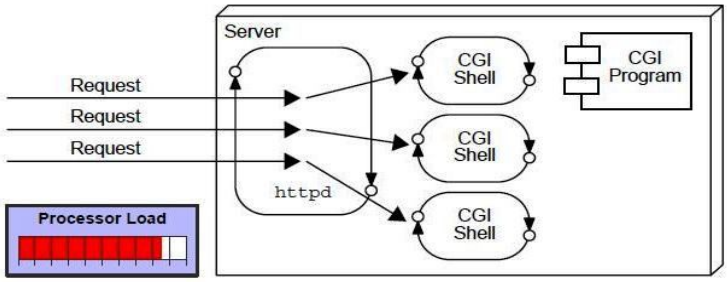
8. JAVA SERVLETS

Servlets provide a *component-based, platform-independent method* for building Web-based applications, without the performance limitations of CGI programs. Servlets have *access to the entire family of Java APIs, including the JDBC API* to access enterprise databases.

CGI (Common 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.



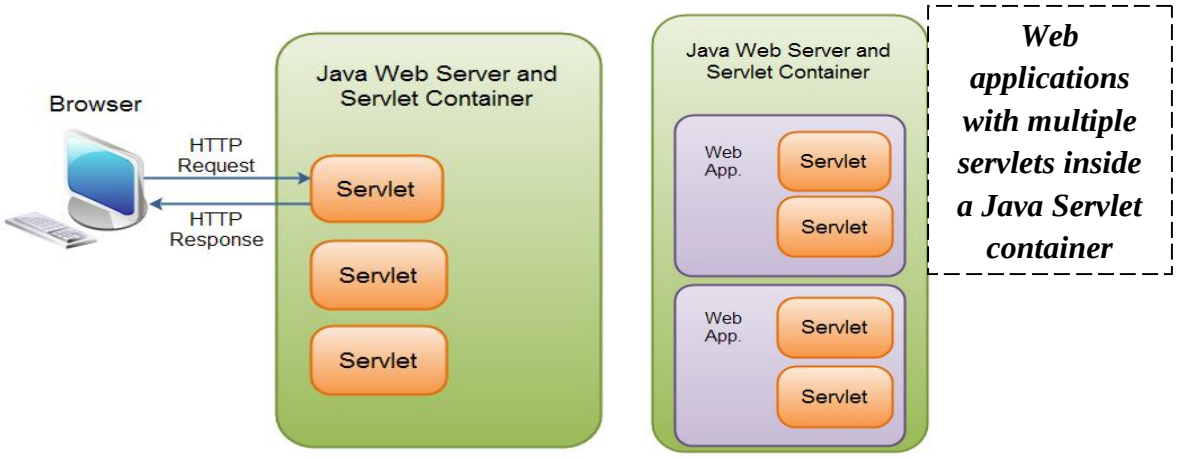
Disadvantages of CGI

There are many problems in CGI technology:

- If number of client's increases, it takes more time for sending response.
- For each request, it starts a process and Web server is limited to start processes.
- It uses platform dependent language e.g. C, C++, perl.

What is Servlet?

A Java Servlet is a Java object that responds to HTTP requests. It runs inside a Servlet container.

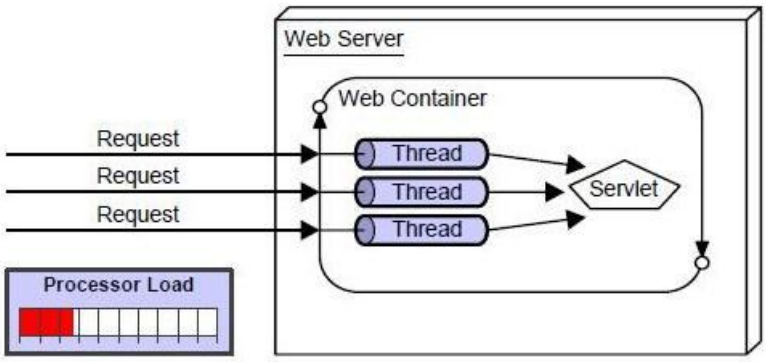


A Servlet is part of a Java web application. A Servlet container may run multiple web applications at the same time, each having multiple servlets running inside.

Java Servlets allow you to develop programs that are executed when a request is made from web client. Java Servlet is a Java class that is loaded into memory of web server. Its methods are called by web server and get executed on web server (Servlet Container). Servlets are totally managed by Web Server and run inside JVM of the Web Server. They are invoked by client using a URL. Servlet is a web component that is deployed on the server to create dynamic web page.

