEvent sessionAttributeEvent)

{

String sessionId = sessionAttributeEvent.getSession().getId();

System.***out***.println("Attribute removed from session Id : "+sessionId);

String name = sessionAttributeEvent.getName();

String value = (String) sessionAttributeEvent.getValue();

System.***out***.println("name :"+name);

System.***out***.println("value :"+value);

}

@Override

**publicvoid** attributeReplaced(HttpSessionBindingEvent sessionAttributeEvent)

{

String sessionId = sessionAttributeEvent.getSession().getId();

System.***out***.println("Attribute replaced in session Id : "+sessionId);

String name = sessionAttributeEvent.getName();

String value = (String) sessionAttributeEvent.getValue();

System.***out***.println("name :"+name);

System.***out***.println("value :"+value);

}

}

## Web.xml:

<web-app>

<listener>

<listener-class>AppContextListener</listener-class>

</listener>

<listener>

<listener-class>AppContextAttributeListener </listener-class>

</listener>

<servlet>

<listener>

<listener-class> AppClentSessionListener</listener-class>

</listener>

<listener>

<listener-class> AppClientSessionAtributeListener</listener-class>

</listener>

<servlet-name>validate</servlet-name>

<servlet-class>MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>validate</servlet-name>

<url-pattern>/validate</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>First</servlet-name>

<servlet-class>First</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>First</servlet-name>

<url-pattern>/First</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>Index.html</welcome-file>

</welcome-file-list>

</web-app>

**HttpSessionBindingListener:**

The HttpSessionBindingListener interface is implemented when an object needs to be notified if it's being bound to a session or unbound from a session.

Objects implement this interface so that they can be notified when they are being bound or unbound from a HttpSession.

HttpSessionBindingEvent communicates the event and identifies the session into which the object is bound. Similarly, when an unbinding occurs (using HttpSession.removeAttribute()) HttpSessionBindingEvent communicates the event and identifies the session from which the object is unbound.

Note, this listener will not be declared in the deployment descriptor as the same as HttpSessionActivationListener interface. The container at runtime will introspect this object to see if it implements the HttpSessionActivationListener and/or HttpSessionBindingListener and fires appropriate events to the object.

These methods have a HttpSessionBindingEvent parameter that can be used to retrieve the session that the object was bound to and the name it was given in the session.

**import** javax.servlet.http.HttpSessionBindingEvent;

**import** javax.servlet.http.HttpSessionBindingListener;

**publicclass** AppClientSessionBindingListener **implements**

HttpSessionBindingListener

{

@Override

**publicvoid** valueBound(HttpSessionBindingEvent event)

{

System.***out***.println("Value Bounded to session");

String name = event.getName();

Object value = event.getValue();

System.***out***.println("name : "+name);

System.***out***.println("value : "+value);

}

@Override

**publicvoid** valueUnbound(HttpSessionBindingEvent event)

{

System.***out***.println("Value UnBounded to session");

String name = event.getName();

Object value = event.getValue();

System.***out***.println("name : "+name);

System.***out***.println("value : "+value);

}

}

**ServletRequestListener:**

**ServletRequestEvent** class gives notification about lifecycle events for a **ServletRequest**. The source of the event is the ServletContext of the web application. **ServletRequestListener** receives the notifications generated by ServletRequestEvent and performs some specified tasks based on them.

void requestDestroyed(ServletRequestEvent event) :

is invoked when request is about to go out of scope of the web application..

void requestInitialized(ServletRequestEvent event):

is invoked when request is about to come into scope of the web application.

As the name suggests these methods will be called when request is initialized and when it is destroyed, hence we can use them for various purposes, like **variable cleanup** when request is destroyed, or performing some **initial checks** when request is received etc.

