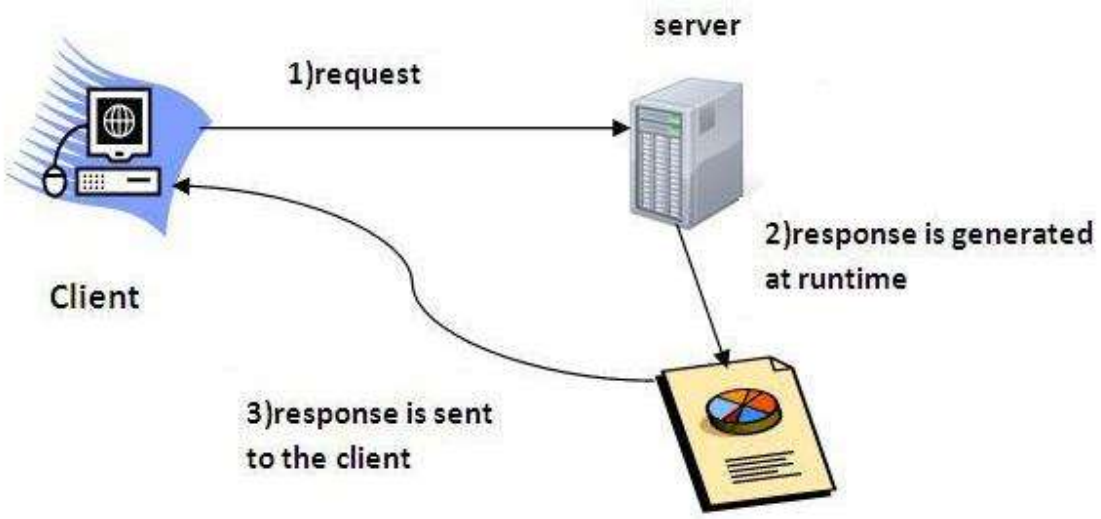
What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.



Do You Know ?

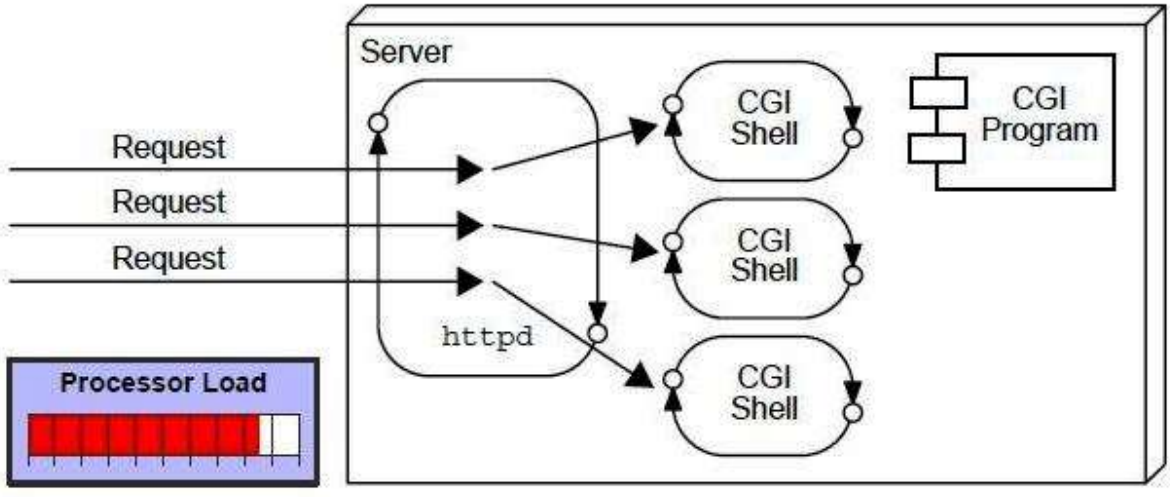
- What is the web application and what is the difference between Get and Post request ?
- What information is received by the web server if we request for a servlet ?
- How to run servlet in Eclipse, MyEclipse and Netbeans IDE ?
- What are the ways for servlet collaboration and what is the difference between RequestDispatcher and sendRedirect() method ?
- What is the difference between ServletConfig and ServletContext interface?
- How many ways we can maintain state of an user ? Which approach is mostly used in web development ?
- How to count total number of visitors and total response time for a request using Filter ?
- How to run servlet with annotation ?
- How to create registration form using Servlet and Oracle database ?
- How can we upload and download file from the server ?

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.

CGI(Commmon Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

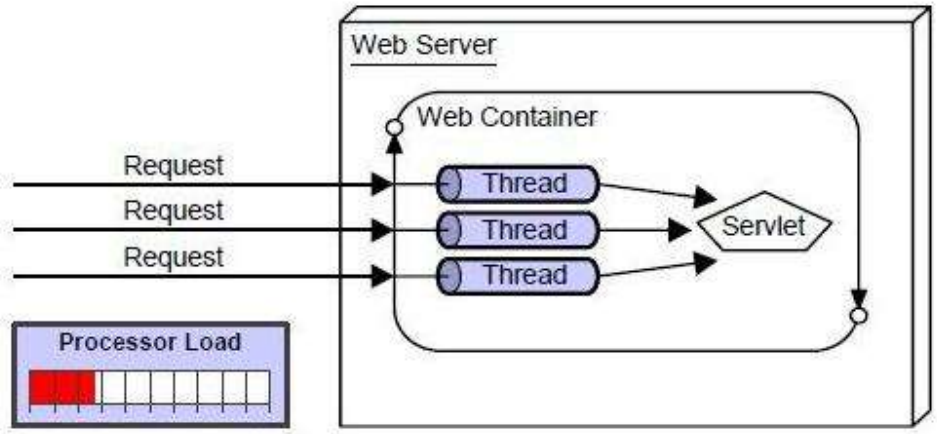


Disadvantages of CGI

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.

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, lighweight, 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 no need to worry about momory leak, garbage collection etc.
4. **Secure:** because it uses java language..

Servlet Terminology

1. Basics of Servlet
2. HTTP
3. Http Request Methods
4. Difference between Get and Post
5. Anatomy of Get Request
6. Anatomy of Post Request
7. Content Type

There are some key points that must be known by the servlet programmer like server, container, get request, post request etc. Let's first discuss these points before starting the servlet technology.

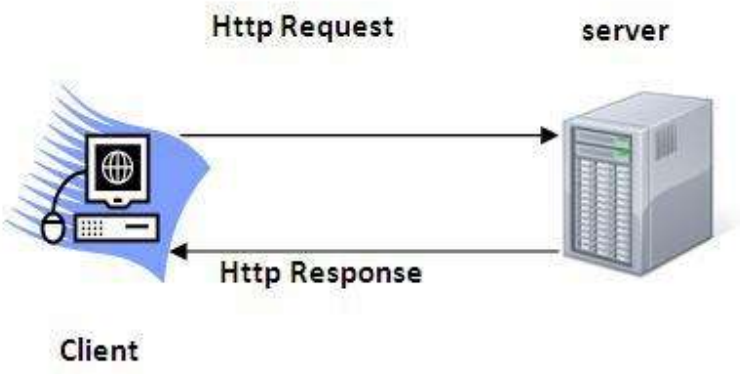
The basic **terminology used in servlet** are given below:

1. HTTP
2. HTTP Request Types
3. Difference between Get and Post method

4. Container
5. Server and Difference between web server and application server
6. Content Type
7. Introduction of XML
8. Deployment

HTTP (Hyper Text Transfer Protocol)

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



Http Request Methods

Every request has a header that tells the status of the client. There are many request methods. Get and Post requests are mostly used.

The http request methods are:

- GET
- POST
- HEAD
- PUT
- DELETE
- OPTIONS
- TRACE

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

What is the difference between Get and Post?

There are many differences between the Get and Post request. Let's see these differences:

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

Anatomy of Get Request

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what informations are sent to the

server.

Anatomy of Post Request

As we know, in case of post request original data is sent in message body. Let's see how informations are passed to the server in case of post

request.

Container

It provides runtime environment for JavaEE (j2ee) applications.

It performs many operations that are given below:

1. Life Cycle Management
2. Multithreaded support
3. Object Pooling
4. Security etc.

Server

It is a running 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.

Example of Web Servers are: **Apache Tomcat** and **Resin**.

Application Server

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc.

Example of Application Servers are:

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

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.

There are many content types:

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip

- images/jpeg
- video/quicktime etc.

Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Servlet Interface

Servlet interface provides common behaviour to all the servlets.

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

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

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

Servlet Example by implementing Servlet interface

Let's see the simple example of servlet by implementing the servlet interface.

It will be better if you learn it after visiting steps to create a servlet.

File: First.java

```
1. import java.io.*;
2. import javax.servlet.*;
3.
4. public class First implements Servlet{
5. ServletConfig config=null;
6.
7. public void init(ServletConfig config){
8. this.config=config;
9. System.out.println("servlet is initialized");
10. }
```

```
11.
12. public void service(ServletRequest req,ServletResponse res)
13. throws IOException,ServletException{
14.
15. res.setContentType("text/html");
16.
17. PrintWriter out=res.getWriter();
18. out.print("<html><body>");
19. out.print("<b>hello simple servlet</b>");
20. out.print("</body></html>");
21.
22. }
23. public void destroy(){System.out.println("servlet is destroyed");}
24. public ServletConfig getServletConfig(){return config;}
25. public String getServletInfo(){return "copyright 2007-1010";}
26. }
```

GenericServlet class

GenericServlet class implements **Servlet**,**ServletConfig** and **Serializable** interfaces. It provides the implementaion of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

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

Methods of GenericServlet class

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

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.

8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

Let's see the simple example of servlet by inheriting the GenericServlet class.

It will be better if you learn it after visiting steps to create a servlet.

File: First.java

```
1. import java.io.*;
2. import javax.servlet.*;
3.
4. public class First extends GenericServlet{
5. public void service(ServletRequest req,ServletResponse res)
6. throws IOException,ServletException{
7.
8. res.setContentType("text/html");
9.
10. PrintWriter out=res.getWriter();
11. out.print("<html><body>");
12. out.print("<b>hello generic servlet</b>");
13. out.print("</body></html>");
14.
15. }
16. }
```

HttpServlet class

1. HttpServlet class
2. Methods of HttpServlet class

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

Methods of HttpServlet class

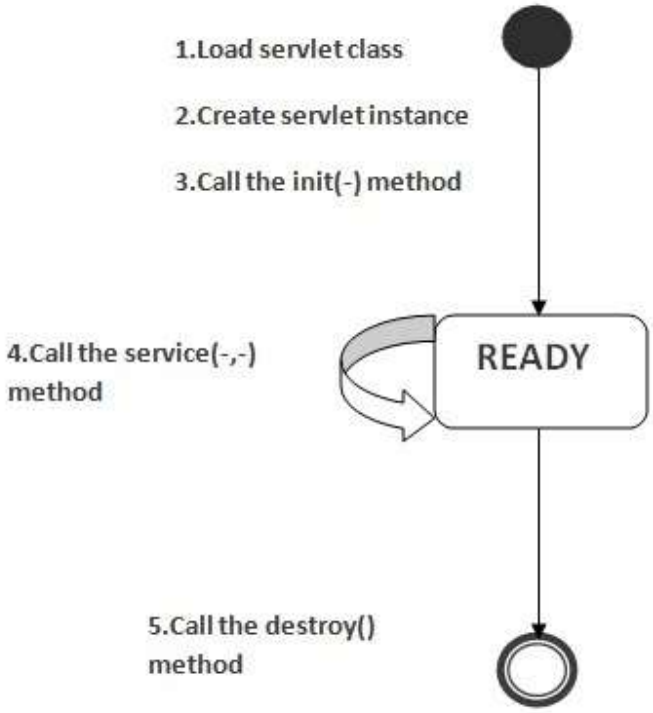
There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

---

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

---

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

---

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
1. public void init(ServletConfig config) throws ServletException
```

---

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
1. public void service(ServletRequest request, ServletResponse response)
2.   throws ServletException, IOException
```

---

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
1. public void destroy()
```

Steps to create a servlet example

1. Steps to create the servlet using Tomcat server
1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the application

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,

2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

---

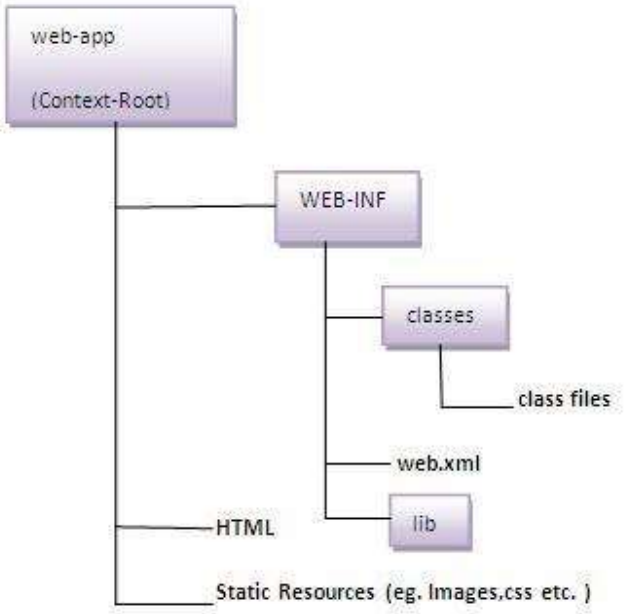
download this example of servlet
download example of servlet by extending GenericServlet
download example of servlet by implementing Servlet interface

---

1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

---

2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