Advantage of Servlets

- 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:

- **better performance:** because it creates a thread for each request not process.
- **Portability:** because it uses java language.
- **Robust:** Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
- **Secure:** because it uses java language…

Http Request and Http Response

- The browser sends an HTTP request to the Java web server.
- The web server checks if the request is for a servlet. If it is, the servlet container is passed the request.
- The servlet container will then find out which servlet the request is for, and activate that servlet.
- Once the servlet has been activated, the servlet processes the request, and generates a response. The response is then sent back to the browser.

Http Protocol

- HTTP is an application protocol implemented on TCP/IP.
- It is a request and response protocol.

- Communication between a server (host) and a client occurs, via a request/response pair. The client initiates an HTTP request message, which is serviced through a HTTP response message in return.

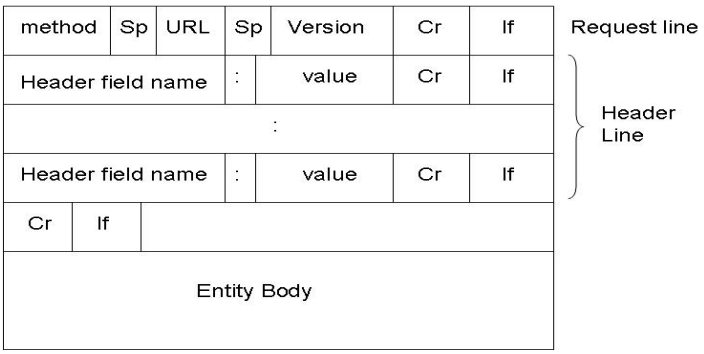
HTTP Headers (Message Formats)

There are two types of messages that HTTP uses –

1. Request message
2. Response message

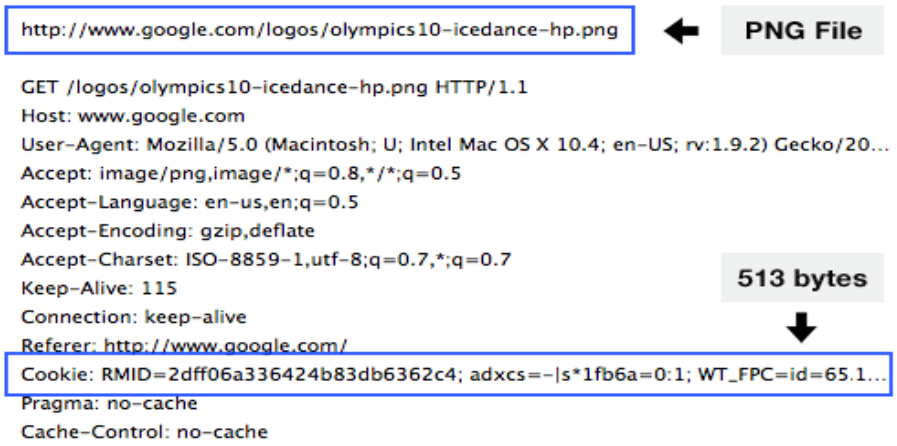
Http Request Header

The request line has three parts, separated by spaces: a method name, the local path of the requested resource and version of HTTP being used. The message format is in ASCII so that it can be read by the humans.



For e.g.:

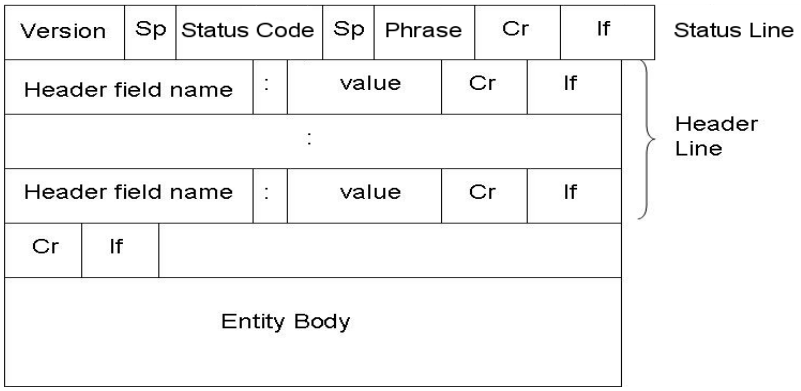
GET /path/to/the/file.html HTTP/1.0 [*Request Line*]