**import** javax.servlet.ServletRequest;

**import** javax.servlet.ServletRequestEvent;

**import** javax.servlet.ServletRequestListener;

**publicclass** AppClientServletRequestListener **implements** ServletRequestListener

{

@Override

**publicvoid** requestDestroyed(ServletRequestEvent event)

{

ServletRequest servletRequest = event.getServletRequest();

System.***out***.println("ServletRequest destroyed. Remote IP="+servletRequest.getRemoteAddr());

}

@Override

**publicvoid** requestInitialized(ServletRequestEvent event)

{

ServletRequest servletRequest = event.getServletRequest();

System.***out***.println("ServletRequest Initialized. Remote IP="+servletRequest.getRemoteAddr());

}

}

**ServletRequestAttributeListener:**

This interface is used for receiving notification events about ServletRequest attribute changes. Notifications will be generated while the request is within the scope of the web application in which the listener is registered. A ServletRequest has a very short lifespan. It only lives when it enters the first servlet or filter and is destroyed when it reaches the last servlet or filter. The ServletRequestAttributeListener can be registerd by @WebListener annotation, adding the listener to the servlet descriptor or programmatically adding a listener with .addListener() to the servlet context. In this example we use the @WebListener annotation.

This interface lets us listen to the following events, the names speak for themselves.

* attributeAdded()
* attributeRemoved()
* attributeReplaced()

To register a listener we can add the @WebListener, define the listener in the servlet descriptor (web.xml) or programatigally add it to the servlet context. In this example we choose to add the listener through the @WebListener annotation.

**import** javax.servlet.\*;

**import** javax.servlet.annotation.WebListener;

@WebListener

**publicclass**AppClientServletRequestAttribute **implements**

ServletRequestAttributeListener

{

@Override

**publicvoid** attributeAdded(ServletRequestAttributeEvent event)

{

System.***out***.println("attribute was added");

System.***out***.println("name :"+event.getName());

System.***out***.println( "value: " + event.getValue());

}

@Override

**publicvoid** attributeRemoved(ServletRequestAttributeEvent event)

{

System.***out***.println("attribute was removed");

System.***out***.println("name :"+event.getName());

System.***out***.println( "value: " + event.getValue());

}

@Override

**publicvoid** attributeReplaced(ServletRequestAttributeEvent event)

{

System.***out***.println("attribute was replaced");

System.***out***.println("name :"+event.getName());

System.***out***.println( "value: " + event.getValue());

}

}

<web-app>

<listener>

<listener-class>AppClientServletRequestAttribute</listener-class>

</listener>

</web-app>

## How does it work

Every time an ServletRequest attribute is added, replaced or removed the corresponding method is invoked. This allows us to track changes to certain attributes in our ServletRequest.

**File Handling:**

## Servlet Read text file and Return contents Example :

## *Sometimes, the client may ask the Servlet to send the file contents of a file existing on the server. It is not file download. Here, the Servlet reads manually line-by-line and send each line separately to client. Read Servlet Read text file.*

## ****TextFileSend.html:****

<BODY>

<H2> Getting File contents from the Server </H2>

<FORMACTION=*"*file*"*METHOD=*"get"*>

Enter File Name <INPUTTYPE=*"text"*NAME=*"filename"*><BR>

<INPUTTYPE=*"submit"*VALUE=*"Send Me"*>

<INPUTTYPE=*"reset"*VALUE=*"Clear"*>

</FORM>

## </BODY>

## **web.xml:**

<web-app>

<servlet>

<servlet-name>file</servlet-name>

<servlet-class>TextFileSendServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>file</servlet-name>

<url-pattern>/file</url-pattern>

</servlet-mapping>

</web-app>

**import** javax.servlet.ServletException;

**import** javax.servlet.http.\*;

**import** java.io.\*;

**publicclass**TextFileSendServlet**extends** HttpServlet

{

**publicvoid** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException

{

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

String filePath = req.getParameter("filename");

BufferedReader br = **new** BufferedReader(**new** FileReader(filePath));

String str;

**while**( (str = br.readLine()) != **null** )

{

pw.println(str + "<BR>");

}

br.close();

pw.close();