```
1. import javax.servlet.http.*;
```

```
2. import javax.servlet.*;
3. import java.io.*;
4. public class DemoServlet extends HttpServlet{
5. public void doGet(HttpServletRequest req,HttpServletResponse res)
6. throws ServletException,IOException
7. {
8. res.setContentType("text/html");//setting the content type
9. PrintWriter pw=res.getWriter();//get the stream to write the data
10.
11. //writing html in the stream
12. pw.println("<html><body>");
13. pw.println("Welcome to servlet");
14. pw.println("</body></html>");
15.
16. pw.close();//closing the stream
17. }}
```

---

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomacat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes**directory.

4)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

---

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

---

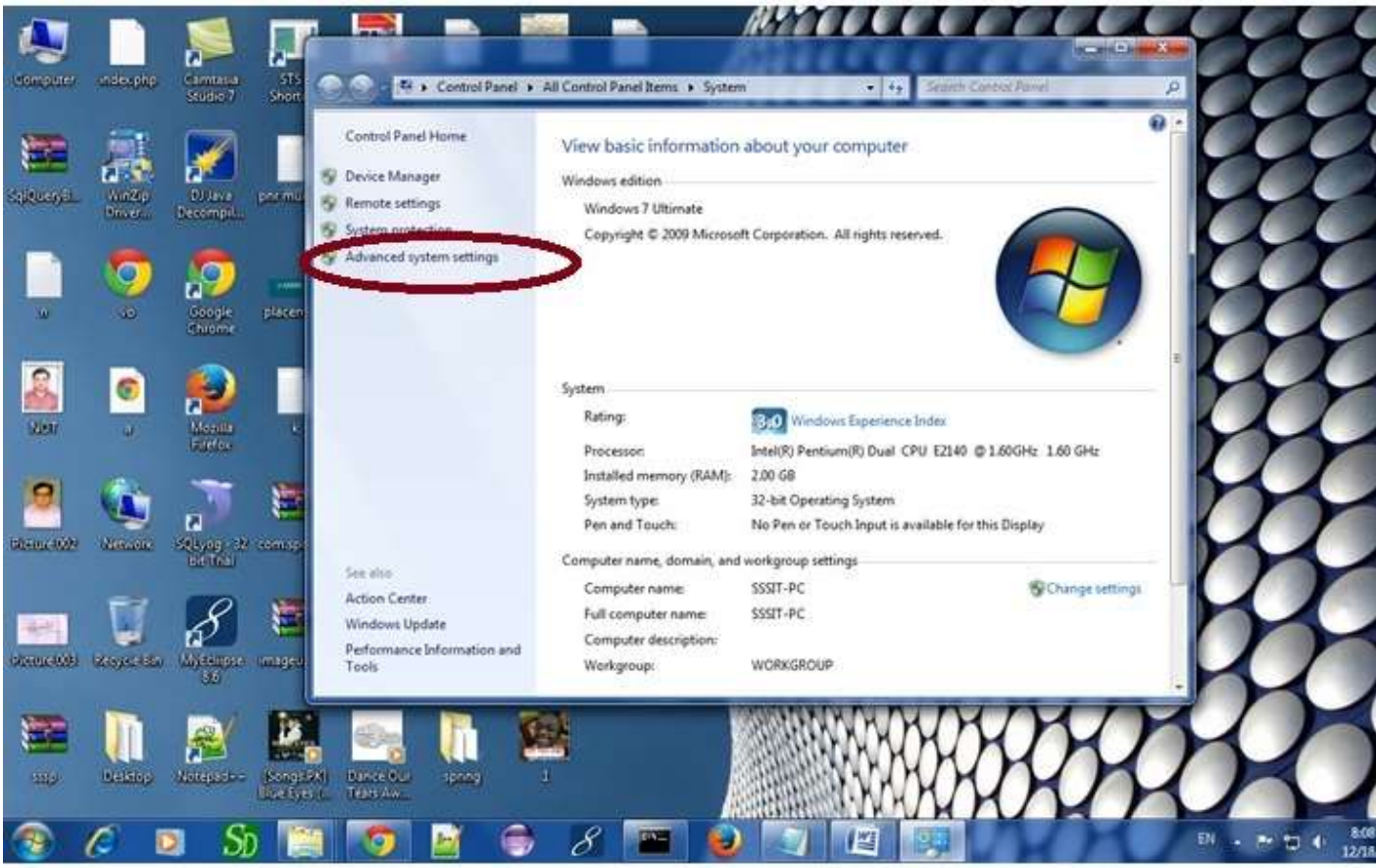
1) How to set JAVA_HOME in environment variable?

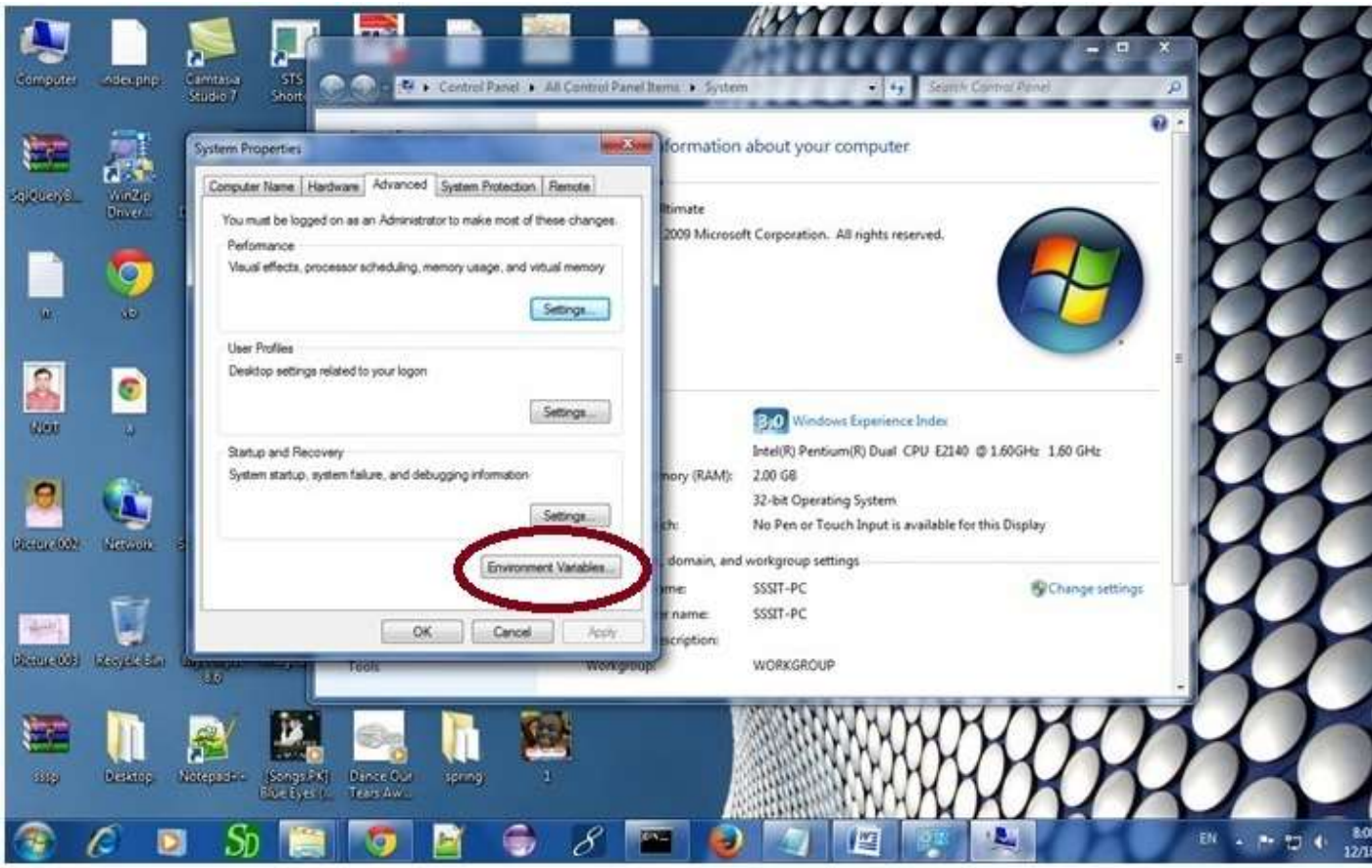
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.

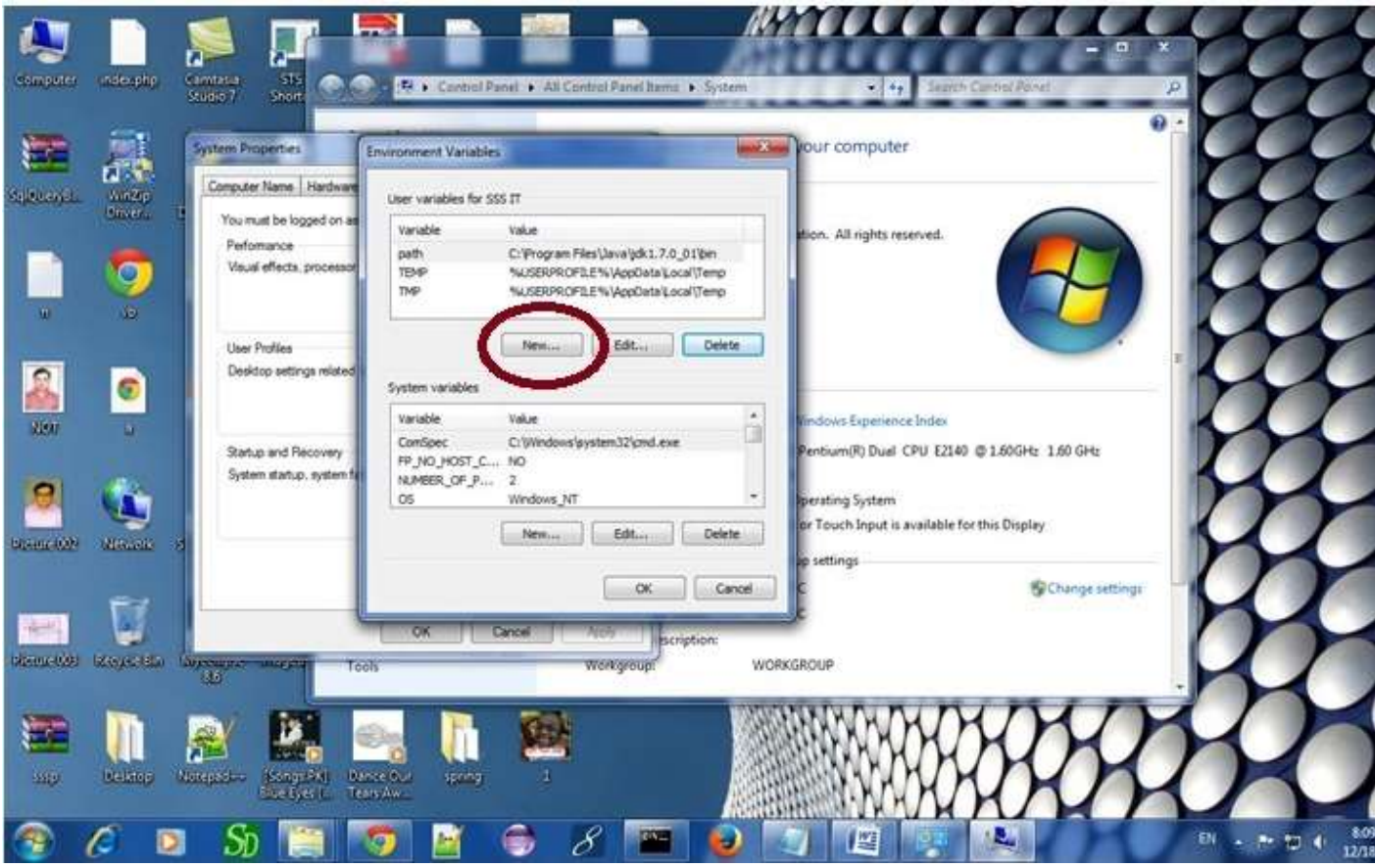
Go to My Computer properties:

Click on advanced system settings tab then environment variables:

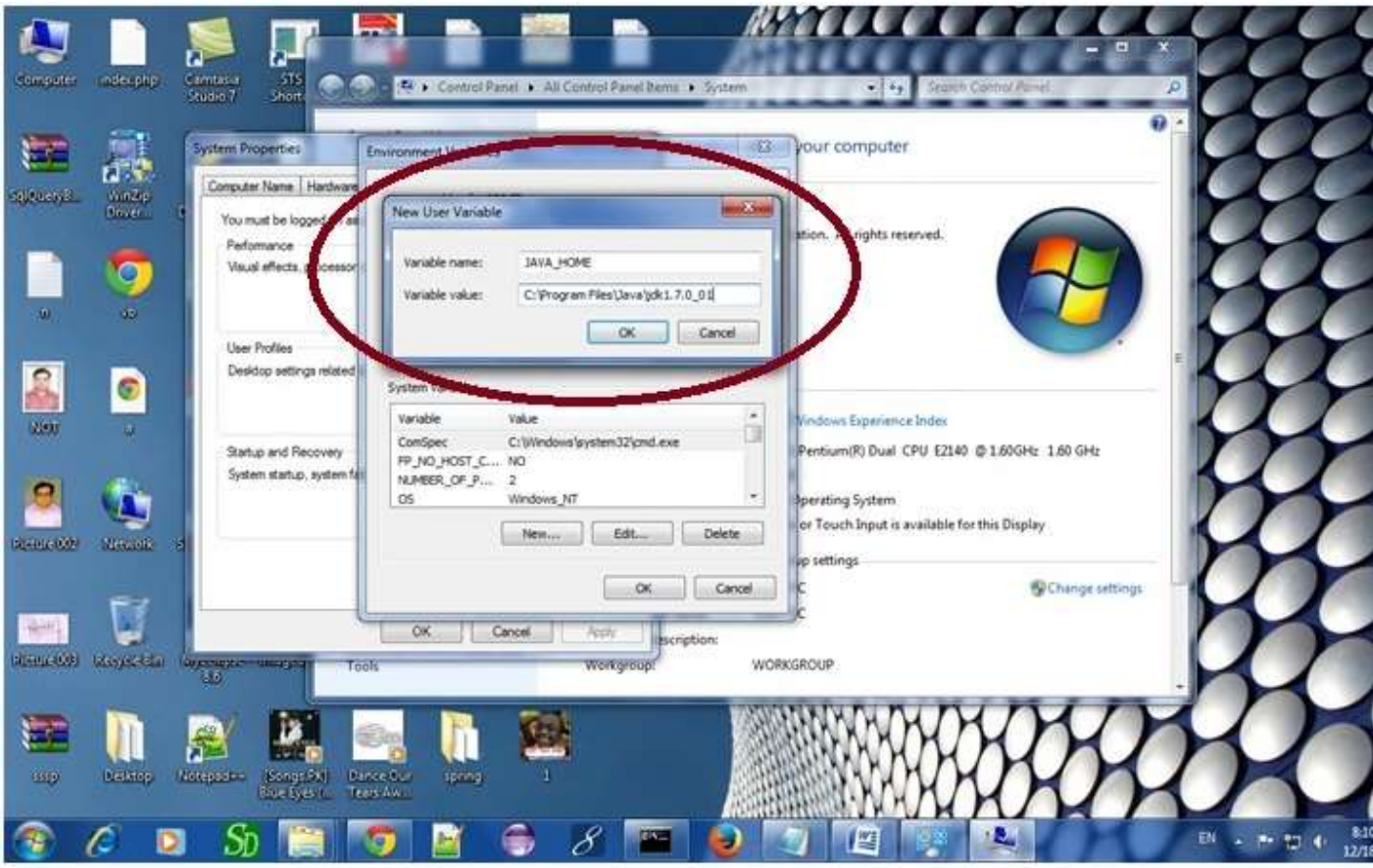

Control Panel ▸ All Control Panel Items ▸ System
Search Control Panel
Control Panel Home
Device Manager
Remote settings
System protection
Advanced system settings
View basic information about your computer
Windows edition
Windows 7 Ultimate
Copyright © 2009 Microsoft Corporation. All rights reserved.
System
Rating:
Windows Experience Index
Processor: Intel(R) Pentium(R) Dual CPU E2140 @ 1.60GHz 1.60 GHz
Installed memory (RAM): 2.00 GB
System type: 32-bit Operating System
Pen and Touch: No Pen or Touch Input is available for this Display
Computer name, domain, and workgroup settings
Computer name: SSSIT-PC
Change settings
Full computer name: SSSIT-PC
Computer description:
Workgroup: WORKGROUP
See also
Action Center
Windows Update
Performance Information and Tools



Click on the new tab of user variable or system variable:



Write JAVA_HOME in variable name and paste the path of jdk folder in variable value:



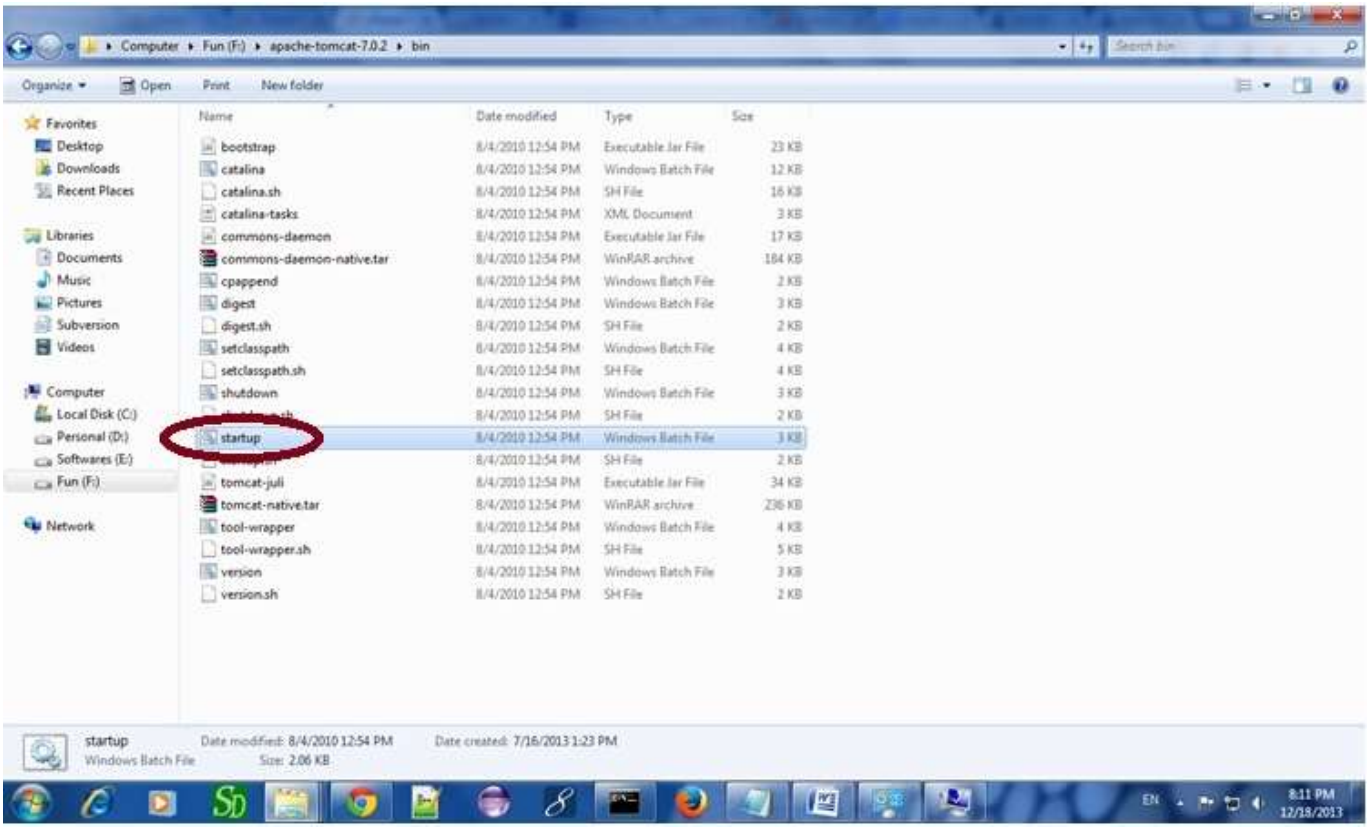
There must not be semicolon (;) at the end of the path.

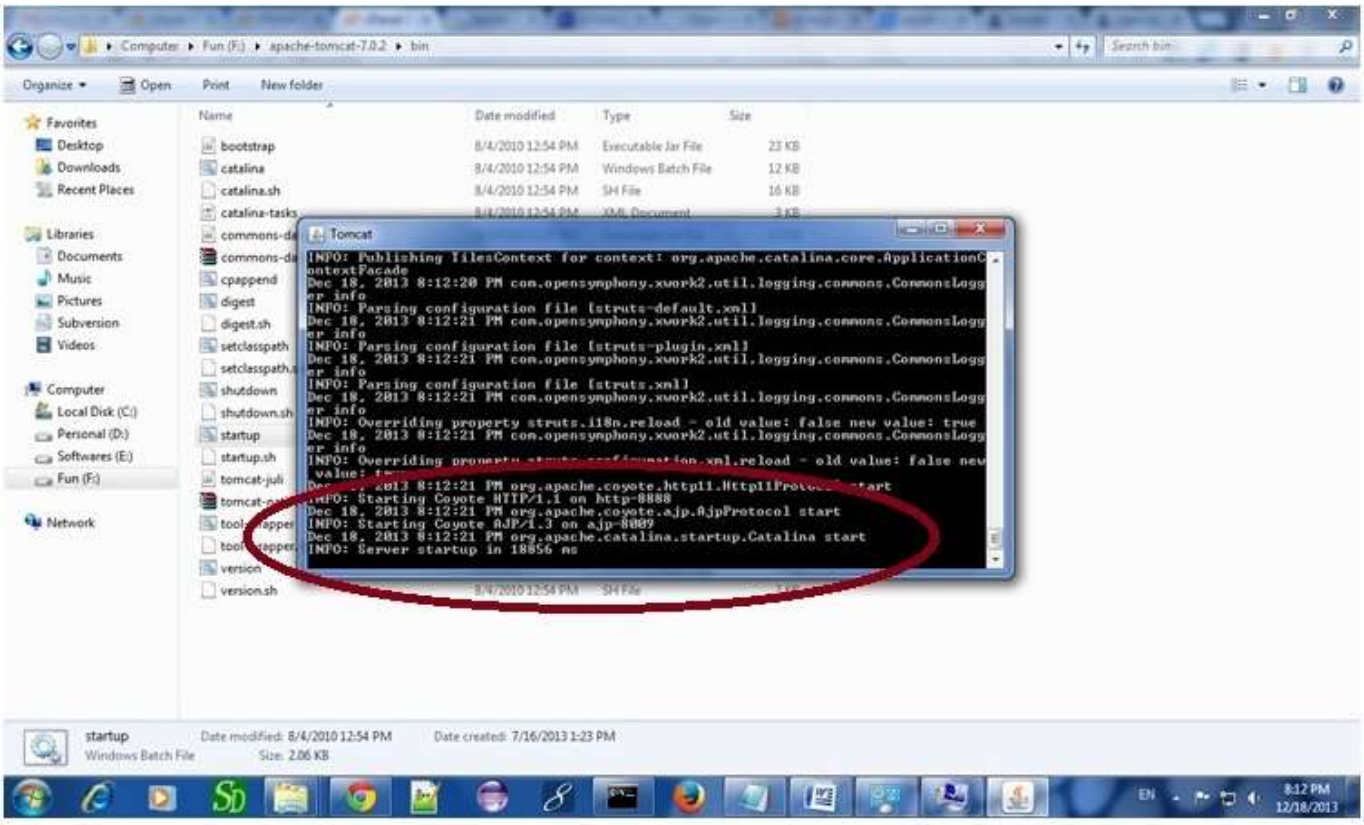