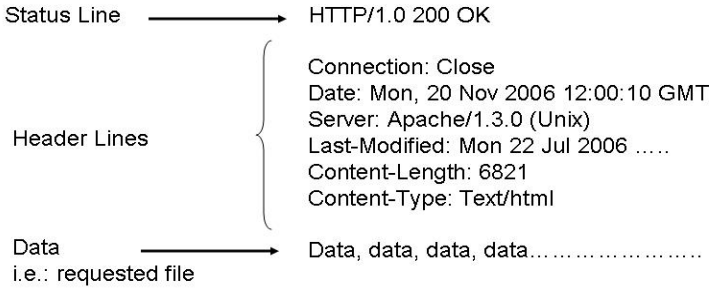


- **GET request vs. POST request**
  - An Http request can either of type GET/POST.
  - In case of GET (default) request, data passed from HTML form is passed along with the URL of the request. The following URL contains data passed from client using GET request:
    `http://localhost:8000/demo/list?price=500&qty=10`
  - The amount of data that can be passed is limited in GET method.
  - In case of POST request, data is passed along with body of HTTP request. You can specify the type of request using method attribute of <form> tag.

Http Response Header

The HTTP message response line also has three parts separated by spaces: the HTTP version, a response status code giving result of the request and English phrase of the status code. This first line is also called as Status line.





HTTP Response Status Codes

Status-codes 2xx - Success

- **200** OK
- **201** POST command successful
- **202** Request accepted
- **203** GET or HEAD request fulfilled
- **204** No content

Status-codes 3xx - Redirection

- **300** Resource found at multiple locations
- **301** Resource moved permanently
- **302** Resource moved temporarily
- **304** Resource has not modified (since date)

Status-codes 4xx - Client error

- 400 Bad request from client
- 401 Unauthorized request
- 402 Payment required for request
- 403 Resource access forbidden
- 404 Resource not found
- 405 Method not allowed for resource
- 406 Resource type not acceptable

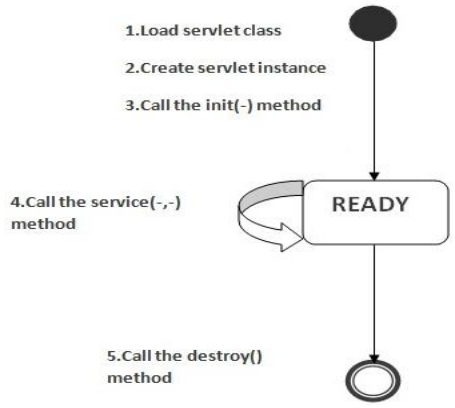
Status-codes 5xx - Server error

- 500 Internal server error
- 501 Method not implemented
- 502 Bad gateway or server overload
- 503 Service unavailable / gateway timeout
- 504 Secondary gateway / server timeout

Servlet Life Cycle

The life cycle of a servlet is controlled by the container in which the servlet is deployed. When a request is mapped to a servlet, the container performs the following steps:

1. Load Servlet Class.
2. Create Instance of Servlet.
3. Call the servlets init() method.
4. Call the servlets service() method.
5. Call the servlets destroy() method.



Step 1, 2 and 3 are executed only once, when the servlet is initially loaded. By default the servlet is not loaded until the first request is received for it. You can force the container to load the servlet when the container starts up though.

Step 4 is executed multiple times - once for every HTTP request to the servlet.

Step 5 is executed when the servlet container unloads the servlet.

Life Cycle Steps

- *Load Servlet Class*
  - Before a servlet can be invoked the servlet container must first load its class definition. This is done just like any other class is loaded.
- *Create Instance of Servlet*
  - When the servlet class is loaded, the servlet container creates an instance of the servlet.
  - Typically, only a single instance of the servlet is created, and concurrent requests to the servlet are executed on the same servlet instance. This is really up to the servlet container to decide, though. But typically, there is just one instance.
- *Call the Servlets init() Method*
  - When a servlet instance is created, its init() method is invoked. The init() method allows a servlet to initialize itself before the first request is processed.
  - You can specify init parameters to the servlet in the web.xml file.
- *Call the Servlets service() Method*
  - For every request received to the servlet, the servlets service() method is called.

- For HttpServlet subclasses, one of the goGet(), doPost() etc. methods are typically called.
- As long as the servlet is active in the servlet container, the service() method can be called. Thus, this step in the life cycle can be executed multiple times.

- *Call the Servlets destroy() Method*
  - When a servlet is unloaded by the servlet container, its destroy() method is called. This step is only executed once, since a servlet is only unloaded once.
  - A servlet is unloaded by the container or if the container shuts down, or if the container reloads the whole web application at runtime.