}

}

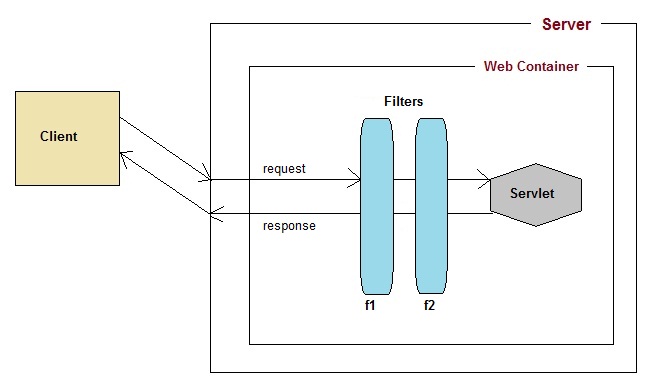
**Servlet Filters:**

**Filters** are compontents that you can use and configure to perform some filtering tasks.

Filter is used for pre-processing of requests and post-processing of responses.

We can have any number of filters for pre-processing of a request and post-processing of a response.

Filters are configured in the deployment descriptor of a web application.



#### How Filters Works?

* When a request reaches the **Web Container**, it checks if any filter has URL patterns that matches the requested URL.
* The **Web Container** locates the first filter with a matching URL pattern and filter's code is executed.
* If another filter has a matching URL pattern, its code is then executed. This continues until there are no filters with matching URL patterns left.
* If no error occurs, the request passes to the target servlet. Hence we know, that the request will be passed to the target servlet only when all the related Filters are successfully executed.
* The servlet returns the response back to its caller. The last filter that was applied to the request is the first filter applied to the response.
* At last the response will be passed to the **Web Container** which passes it to the client.

**Filter API** is part of **Servlet API**. Filter interface is found in the **javax.servlet** package.

For creating a filter, we must implement Filter interface. Filter interface gives the following life cycle methods for a filter:

* 1. void init(FilterConfig filterConfig):

invoked by the web container to indicate to a filter that it is being placed into service.

* 1. void doFilter(ServletRequest request, ServletResponse response, FilterChain chain):

invoked by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.

* 1. void destroy():

invoked by the web container to indicate to a filter that it is being taken out of service

#### What is FilterChain Interface?

**FilterChain** object is used to invoke the next filter in the chain, or if the calling filter is the last filter in the chain then the resource at the end of the chain invoked. The resources at the end of Filter chain can either be a target Servlet(in case of request flow) or the Client(in case of response flow) as described in the diagram above.

#### Declaring a Filter inside Deployment Descriptor

<web-app>

<filter>

<filter-name>file</filter-name>

<filter-class>MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>file</filter-name>

<url-pattern>/file</url-pattern>

(or)

<servlet-name>first</servlet-name>

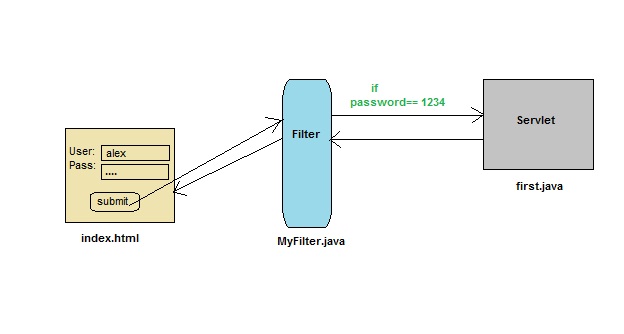
</filter-mapping>

</web-app>

Example demonstrating Filter usage:

In this example we are using Filter to authenticate(check correct username and password). Here **index.html** will ask username and password from the user, **MyFilter** will validate the password entered by the user, if the user has entered "1234" as password, then he will be forwarded to **first** servlet else the index.html will be shown again to the user.

This is exactly what we used to do earlier using two servlet classes earlier, one for validation and the other to Welcome the user. Now we will insert a Filter for validating the user.