After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

Note: There are two types of tomcat available:

1. Apache tomcat that needs to extract only (no need to install)
2. Apache tomcat that needs to install

It is the example of apache tomcat that needs to extract only.


Computer ▸ Fun (F:) ▸ apache-tomcat-7.0.2 ▸ bin
Organize
Open
Print
New folder
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Subversion
Videos
Computer
Local Disk (C:)
Personal (D:)
Softwares (E:)
Fun (F:)
Network
Name
Date modified
Type
Size
bootstrap 8/4/2010 12:54 PM Executable Jar File 23 KB
catalina 8/4/2010 12:54 PM Windows Batch File 12 KB
catalina.sh 8/4/2010 12:54 PM SH File 16 KB
catalina-tasks 8/4/2010 12:54 PM XML Document 3 KB
commons-daemon 8/4/2010 12:54 PM Executable Jar File 17 KB
commons-daemon-native.tar 8/4/2010 12:54 PM WinRAR archive 184 KB
cpappend 8/4/2010 12:54 PM Windows Batch File 2 KB
digest 8/4/2010 12:54 PM Windows Batch File 3 KB
digest.sh 8/4/2010 12:54 PM SH File 2 KB
setclasspath 8/4/2010 12:54 PM Windows Batch File 4 KB
setclasspath.sh 8/4/2010 12:54 PM SH File 4 KB
shutdown 8/4/2010 12:54 PM Windows Batch File 3 KB
startup 8/4/2010 12:54 PM Windows Batch File 3 KB
tomcat-juli 8/4/2010 12:54 PM Executable Jar File 34 KB
tomcat-native.tar 8/4/2010 12:54 PM WinRAR archive 236 KB
tool-wrapper 8/4/2010 12:54 PM Windows Batch File 4 KB
tool-wrapper.sh 8/4/2010 12:54 PM SH File 5 KB
version 8/4/2010 12:54 PM Windows Batch File 3 KB
version.sh 8/4/2010 12:54 PM SH File 2 KB
startup
Windows Batch File
Date modified: 8/4/2010 12:54 PM
Size: 2.06 KB
Date created: 7/16/2013 1:23 PM
EN
8:11 PM
12/18/2013



Now server is started successfully.

2) How to change port number of apache tomcat

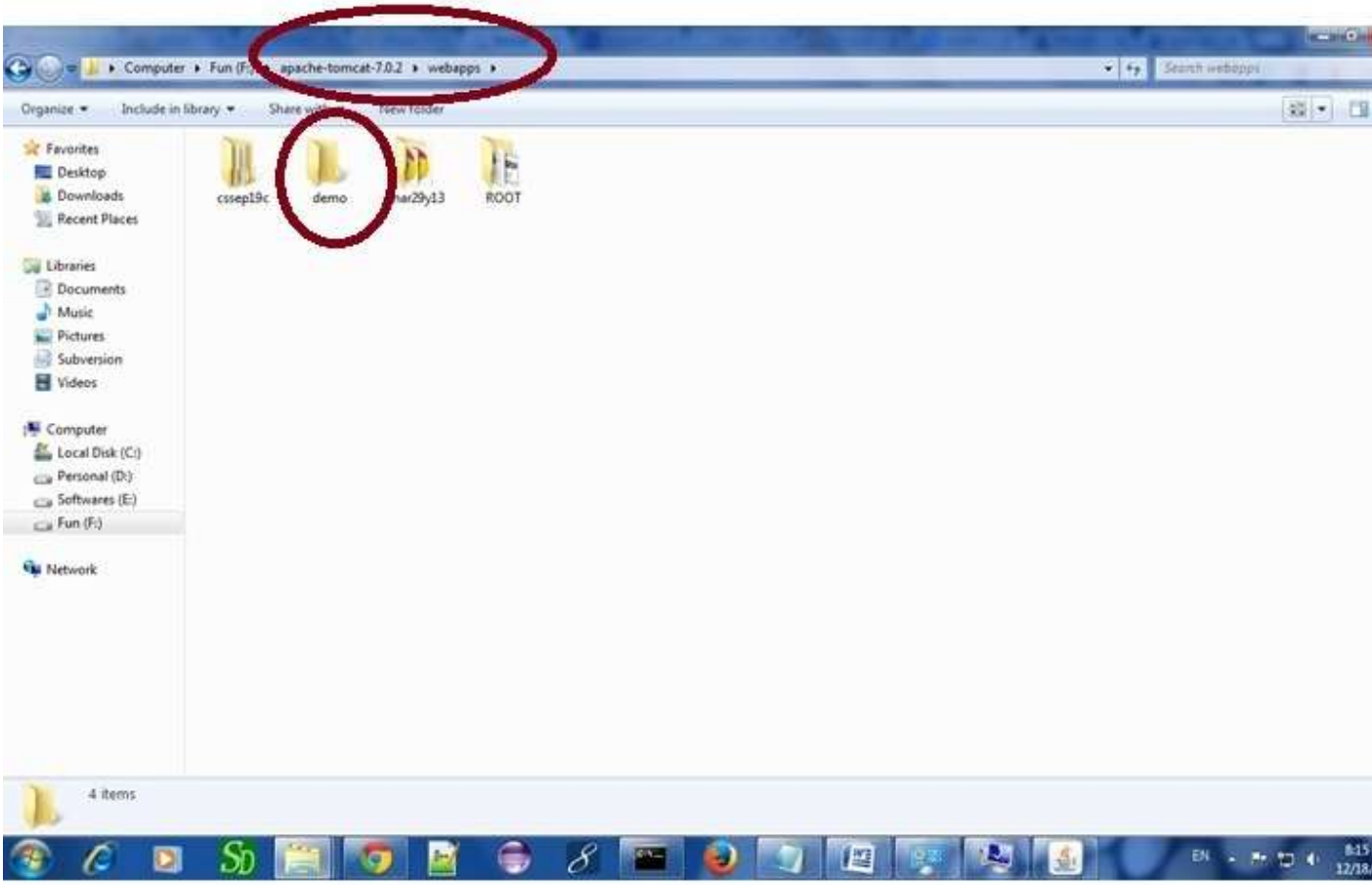
Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

---

5) How to deploy the servlet project

Copy the project and paste it in the webapps folder under apache tomcat.



But there are several ways to deploy the project. They are as follows:

- By copying the context(project) folder into the webapps directory
- By copying the war folder into the webapps directory
- By selecting the folder path from the server
- By selecting the war file from the server

Here, we are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