Example: Servlet Program to Display Date and Time

```
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Date d = new Date();
        out.println("<html>");
        out.println("<head><title> Simple Servlet Program Demo );
         out.println(</title></head>");
        out.println("<body><h1>Hello, From Servlet</h1>");
        out.println("Today Date is : " + d + "</body></html>");
    }
}
```

Web.xml

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

Steps to create a new web application using NetBeans IDE

- Start NetBeans
- Select File → Project
- Select **Java Web** under Categories, and select **Web Application** under Projects
- In the next screen, enter **project name** and select the **directory for project**.
- In next screen, select Server to be used, Java EE version and context path – the name of the project by default.
- Select none in the **Frameworks** screen.

Steps to create a new Servlet

1. Open the web application (if required).
2. Right click on the project name to bring up Popup menu.
3. Select **New → Other**
4. Select **Web** under **Categories**
5. Select **Servlet** under **File Types**
6. Enter the name of the servlet – E.g.: SimpleServlet
7. In the next screen, specify the URL pattern to be used to invoke servlet (E.g.: /simple). Also check the option "*Add to Deployment Descriptor web.xml*"
8. Click on Finish.
9. NetBeans creates a class called SimpleServlet.java and places required entries in web.xml
10. Remove the content of the class and write the required logic for the servlet.

Creating a Servlet without any IDE

Steps to show how to create a simple servlet without using any IDE like NetBeans or Eclipse:

The following are the steps to be taken by you to create a simple servlet.

1. Download Tomcat latest version from http://tomcat.apache.org
2. Install Tomcat
3. Go to **webapps** directory of Tomcat and create the following directory structure for **demo** application. Any directory placed with required structure of Java EE web application is treated as a web application by Tomcat.

```
demo
  WEB-INF
    classes
      TestServlet.java
    web.xml
```

Creating Servlet Source Code

The following is the code for **TestServlet.java**. It simply sends a message to browser.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class TestServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,  HttpServletResponse response) throws
IOException, ServletException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><body><h1>Test Servlet </h1></body></html>");
   }
   public void doPost(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException {  doGet(request, response);  } }
```

Deployment Descriptor - web.xml

Create **web.xml** as follows. It is better you copy **web.xml** from some other application in Tomcat **webapps** directory instead of typing it from scratch. Make sure it is placed in WEB-

INF (in uppercase) folder of **demo** application with the following content.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app 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"
   version="2.5">
    <servlet>
      <servlet-name>TestServlet</servlet-name>
      <servlet-class>TestServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>TestServlet</servlet-name>
        <url-pattern>/test</url-pattern>
    </servlet-mapping>
</web-app>
```

TestServlet is associated with url pattern **/test** in web.xml. From the client, we have to make request for url pattern /test.

Compiling Servlet

Go to **classes** directory, where .java file is placed as enter the following commands at the command prompt to compile the servlet.

```
path=d:\jdk1.6.0\bin
set classpath=.;c:\apachetomcat6.0\lib\servlet-api.jar
javac TestServlet.java
```

- Make sure you change JDK and Tomcat directories according to your installation.
- Set path to the directory where JDK is installed. **Servlet-api.jar** contains API related to servlet like HttpServlet, HttpServletRequest etc., so it must be placed in the classpath. Compile the servlet (.java) to create .class file.

Starting Tomcat and Running Servlet

Start Tomcat by taking the follwing steps from BIN directory of Tomcat.

```
set java_home=d:\jdk1.6.0
startup
```

Once, tomcat is successfully started, go to browser and enterer the following URL.

```
http://localhost:8080/demo/test
```

You must see the output as follows.

Running Servlet

Select the servlet in the project window of IDE, click on the right button and select Run File from popup menu. This causes web application to be deployed first. If server is not yet started then NetBeans starts the server first and deploys the web application.

Then it invokes browser and start the servlet by giving the required URL –

http://localhost:8084/demo/simple

localhost	It is a special name that refers to the current system. In case you have to run servlet from a remote system, give the name of the server here.
8084	It is the port number at which server is running. Default is 8084 for tomcat that comes with NetBeans IDE.
demo	It is the name of the application.
simple	It is the URL pattern related to servlet.