Index.html:

<formaction=*"first"*method=*"post"*>

Name:<inputtype=*"text"*name=*"uid"*/><br/>

Password:<inputtype=*"text"*name=*"pwd"*/><br/>

<inputtype=*"submit"*value=*"submit"*/>

</form>

Web.xml:

<web-app>

<filter>

<filter-name>MyFilter</filter-name>

<filter-class>MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

<servlet-name>first</servlet-name>

</filter-mapping>

<servlet>

<servlet-name>first</servlet-name>

<servlet-class>FirstServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>first</servlet-name>

<url-pattern>/first</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

**import** java.io.\*;

**import** javax.servlet.\*;

**publicclass** MyFilter **implements** Filter

{

**publicvoid** init(FilterConfig fc) **throws** ServletException

{ }

**publicvoid** doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) **throws** IOException, ServletException

{

PrintWriter out = response.getWriter();

String pass = request.getParameter("pwd");

**if**(pass.equals("1234"))

{

chain.doFilter(request, response);

}

**else**

{

out.println("You have enter a wrong password");

RequestDispatcher rs = request.getRequestDispatcher("index.html");

rs.include(request, response);

}

}

**publicvoid** destroy()

{ }

}

**FirstServlet.java:**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**FirstServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String user = request.getParameter("uid");

out.println("Wellcome "+user);

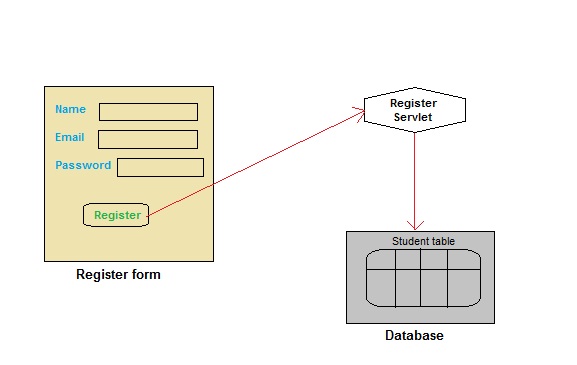
}

}

**Some Important Examples:**

**Registration form in Servlet**

In this example you will see how to develop a registration form in Servlet. To develop a registration form you will need to connect your servlet application with database. Here we are using **MySQL** database.



#### Create a Table in your Database:

create table **Student ( name** varchar(60),

**email** varchar(60),

**pass** varchar(100)

);

**Index.html:**

<html>

<head>

<title>Register form</title>

</head>

<body>

<formaction=*"register"*method=*"post"*>

Name:<inputtype=*"text"*name=*"name"*/><br/>

Email ID:<inputtype=*"text"*name=*"email"*/><br/>

Password:<inputtype=*"text"*name=*"pass"*/><br/>

<inputtype=*"submit"*value=*"submit"*/>

</form>

</body>

</html>

#### RegisterServlet.java

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**import** java.sql.\*;

**publicclass**RegisterServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

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

String email = request.getParameter("email");

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

**try**

{

//loading drivers for mysql

Class.*forName*("com.mysql.jdbc.Driver");

//creating connection with the database

Connection con=DriverManager.*getConnection*

("jdbc:mysql://localhost:3306/test","root","hadoop");

PreparedStatement ps=con.prepareStatement

("insert into Student values(?,?,?)");

ps.setString(1, name);

ps.setString(2, email);

ps.setString(3, pass);

**int**i=ps.executeUpdate();

**if**(i>0)

{

out.println("You are sucessfully registered");

}

}

**catch**(Exception se)

{

se.printStackTrace();

}

}

}

Web.xml:

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appversion=*"3.0"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>

<servlet>

<servlet-name>register</servlet-name>

<servlet-class>RegisterServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>register</servlet-name>