```
1. projectfolder> jar cvf myproject.war *
```

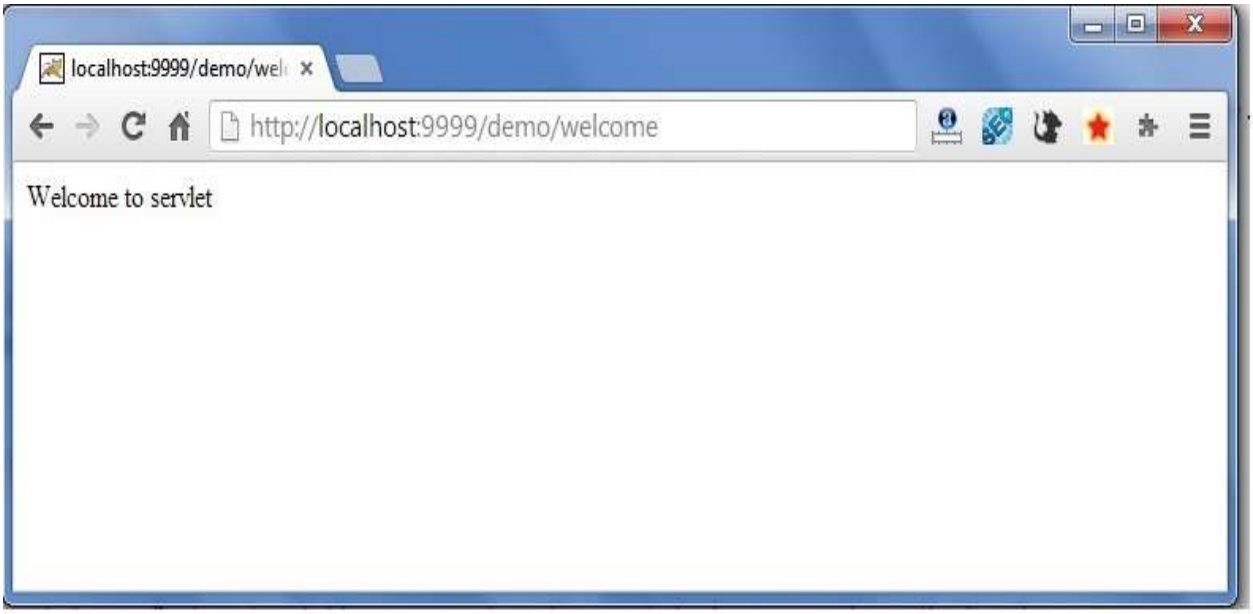
Creating war file has an advantage that moving the project from one location to another takes less time.

---

6) How to access the servlet

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

```
1. http://localhost:9999/demo/welcome
```



ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

Example of ServletRequest to display the name of the user

In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

index.html

```
1. <form action="welcome" method="get">
2. Enter your name<input type="text" name="name"><br>
3. <input type="submit" value="login">
4. </form>
```

DemoServ.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
```

```
4.  public class DemoServ extends HttpServlet{
5.  public void doGet(HttpServletRequest req,HttpServletResponse res)
6.  throws ServletException,IOException
7.  {
8.  res.setContentType("text/html");
9.  PrintWriter pw=res.getWriter();
10.
11. String name=req.getParameter("name");//will return value
12. pw.println("Welcome "+name);
13.
14. pw.close();
15. }}
```

download this example (developed without IDE)
download this example (developed using Eclipse IDE)
download this example (developed using Netbeans IDE)

Other examples of ServletRequest interface

Example of ServletRequest to display all the header information

In this example, we are displaying the header information of the servlet such as content type, content length, user agent etc.

RequestDispatcher in Servlet

1. RequestDispatcher Interface
2. Methods of RequestDispatcher interface
1. forward method
2. include method
3. How to get the object of RequestDispatcher
4. Example of RequestDispatcher interface

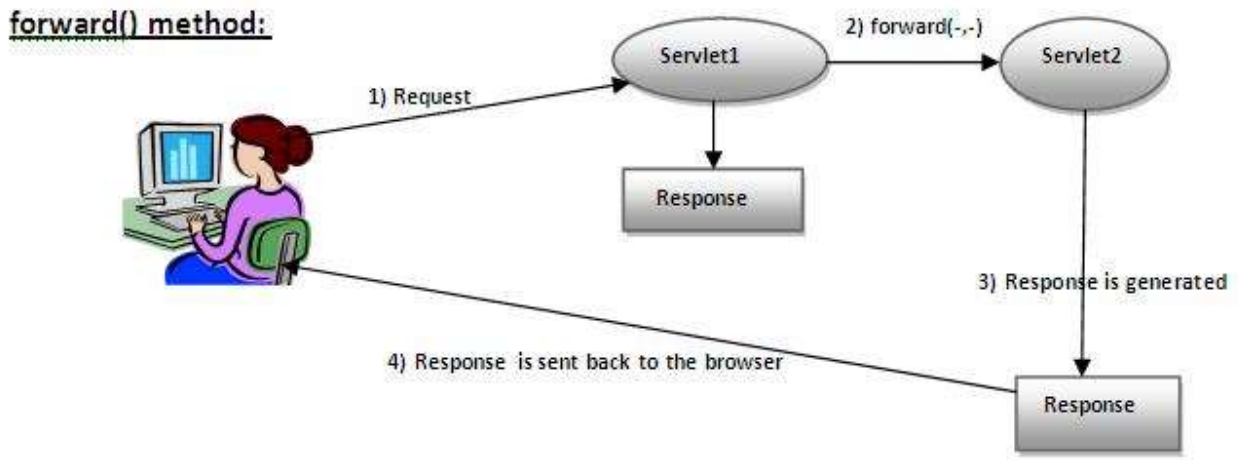
The RequestDispacher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.This interface can also be used to include the content of antoher resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

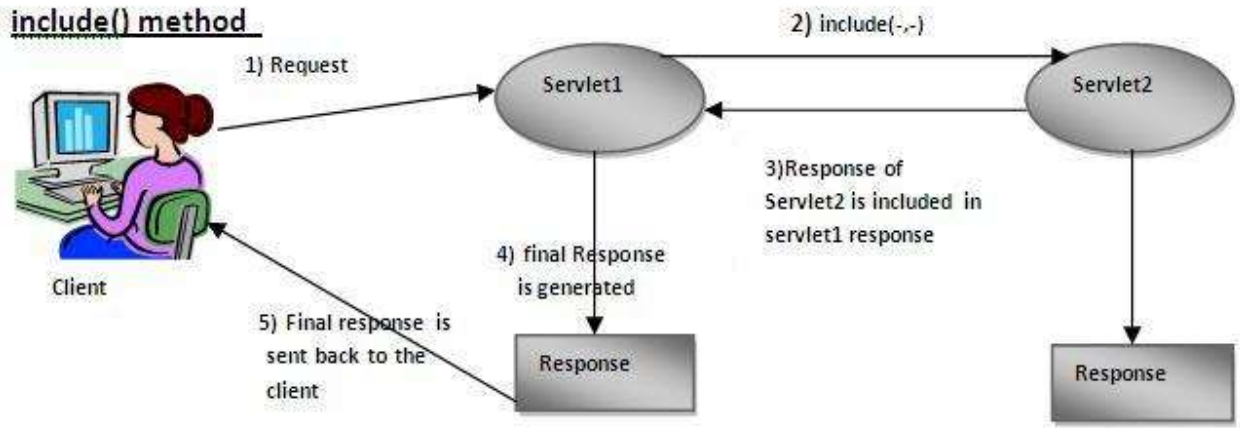
Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

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.
2. **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.