<url-pattern>/register</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

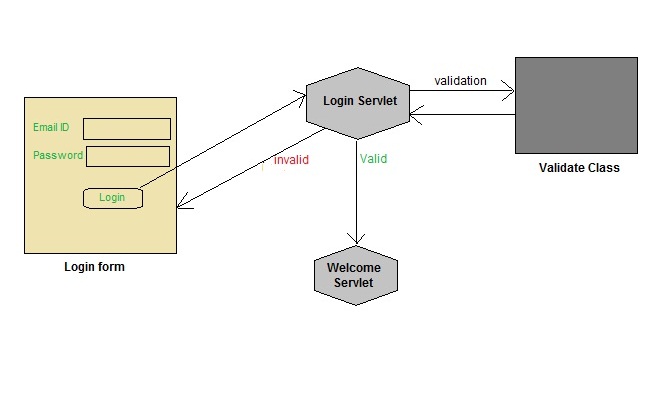
</welcome-file-list>

</web-app>

### Login System in Servlet

In this example we will show you how to develop a login form using servlet. Here we are using **MySql** database. List of file to be created are:

* **index.html**
* **LoginServlet.java**
* **Validate.java**
* **WelcomeSevlet.java**
* **web.xml**



To try this application you will need to create a table in your database and enter some record into it.

#### index.html

<html>

<head>

<title>login form</title>

</head>

<body>

<formaction=*"login"*method=*"post"*>

Email ID:<inputtype=*"text"*name=*"email"*/><br/>

Password:<inputtype=*"text"*name=*"pass"*/><br/>

<inputtype=*"submit"*value=*"login"*/>

</form>

</body>

</html>

**LoginServlet.java**

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**LoginServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String email = request.getParameter("email");

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

**if**(Validate.*checkUser*(email, pass))

{

RequestDispatcher rs = request.getRequestDispatcher("Welcome");

rs.forward(request, response);

}

**else**

{

out.println("Username or Password incorrect");

RequestDispatcher rs = request.getRequestDispatcher("index.html");

rs.include(request, response);

}

}

}

**Validate.java:**

**import** java.sql.\*;

**publicclass**Validate

{

**publicstaticboolean** checkUser(String email,String pass)

{

**boolean**validity =**false**;

**try**

{

//loading drivers for mysql

Class.*forName*("com.mysql.jdbc.Driver");

//creating connection with the database

Connection con=DriverManager.*getConnection*

("jdbc:mysql://localhost:3306/test","root","hadoop");

PreparedStatement ps =con.prepareStatement

("select \* from register where email=? and pass=?");

ps.setString(1, email);

ps.setString(2, pass);

ResultSet rs =ps.executeQuery();

validity = rs.next();

}

**catch**(Exception e)

{

e.printStackTrace();

}

**return**validity;

}

}

WelcomeServlet.java

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**WelcomeServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

out.println("Welcome user");

}

}

Web.xml:

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appversion=*"3.0"*xmlns=*"http://java.sun.com/xml/ns/javaee"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>LoginServlet</servlet-class>

</servlet>

<servlet>

<servlet-name>welcome</servlet-name>

<servlet-class>WelcomeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>welcome</servlet-name>

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

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

### Email Sending using Servlet

In this example we will see how to send an email in Servlet application. We will be using the **JavaMail** API that provides all the classes required for sending an email. JavaMail API encapsulates two important packages **javax.mail** and **javax.mail.internet**. These packages provide classes that can be used to send and recieve simple emails. You simply need an Internet connection to send email using this simple Application.

Following are the names of Files to be created :

* **index.html** will get the input from user
* **MailApp.java** servlet file will control the request and response. It will invoke **send()** of **SendMail** class that we have created to send the mail.
* **SendMail.java**, a java class that contains method to send mail.

#### index.html

<formaction=*"mail"*method=*"post"*>

To:<inputtype=*"text"*name=*"to"*/><br/>

Subject:<inputtype=*"text"*name=*"subject"*/><br/>

Message:<inputtype=*"text"*name=*"message"*/><br/>

Your Email id:<inputtype=*"text"*name=*"user"*><br/>

Password :<inputtype=*"password"*name=*"pass"*/><br/>

<inputtype=*"submit"*value=*"send"*/>

</form>

#### MailAppServlet.java

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**MailAppServlet**extends** HttpServlet

{

**protectedvoid** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String to = request.getParameter("to");

String subject = request.getParameter("subject");

String message = request.getParameter("message");

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

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

SendMail.*send*(to,subject, message, user, pass);

out.println("Mail send successfully");

}

}

SendMail.java:

**import** java.util.\*;

**import** javax.mail.\*;

**import** javax.mail.internet.\*;

**publicclass** SendMail

{

**publicstaticvoid** send(String to, String sub,

String msg, **final** String user,**final** String pass)

{

//create an instance of Properties Class

Properties props = **new** Properties();

/\* Specifies the IP address of your default mail server

for e.g if you are using gmail server as an email sever

you will pass smtp.gmail.com as value of mail.smtp host.

As shown here in the code.

Change accordingly, if your email id is not a gmail id

\*/

props.put("mail.smtp.host", "smtp.gmail.com");

//below mentioned mail.smtp.port is optional

props.put("mail.smtp.port", "27");

props.put("mail.smtp.auth", "true");

props.put("mail.smtp.starttls.enable", "true");

/\* Pass Properties object(props) and Authenticator object

for authentication to Session instance

\*/

Session session = Session.*getInstance*(props,**new** javax.mail.Authenticator()

{

**protected** PasswordAuthentication getPasswordAuthentication()

{

**returnnew** PasswordAuthentication(user,pass);

}

});

**try**

{

/\* Create an instance of MimeMessage,

it accept MIME types and headers

\*/

MimeMessage message = **new** MimeMessage(session);

message.setFrom(**new** InternetAddress(user));

message.addRecipient(Message.RecipientType.***TO***,**new** InternetAddress(to));

message.setSubject(sub);

message.setText(msg);

/\* Transport class is used to deliver the message to the recipients \*/

Transport.*send*(message);

}

**catch**(Exception e)

{

e.printStackTrace();

}

}

}

Web.xml:

<?xmlversion=*"1.0"*encoding=*"UTF-8"*?>

<web-appversion=*"3.0"*xmlns=*"http://java.sun.com/xml/ns/javaee"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*>

<servlet>

<servlet-name>mail</servlet-name>

<servlet-class>MailAppServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>mail</servlet-name>

<url-pattern>/mail</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

**Improving Servlet performance to fetch records from database**

In this example, we are going to improve the performance of web application to fetch records from the database. To serve this, we are storing the data of the table in a collection, and reusing this collection in our servlet. So, we are not directly hitting the database again and again. By this, we are improving the performance.

To run this application, you need to create following table with some records.

CREATE TABLE  "CSUSER"    (    "USERID" NUMBER,

     "USERNAME" VARCHAR2(4000),

     "USERPASS" VARCHAR2(4000),

     "USEREMAIL" VARCHAR2(4000),

     "USERCOUNTRY" VARCHAR2(4000),

     "CONTACT" NUMBER,

     CONSTRAINT "CSUSER\_PK" PRIMARY KEY ("USERID") ENABLE

   )

/

**Example to Improve the performance of servlet to fetch records from database**

In this example, we have created 6 pages.

1. **index.html**
2. **User.java**
3. **MyListener.java**
4. **MyServlet1.java**
5. **MyServlet2.java**
6. **web.xml**
7. **index.html:**

<ahref=*"servlet1"*>first servlet</a>|

<ahref=*"servlet2"*>second servlet</a>

1. **User.java**

**publicclass** User

{

**privateint**id;

**private** String name;

**private** Stringpassword;

**publicint** getId()

{

**return**id;

}

**publicvoid** setId(**int**id) {

**this**.id = id;

}

**public** String getName() {

**return**name;

}

**publicvoid** setName(String name) {

**this**.name = name;

}

**public** String getPassword() {

**return**password;

}

**publicvoid** setPassword(String password) {

**this**.password = password;

}

}

**3)** MyContextListener**.java**

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletContextEvent;

**import** javax.servlet.ServletContextListener;

**import** java.sql.\*;

**import** java.util.ArrayList;

**publicclass** MyContextListener **implements** ServletContextListener{

**publicvoid** contextInitialized(ServletContextEvent e) {

ArrayList <User>list=**new** ArrayList<User>();