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

How to get the object of RequestDispatcher

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

Syntax of getRequestDispatcher method

```
1. public RequestDispatcher getRequestDispatcher(String resource);
```

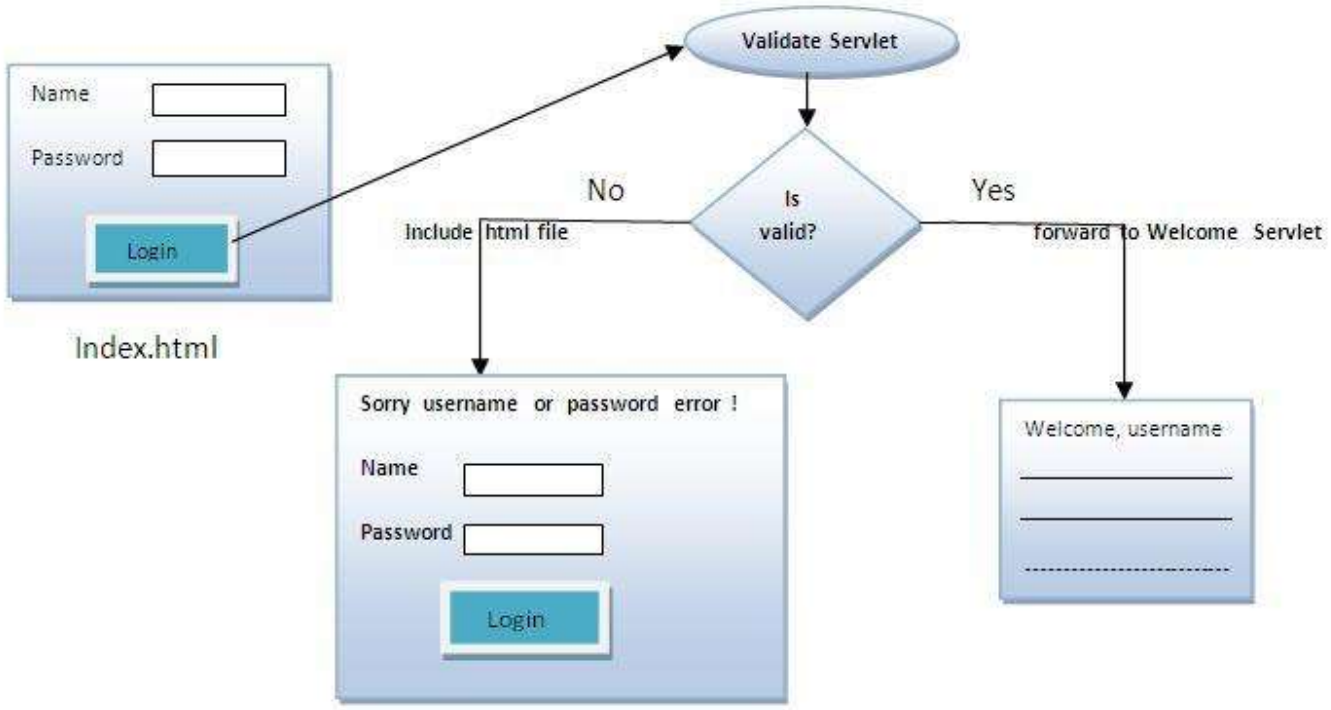
Example of using getRequestDispatcher method

```
1. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
2. //servlet2 is the url-pattern of the second servlet
3.
4. rd.forward(request, response);//method may be include or forward
```

Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servlet, 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:

- **index.html file:** for getting input from the user.
- **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
- **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
- **web.xml file:** a deployment descriptor file that contains the information about the servlet.



---

index.html

```
1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. Password:<input type="password" name="userPass"/><br/>
4. <input type="submit" value="login"/>
5. </form>
```

Login.java

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


public class Login extends HttpServlet {

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

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

    String n=request.getParameter("userName");
    String p=request.getParameter("userPass");

    if(p.equals("servlet"){
        RequestDispatcher rd=request.getRequestDispatcher("servlet2");
        rd.forward(request, response);
    }
    else{
        out.print("Sorry UserName or Password Error!");
        RequestDispatcher rd=request.getRequestDispatcher("/index.html");
        rd.include(request, response);

        }
    }

}
```

WelcomeServlet.java

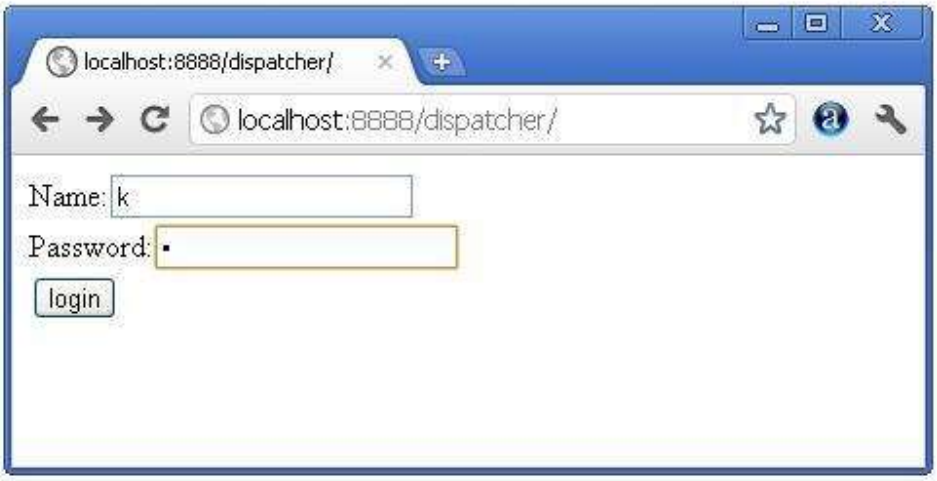
```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WelcomeServlet extends HttpServlet {

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

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

    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    }

}
```

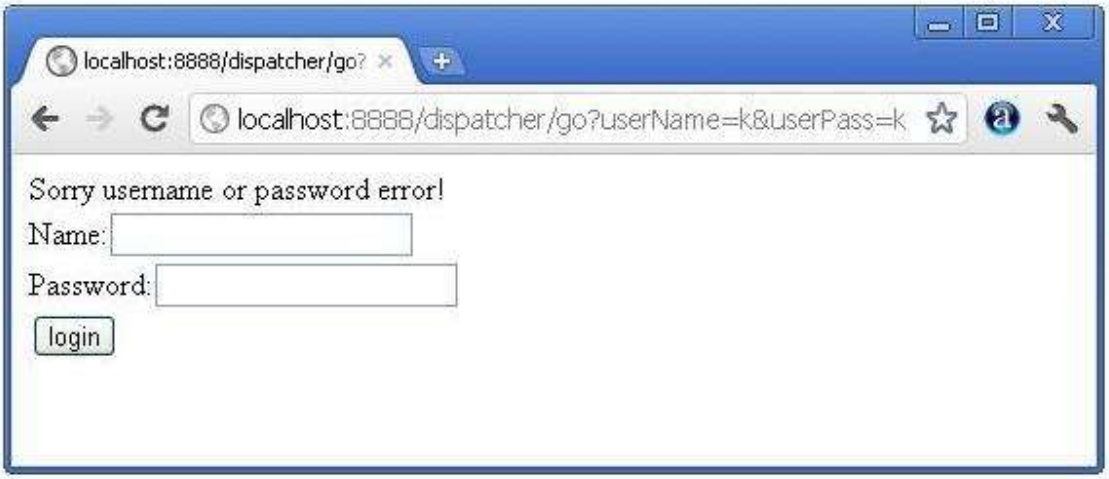
---

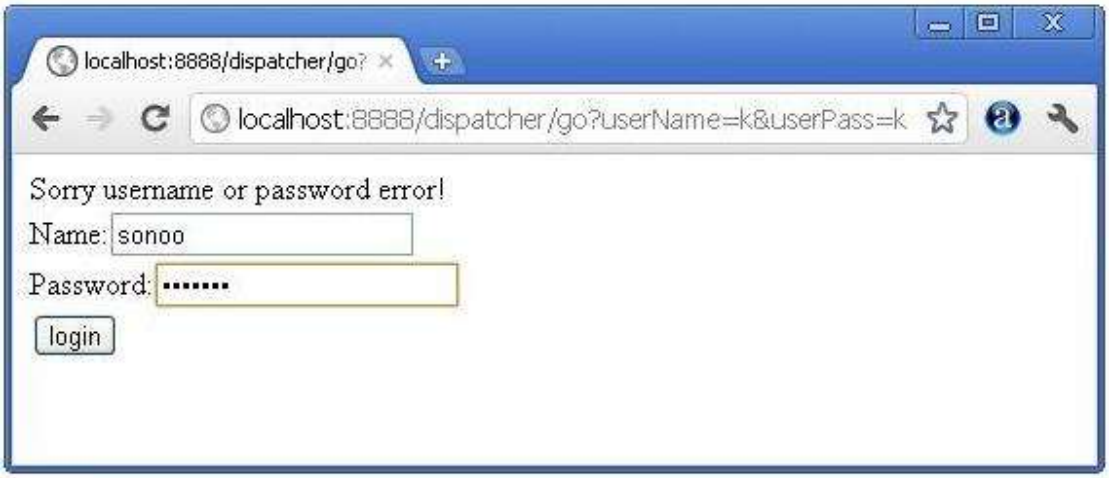
web.xml

```
1.  <web-app>
2.   <servlet>
3.     <servlet-name>Login</servlet-name>
4.     <servlet-class>Login</servlet-class>
5.    </servlet>
6.    <servlet>
7.     <servlet-name>WelcomeServlet</servlet-name>
8.     <servlet-class>WelcomeServlet</servlet-class>
9.    </servlet>
10.
11.
12.  <servlet-mapping>
13.    <servlet-name>Login</servlet-name>
14.    <url-pattern>/servlet1</url-pattern>
15.  </servlet-mapping>
16.  <servlet-mapping>
17.    <servlet-name>WelcomeServlet</servlet-name>
18.    <url-pattern>/servlet2</url-pattern>
19.  </servlet-mapping>
20.
21.  <welcome-file-list>
22.   <welcome-file>index.html</welcome-file>
23.  </welcome-file-list>
24. </web-app>
```

download this example
download this example (developed in Myeclipse IDE)
download this example (developed in eclipse IDE)
download this example (developed in netbeans IDE)


localhost:8888/dispatcher/
localhost:8888/dispatcher/
Name: k
Password:
login


localhost:8888/dispatcher/go?
localhost:8888/dispatcher/go?userName=k&userPass=k
Sorry username or password error!
Name:
Password:
login


localhost:8888/dispatcher/go?
localhost:8888/dispatcher/go?userName=k&userPass=k
Sorry username or password error!
Name: sonoo
Password:
login

SendRedirect in servlet

1. sendRedirect method
2. Syntax of sendRedirect() method
3. Example of RequestDispatcher interface

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 bar of the browser to make another request. So, it can work inside and outside the server.

Difference between forward() and sendRedirect() method

There are many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface. They are given below:

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method w at client side.
It sends the same request and response objects to another servlet.	It always sends a new reques

It can work within the server only.	It can be used within and outs the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servl

Syntax of sendRedirect() method

```
1. public void sendRedirect(String URL)throws IOException;
```

Example of sendRedirect() method

```
1. response.sendRedirect("http://www.javatpoint.com");
```

Full example of sendRedirect method in servlet

In this example, 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.

DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class DemoServlet extends HttpServlet{
6. public void doGet(HttpServletRequest req,HttpServletResponse res)
7. throws ServletException,IOException
8. {
9. res.setContentType("text/html");
10. PrintWriter pw=res.getWriter();
11.
12. response.sendRedirect("http://www.google.com");
13.
14. pw.close();
15. }}
```

Creating custom google search using sendRedirect

In this example, we are using sendRedirect method to send request to google server with the request data.

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>sendRedirect example</title>
</head>
<body>


<form action="MySearcher">
<input type="text" name="name">
<input type="submit" value="Google Search">
</form>

</body>
</html>
```

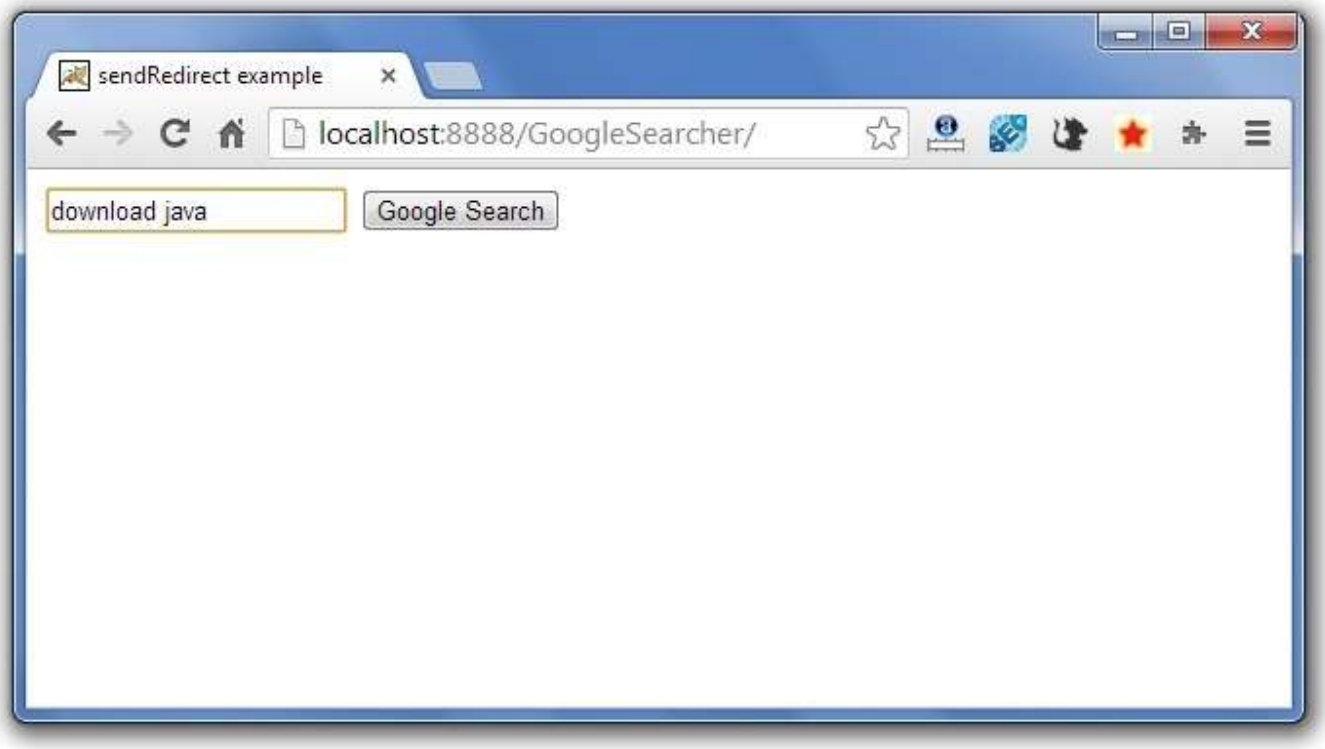
MySearcher.java

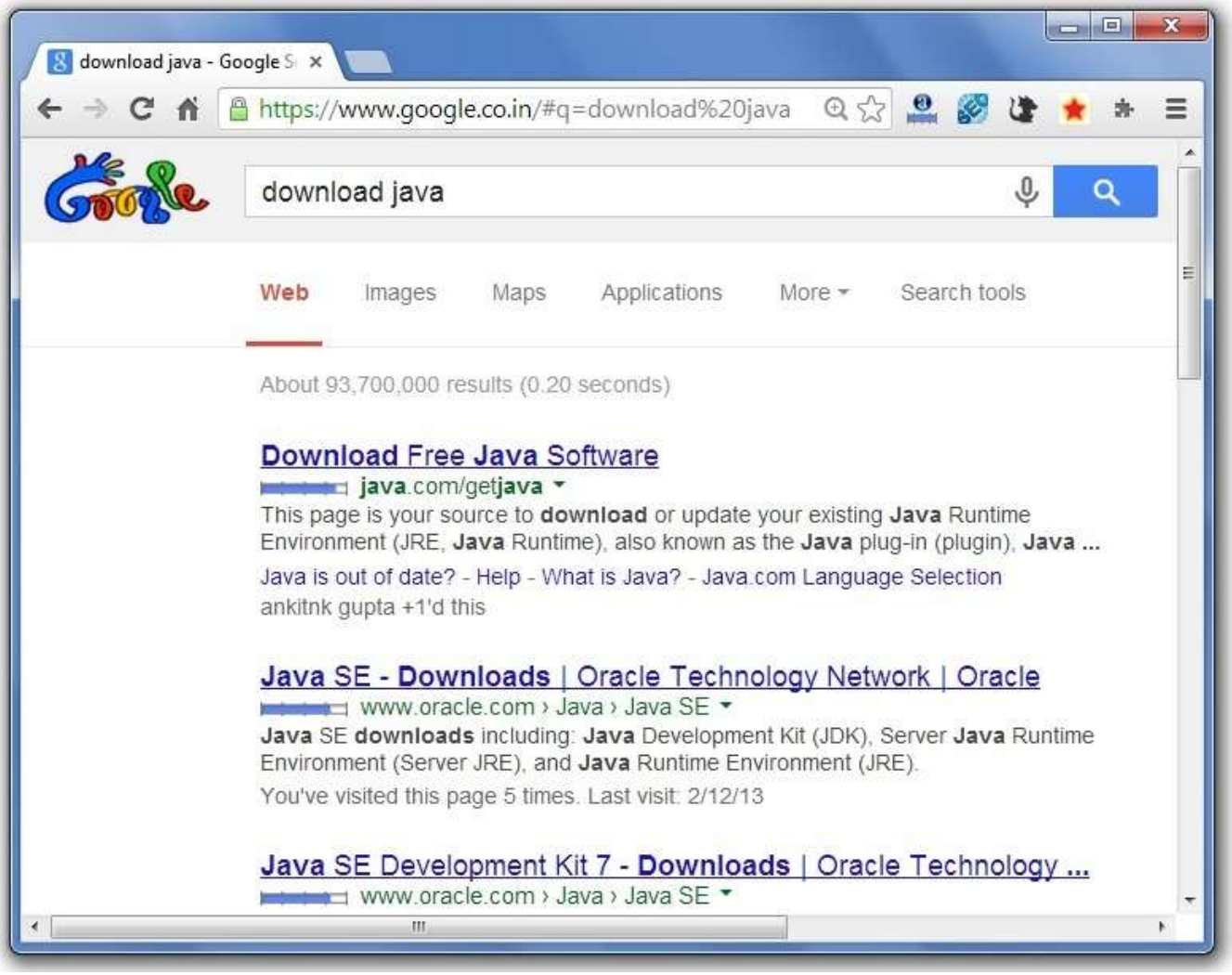
```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MySearcher extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String name=request.getParameter("name");
        response.sendRedirect("https://www.google.co.in/#q="+name);
    }
}
```

---

download this example (developed in Eclipse)

Output


sendRedirect example
localhost:8888/GoogleSearcher/
download java
Google Search



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.

Advantage of ServletConfig

The core 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.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

Syntax of getServletConfig() method

```
1. public ServletConfig getServletConfig();
```

Example of getServletConfig() method

```
1. ServletConfig config=getServletConfig();
2. //Now we can call the methods of ServletConfig interface
```

Syntax to provide the initialization parameter for a servlet

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

```
1. <web-app>
2.   <servlet>
3.     ......
4.
5.     <init-param>
6.       <param-name>parametername</param-name>
7.       <param-value>parametervalue</param-value>
8.     </init-param>
9.     ......
10.  </servlet>
11. </web-app>
```

Example of ServletConfig to get initialization parameter

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class DemoServlet extends HttpServlet {
6. public void doGet(HttpServletRequest request, HttpServletResponse response)
7.    throws ServletException, IOException {
8.
9.    response.setContentType("text/html");
10.   PrintWriter out = response.getWriter();
11.
12.   ServletConfig config=getServletConfig();
13.   String driver=config.getInitParameter("driver");
14.   out.print("Driver is: "+driver);
15.
16.   out.close();
17.   }
18.
19. }
```

web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>DemoServlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6.
7. <init-param>
8. <param-name>driver</param-name>
9. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
10. </init-param>
11.
12. </servlet>
13.
14. <servlet-mapping>
15. <servlet-name>DemoServlet</servlet-name>
16. <url-pattern>/servlet1</url-pattern>
17. </servlet-mapping>
18.
19. </web-app>
```

download this example (developed in Myeclipse IDE)
download this example(developed in Eclipse IDE)
download this example(developed in Netbeans IDE)

Example of ServletConfig to get all the initialization parameters

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

DemoServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3. import java.util.Enumeration;
4.
5. import javax.servlet.ServletConfig;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10.
11.
12. public class DemoServlet extends HttpServlet {
13. public void doGet(HttpServletRequest request, HttpServletResponse response)
14.       throws ServletException, IOException {
15.
16.    response.setContentType("text/html");
17.    PrintWriter out = response.getWriter();
18.
19.    ServletConfig config=getServletConfig();
20.    Enumeration<String> e=config.getInitParameterNames();
21.
22.    String str="";
23.    while(e.hasMoreElements()){
24.    str=e.nextElement();
25.    out.print("<br>Name: "+str);
26.    out.print(" value: "+config.getInitParameter(str));
27.    }
28.
29.    out.close();
30. }
31.
32. }
```

web.xml

```
1. <web-app>
```

```
2.
3.  <servlet>
4.  <servlet-name>DemoServlet</servlet-name>
5.  <servlet-class>DemoServlet</servlet-class>
6.
7.  <init-param>
8.  <param-name>username</param-name>
9.  <param-value>system</param-value>
10. </init-param>
11.
12. <init-param>
13. <param-name>password</param-name>
14. <param-value>oracle</param-value>
15. </init-param>
16.
17. </servlet>
18.
19. <servlet-mapping>
20. <servlet-name>DemoServlet</servlet-name>
21. <url-pattern>/servlet1</url-pattern>
22. </servlet-mapping>
23.
24. </web-app>
```

ServletContext Interface

1. ServletContext Interface
2. Usage of ServletContext Interface
3. Methods of ServletContext interface
4. How to get the object of ServletContext
5. Syntax to provide the initialization parameter in Context scope
6. Example of ServletContext to get initialization parameter
7. Example of ServletContext to get all the initialization parameter

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the**<context-param>** element.

Advantage of ServletContext

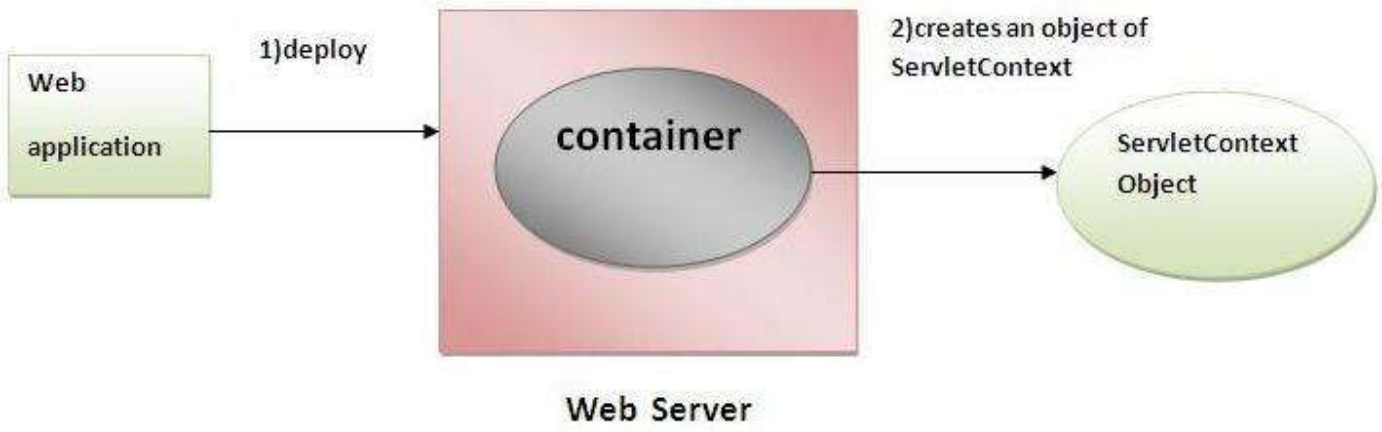
Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the

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.



Commonly used methods of ServletContext interface

There is given some commonly used methods of ServletContext interface.

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
3. **public void setAttribute(String name,Object object):**sets the given object in the application scope.
4. **public Object getAttribute(String name):**Returns the attribute for the specified name.
5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the

given name from the servlet context.

How to get 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.

Syntax of getServletContext() method

```
1. public ServletContext getServletContext()
```

Example of getServletContext() method

```
1. //We can get the ServletContext object from ServletConfig object
2. ServletContext application=getServletConfig().getServletContext();
3.
4. //Another convenient way to get the ServletContext object
5. ServletContext application=getServletContext();
```

Syntax to provide the initialization parameter in Context scope

The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value.

```
1. <web-app>
2.  ......
3.
4.   <context-param>
5.     <param-name>parametername</param-name>
6.     <param-value>parametervalue</param-value>
7.   </context-param>
8.  ......
9. </web-app>
```

Example of ServletContext to get the initialization parameter

In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to

modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example:

DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class DemoServlet extends HttpServlet{
7. public void doGet(HttpServletRequest req,HttpServletResponse res)
8. throws ServletException,IOException
9. {
10. res.setContentType("text/html");
11. PrintWriter pw=res.getWriter();
12.
13. //creating ServletContext object
14. ServletContext context=getServletContext();
15.
16. //Getting the value of the initialization parameter and printing it
17. String driverName=context.getInitParameter("dname");
18. pw.println("driver name is="+driverName);
19.
20. pw.close();
21.
22. }}
```

web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7.
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
12.
13. <servlet-mapping>
14. <servlet-name>sonoojaiswal</servlet-name>
15. <url-pattern>/context</url-pattern>
16. </servlet-mapping>
17.
18. </web-app>
```

Example of ServletContext to get all the initialization parameters

In this example, we are getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class.

DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class DemoServlet extends HttpServlet{
7. public void doGet(HttpServletRequest req,HttpServletResponse res)
8. throws ServletException,IOException
9. {
10. res.setContentType("text/html");
11. PrintWriter out=res.getWriter();
12.
13. ServletContext context=getServletContext();
14. Enumeration<String> e=context.getInitParameterNames();
15.
16. String str="";
17. while(e.hasMoreElements()){
18.    str=e.nextElement();
19.    out.print("<br> "+context.getInitParameter(str));
20. }
21. }}
```

web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7.
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
12.
13. <context-param>
14. <param-name>username</param-name>
15. <param-value>system</param-value>
16. </context-param>
17.
18. <context-param>
```

```
<param-name>password</param-name>
<param-value>oracle</param-value>
</context-param>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/context</url-pattern>
</servlet-mapping>

</web-app>
```

Session Tracking in Servlets

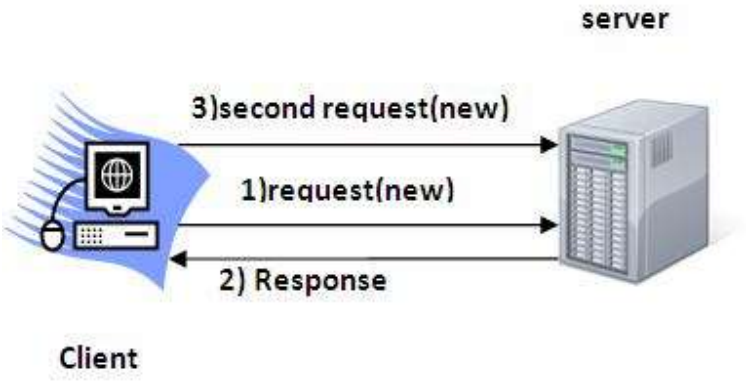
1. Session Tracking
2. Session Tracking Techniques

Session simply means a particular interval of time.

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 stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

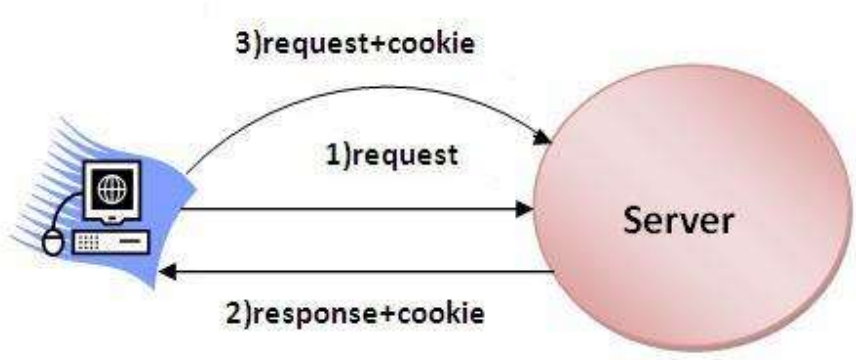
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

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

Persistent cookie

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

Advantage of Cookies

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

Disadvantage of Cookies

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

> **Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.**

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description

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

Useful Methods of Cookie class

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

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

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

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

How to create Cookie?

Let's see the simple code to create cookie.

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

How to delete Cookie?

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

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

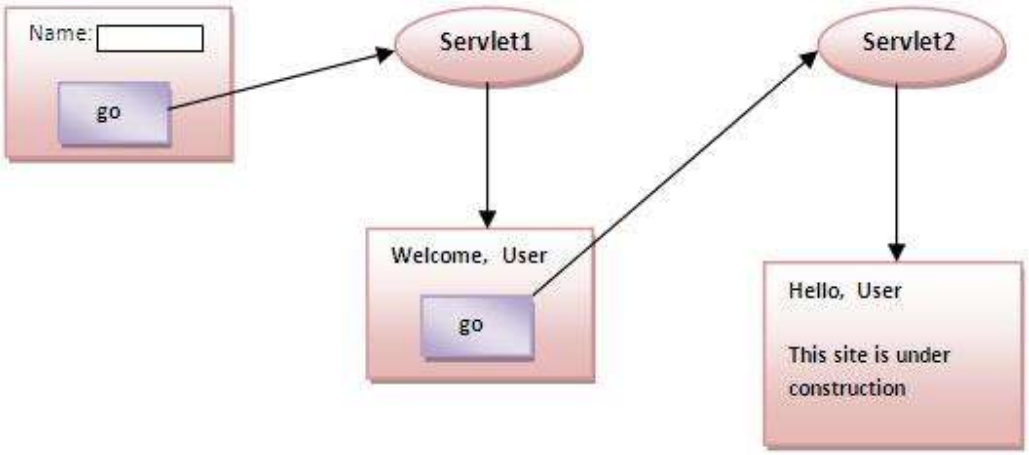
How to get Cookies?

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

```
1. Cookie ck[]=request.getCookies();
2. for(int i=0;i<ck.length;i++){
3.  out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cook
   ie
4. }
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>
```

FirstServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class FirstServlet extends HttpServlet {
7.
8.   public void doPost(HttpServletRequest request, HttpServletResponse response){
9.     try{
10.
11.    response.setContentType("text/html");
12.    PrintWriter out = response.getWriter();
13.
14.    String n=request.getParameter("userName");
15.    out.print("Welcome "+n);
16.
17.    Cookie ck=new Cookie("uname",n);//creating cookie object
18.    response.addCookie(ck);//adding cookie in the response
19.
20.    //creating submit button
21.    out.print("<form action='servlet2'>");
22.    out.print("<input type='submit' value='go'>");
23.    out.print("</form>");
24.
```

```
    out.close();

        }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{

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

    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());

    out.close();

        }catch(Exception e){System.out.println(e);}
    }


}
```

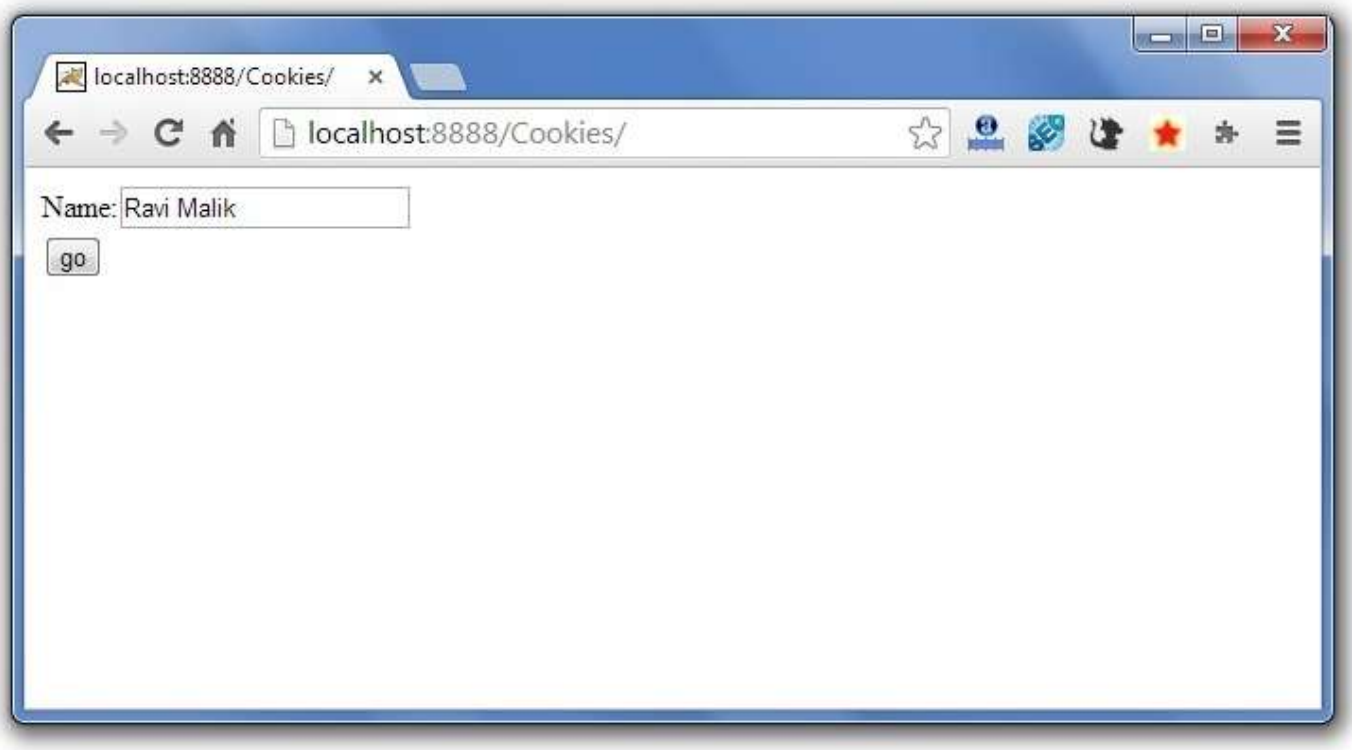
web.xml

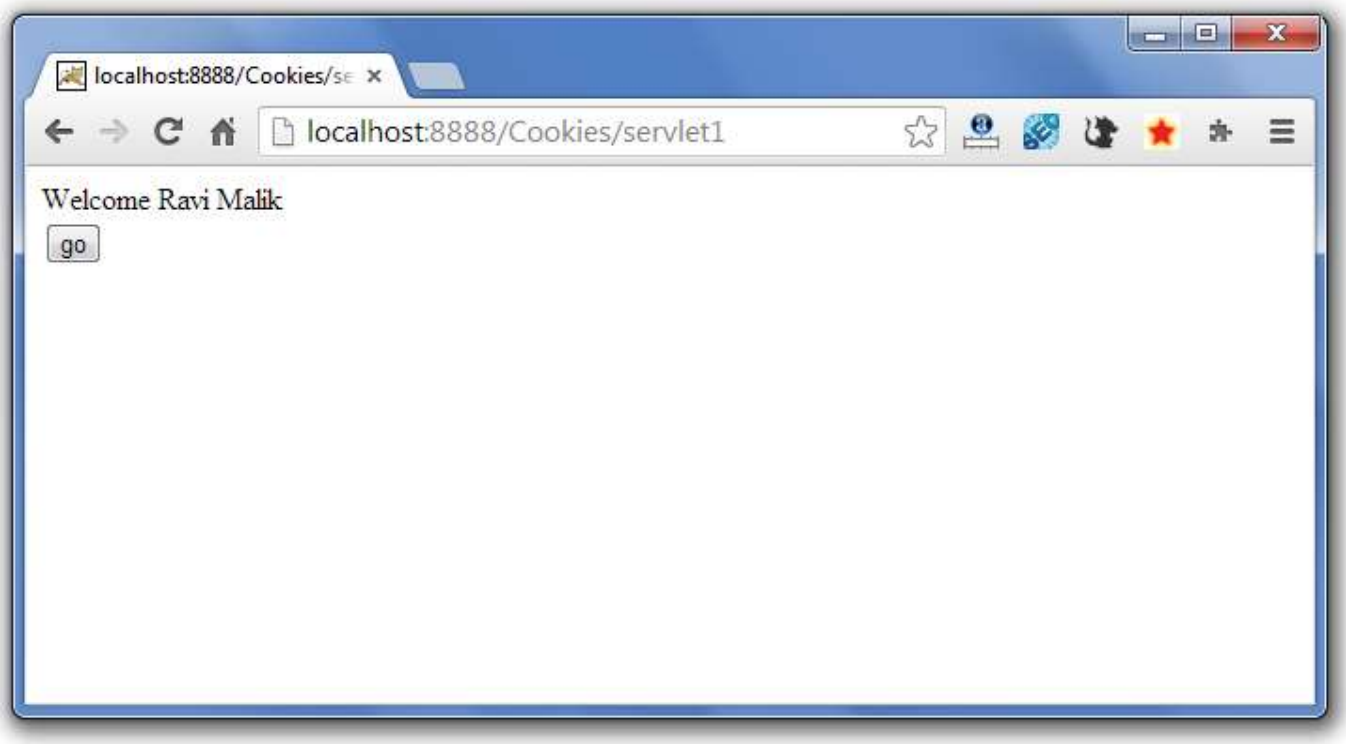
```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
```

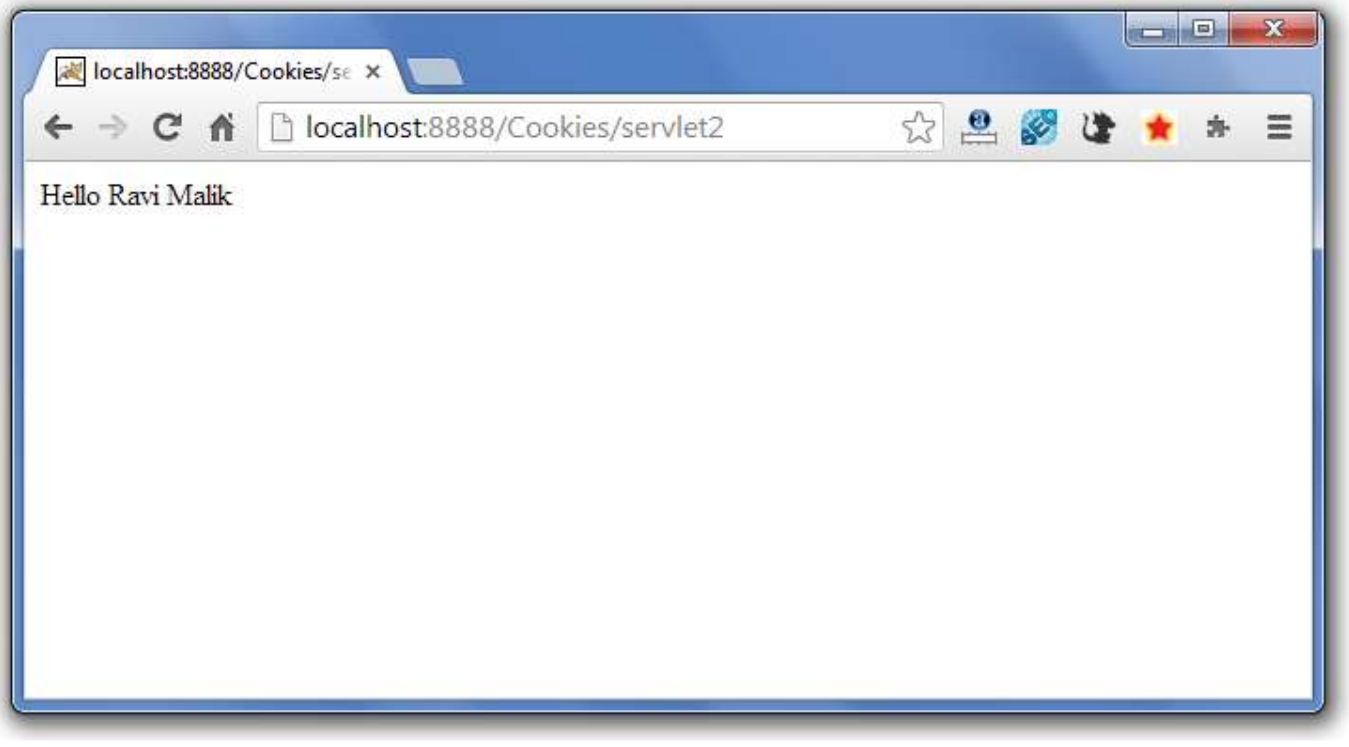
```
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
22.
23. </web-app>
```

download this example (developed using Myeclipse IDE)
download this example (developed using Eclipse IDE)
download this example (developed using Netbeans IDE)

Output







Event and Listener in Servlet

1. Event and Listener in Servlet
2. Event classes
3. Event interfaces

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 tables of the database at time of deploying the project, creating database connection object etc.

There are many Event classes and Listener interfaces in the javax.servlet and javax.servlet.http packages.

Event classes

The event classes are as follows:

1. ServletRequestEvent
2. ServletContextEvent
3. ServletRequestAttributeEvent
4. ServletContextAttributeEvent
5. HttpSessionEvent
6. HttpSessionBindingEvent

Event interfaces

The event interfaces are as follows:

1. ServletRequestListener
2. ServletRequestAttributeListener
3. ServletContextListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSessionBindingListener
8. HttpSessionActivationListener

Upcoming topics in Servlet Events and Listeners

ServletContextEvent

Let's see the simple example of ServletContextEvent and ServletContextListener

HttpSessionEvent

Let's see the simple example of HttpSessionEvent and HttpSessionListener

ServletRequestEvent

Let's see the simple example of ServletRequestEvent and ServletRequestListener

ServletContext AttributeEvent

Let's see the simple example of ServletContextAttributeEvent and ServletContextAttributeListener

HttpSessionBindingEvent

Let's see the simple example of HttpSessionBindingEvent and HttpSessionAttributeListener

ServletRequestAttributeEvent

Let's see the simple example of ServletRequestAttributeEvent and ServletRequestAttributeEvent

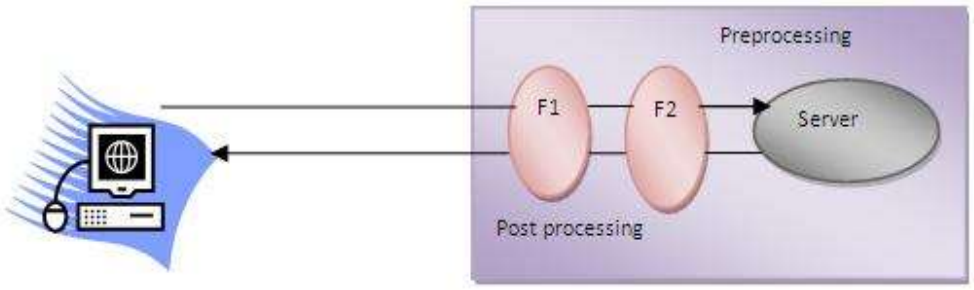
Servlet Filter

A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

So maintenance cost will be less.



Note: Unlike Servlet, One filter doesn't have dependency on another filter.

Usage of Filter

- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- input validation etc.

Advantage of Filter

1. Filter is pluggable.
2. One filter don't have dependency onto another resource.
3. Less Maintenance

Filter API

Like servlet filter have its own API. The javax.servlet package contains the three interfaces of Filter API.

1. Filter
2. FilterChain
3. FilterConfig

1) Filter interface

For creating any filter, you must implement the Filter interface. Filter interface provides the life cycle methods for a filter.

Method	Description
public void init(FilterConfig config)	init() method is invoked only once. It is used to initialize the filter.
public void doFilter(HttpServletRequest request,HttpServletResponse response, FilterChain chain)	doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks.
public void destroy()	This is invoked only once when filter is taken out of the service.

2) FilterChain interface

The object of FilterChain is responsible to invoke the next filter or resource in the chain.This object is passed in the doFilter method of Filter interface.The FilterChain interface contains only one method:

1. **public void doFilter(HttpServletRequest request, HttpServletResponse response):** it passes the control to the next filter or resource.

How to define Filter

We can define filter same as servlet. Let's see the elements of filter and filter-mapping.