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.*getConnection*(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select \* from csuser");

ResultSet rs=ps.executeQuery();

**while**(rs.next()){

User u=**new** User();

u.setId(rs.getInt(1));

u.setName(rs.getString(2));

u.setPassword(rs.getString(3));

list.add(u);

}

con.close();

}

**catch**(Exception ex){System.***out***.print(ex);}

//storing the ArrayList object in ServletContext

ServletContext context=e.getServletContext();

context.setAttribute("data",list);

}

**publicvoid** contextDestroyed(ServletContextEvent arg0) {

System.***out***.println("project undeployed...");

}

}

**4) MyServlet1.java**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Iterator;

**import** java.util.List;

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass**MyServlet1**extends** HttpServlet

{

**publicvoid** doGet(HttpServletRequest request, HttpServletResponse

response)**throws** ServletException, IOException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**long**before=System.*currentTimeMillis*();

ServletContext context=getServletContext();

Listlist=(List)context.getAttribute("data");

Iteratoritr=list.iterator();

**while**(itr.hasNext()){

User u=(User)itr.next();

out.print("<br>"+u.getId()+" "+u.getName()+" "+u.getPassword());

}

**long**after=System.*currentTimeMillis*();

out.print("<br>total time :"+(after-before));

out.close();

}

}

**5) MyServlet2.java:**

It is same as MyServlet1. This servlet gets the information from the servlet context object and prints it.

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** java.util.Iterator;

**import** java.util.List;

**import** javax.servlet.ServletContext;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**publicclass**MyServlet2**extends** HttpServlet {

**publicvoid** doGet(HttpServletRequestrequest, HttpServletResponse

response)**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**long**before=System.*currentTimeMillis*();

ServletContext context=getServletContext();

Listlist=(List)context.getAttribute("data");

Iteratoritr=list.iterator();

**while**(itr.hasNext()){

User u=(User)itr.next();

out.print("<br>"+u.getId()+" "+u.getName()+" "+u.getPassword());

}

**long**after=System.*currentTimeMillis*();

out.print("<br>total time :"+(after-before));

out.close();

}

}

**6) web.xml:**

Here we are containing the information about servlets and listener.

<web-appversion=*"2.5"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<listener>

<listener-class>MyListener</listener-class>

</listener>

<servlet>

<servlet-name>MyServlet1</servlet-name>

<servlet-class>MyServlet1</servlet-class>

</servlet>

<servlet>

<servlet-name>MyServlet2</servlet-name>

<servlet-class>MyServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>MyServlet1</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>MyServlet2</servlet-name>

<url-pattern>/servlet2</url-pattern>

</servlet-mapping>

</web-app>

# Example to display image using Servlet

In this example, we are using **FileInputStream** class to read image and **ServletOutputStream** class for writing this image content as a response. To make the performance faster, we have used BufferedInputStream and BufferedOutputStream class.

You need to use the content type **image/jpeg**.

In this example, we are assuming that you have java.jpg image inside the c:\test directory. You may change the location accordingly.

To create this application, we have created three files:

1. index.html
2. DisplayImage.java
3. web.xml

**index.html:**

<a href="servlet1">click for photo</a>

**DisplayImage.java**

This servlet class reads the image from the mentioned directory and writes the content in the response object using ServletOutputStream and BufferedOutputStream classes.

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**publicclass**DisplayImage**extends** HttpServlet {

**publicvoid** doGet(HttpServletRequest request,HttpServletResponse response)

**throws** IOException

{

response.setContentType("image/jpeg");

ServletOutputStream out;

out = response.getOutputStream();

FileInputStream fin = **new** FileInputStream("c:\\test\\myphoto.jpg");

BufferedInputStream bin = **new** BufferedInputStream(fin);

BufferedOutputStream bout = **new** BufferedOutputStream(out);

**int**ch =0; ;

**while**((ch=bin.read())!=-1)

{

bout.write(ch);

}

bin.close();

fin.close();

bout.close();

out.close();

}

}