```
<web-app>

<filter>
<filter-name>...</filter-name>
<filter-class>...</filter-class>
</filter>

<filter-mapping>
<filter-name>...</filter-name>
<url-pattern>...</url-pattern>
</filter-mapping>

</web-app>
```

For mapping filter we can use, either url-pattern or servlet-name. The url-pattern elements has an advantage over servlet-name element i.e. it can be applied on servlet, JSP or HTML.

Simple Example of Filter

In this example, we are simply displaying information that filter is invoked automatically after the post processing of the request.

index.html

```
1. <a href="servlet1">click here</a>
```

MyFilter.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3.
4. import javax.servlet.*;
5.
6. public class MyFilter implements Filter{
7.
8. public void init(FilterConfig arg0) throws ServletException {}
9.
10. public void doFilter(ServletRequest req, ServletResponse resp,
11.     FilterChain chain) throws IOException, ServletException {
12.
13.     PrintWriter out=resp.getWriter();
14.     out.print("filter is invoked before");
15.
16.     chain.doFilter(req, resp);//sends request to next resource
17.
18.     out.print("filter is invoked after");
19.     }
20.     public void destroy() {}
21. }
```

HelloServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3.
4. import javax.servlet.ServletException;
5. import javax.servlet.http.*;
6.
7. public class HelloServlet extends HttpServlet {
8.     public void doGet(HttpServletRequest request, HttpServletResponse response)
9.             throws ServletException, IOException {
```

```
10.
11.         response.setContentType("text/html");
12.         PrintWriter out = response.getWriter();
13.
14.         out.print("<br>welcome to servlet<br>");
15.
16.     }
17.
18. }
```

web.xml

For defining the filter, filter element of web-app must be defined just like servlet.

```
1.  <web-app>
2.
3.  <servlet>
4.  <servlet-name>s1</servlet-name>
5.  <servlet-class>HelloServlet</servlet-class>
6.  </servlet>
7.
8.  <servlet-mapping>
9.  <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
12.
13. <filter>
14. <filter-name>f1</filter-name>
15. <filter-class>MyFilter</filter-class>
16. </filter>
17.
18. <filter-mapping>
19. <filter-name>f1</filter-name>
20. <url-pattern>/servlet1</url-pattern>
21. </filter-mapping>
22.
23.
24. </web-app>
```

Authentication Filter

We can perform authentication in filter. Here, we are going to check to password given by the user in filter class, if given password is admin, it will forward the request to the WelcomeAdmin servlet otherwise it will display error message.

Example of authenticating user using filter

Let's see the simple example of authenticating user using filter.

Here, we have created 4 files:

- index.html

- MyFilter.java
- AdminServlet.java
- web.xml

index.html

```
<form action="servlet1">
Name:<input type="text" name="name"/><br/>
Password:<input type="password" name="password"/><br/>

<input type="submit" value="login">

</form>
```

MyFilter.java

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

public class MyFilter implements Filter{

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp,
        FilterChain chain) throws IOException, ServletException {

    PrintWriter out=resp.getWriter();

    String password=req.getParameter("password");
    if(password.equals("admin")){
    chain.doFilter(req, resp);//sends request to next resource
    }
    else{
    out.print("username or password error!");
    RequestDispatcher rd=req.getRequestDispatcher("index.html");
    rd.include(req, resp);
    }

}
    public void destroy() {}

}
```

AdminServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.*;

public class AdminServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
9.        throws ServletException, IOException {
10.
11.       response.setContentType("text/html");
12.       PrintWriter out = response.getWriter();
13.
14.       out.print("welcome ADMIN");
15.       out.close();
16.    }
17. }
```

web.xml

```
1.  <web-app>
2.   <servlet>
3.     <servlet-name>AdminServlet</servlet-name>
4.     <servlet-class>AdminServlet</servlet-class>
5.    </servlet>
6.
7.    <servlet-mapping>
8.     <servlet-name>AdminServlet</servlet-name>
9.     <url-pattern>/servlet1</url-pattern>
10.   </servlet-mapping>
11.
12.  <filter>
13.   <filter-name>f1</filter-name>
14.   <filter-class>MyFilter</filter-class>
15.   </filter>
16.   <filter-mapping>
17.   <filter-name>f1</filter-name>
18.   <url-pattern>/servlet1</url-pattern>
19.   </filter-mapping>
20.
21. </web-app>
```

FilterConfig

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

Methods of FilterConfig interface

There are following 4 methods in the FilterConfig interface.

1. **public void init(FilterConfig config):** init() method is invoked only once it is used to initialize the filter.
2. **public String getInitParameter(String parameterName):** Returns the parameter value for the specified parameter name.
3. **public java.util.Enumeration getInitParameterNames():** Returns an enumeration containing all the parameter names.
4. **public ServletContext getServletContext():** Returns the ServletContext object.

Example of FilterConfig

In this example, if you change the param-value to no, request will be forwarded to the servlet otherwise filter will create the response with the message: this page is underprocessing. Let's see the simple example of FilterConfig. Here, we have created 4 files:

- index.html
- MyFilter.java
- HelloServlet.java
- web.xml

index.html

```
1. <a href="servlet1">click here</a>
```

MyFilter.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3.
4. import javax.servlet.*;
5.
6. public class MyFilter implements Filter{
7. FilterConfig config;
8.
9. public void init(FilterConfig config) throws ServletException {
10.    this.config=config;
11. }
12.
13. public void doFilter(ServletRequest req, ServletResponse resp,
14.    FilterChain chain) throws IOException, ServletException {
15.
16.    PrintWriter out=resp.getWriter();
17.
18.    String s=config.getInitParameter("construction");
19.
20.    if(s.equals("yes")){
21.        out.print("This page is under construction");
22.    }
23.    else{
24.        chain.doFilter(req, resp);//sends request to next resource
25.    }
26.
27. }
28. public void destroy() {}
29. }
```

HelloServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3.
4. import javax.servlet.ServletException;
5. import javax.servlet.http.*;
6.
7. public class HelloServlet extends HttpServlet {
8. public void doGet(HttpServletRequest request, HttpServletResponse response)
9.         throws ServletException, IOException {
10.
11.         response.setContentType("text/html");
12.         PrintWriter out = response.getWriter();
13.
14.         out.print("<br>welcome to servlet<br>");
15.
16.     }
17.
18. }
```

---

web.xml

```
1. <web-app>
2.
3.  <servlet>
4.     <servlet-name>HelloServlet</servlet-name>
5.     <servlet-class>HelloServlet</servlet-class>
6.   </servlet>
7.
8.   <servlet-mapping>
9.     <servlet-name>HelloServlet</servlet-name>
10.    <url-pattern>/servlet1</url-pattern>
11.  </servlet-mapping>
12.
13.  <filter>
14.  <filter-name>f1</filter-name>
15.  <filter-class>MyFilter</filter-class>
16.  <init-param>
17.  <param-name>construction</param-name>
18.  <param-value>no</param-value>
19.  </init-param>
20.  </filter>
21.  <filter-mapping>
22.  <filter-name>f1</filter-name>
23.  <url-pattern>/servlet1</url-pattern>
24.  </filter-mapping>
25.
26.
27. </web-app>
```