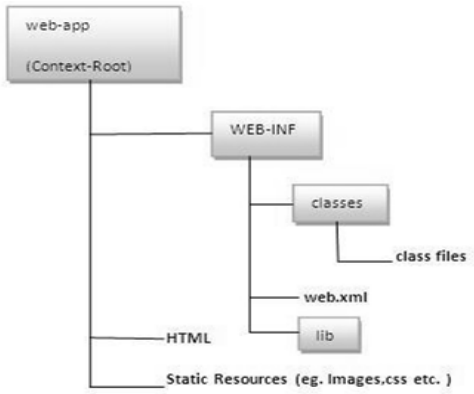
Deployment Descriptor (web.xml)

The deployment descriptor is an xml file, from which Web Container gets the information about the servlet to be invoked. The web container uses the Parser to get the information from the web.xml file.

Elements in web.xml

- <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.

Directory Structure



Example: Program to Read Data from Form and Display the Content

Sampleform.html

```
<html>
    <head>
        <title>Servlet Demo</title>
        <meta charset="UTF-8">
    </head>
    <body>
```

```
        <h1>Form Sends Data to Servlet</h1>
        <form name="log" action="display" method="post">
            <table>
                <tr><td>Enter Username: </td>
           <td><input type="text" name="uname" size="20"></td></tr>
                <tr><td>Enter Password: </td>
           <td><input type="password" name="pass" size="20"></td></tr>
                <tr><td colspan="2" align="center">
           <input type="submit" value="LOGIN"/></td></tr>
            </table>
        </form>
    </body>
</html>
```

DisplayContent.java

```
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
public class DisplayContent extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Date d = new Date();
        String un = request.getParameter("uname");
        String pwd = request.getParameter("pass");
        out.println("<h1>Username: " + un + "</h1>");
        out.println("<h1>Password: " + pwd + "</h1>");
        out.println("Logged in today at " + d.toString());
    }
    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Example: Program To Read Data From Form And Process The Content

Currency.html

```
<html>
    <head>
        <title>Servlet Demo</title>
        <meta charset="UTF-8">
    </head>
    <body>
```

```
        <h1>Currency Converter: INR to $</h1>
        <form action="currency" method="get">
            <table>
                <tr><td>Enter Amount in INR: </td><td>
                 <input type="text" name="amount" size="20"></td></tr>
                <tr><td colspan="2" align="center">
                 <input type="submit" value="CONVERT TO $"/></td></tr>
            </table>
        </form>
    </body>
</html>
```

CurrencyServlet.java

```
import java.io.*;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
public class CurrencyServlet extends HttpServlet {
    public static double RATE = 62.21;
    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try {
           String samt = request.getParameter("amount");
           if(!"".equals(samt)) {
            double amount = Double.parseDouble(samt);
            double usd = amount/RATE;
            out.println("<h2><i>US Dollar for INR " + samt + " = "
                                + usd + "</i></h2>");
           }
            else {
            out.println("<h2><font color=red><i>Amount is
                                required.</i></font></h2>");
            }
        }
        catch(NullPointerException e) {
          out.println("<h2><font color=red><i>"+e+"</i></font></h2>");
           out.println("<h2><font color=red><i>No amount value Found.
           <br/>Goto <a href=currency.html>This Page</a> and give the
           value</i></font></h2>");
        }
    }
}
```

Ways of Creating Servlets

There are three ways to create the servlet.

- By implementing the Servlet interface
- By inheriting the GenericServlet class

- 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.

Servlet Interface

This interface specifies the method to be implemented by the class that is to be run by Web Container. Each Servlet must directly or indirectly implement this interface. It provides common behavior to all the servlets.

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.

Example: Program to illustrate Servlet Interface

```
import java.io.*;
import javax.servlet.*;
public class ServletInterfaceDemo implements Servlet {
    ServletConfig config = null;
    @Override
    public void init(ServletConfig config) {
        this.config = config;
        System.out.println("servlet is initialized");
    }
    @Override
    public void service(ServletRequest req, ServletResponse res)
            throws IOException, ServletException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.print("<html><body><b>hello simple servlet</b>");
        out.print("</body></html>");
    }
    @Override
    public void destroy() {
        System.out.println("servlet is destroyed");
```

```
    }
    @Override
    public ServletConfig getServletConfig() {
        return config;
    }
    @Override
    public String getServletInfo() {
        return "copyright 2014-15";
    }
}
```

GenericServlet Class

GenericServlet class implements Servlet interface. It is an abstract class. It provides the implementation of all the methods of these interfaces except the service method. All subclasses of this class must implement service() method. GenericServlet class can handle any type of request so it is protocol-independent. It has additional methods to Servlet Interface:

- **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.
- **void init()** - it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
- **void log(String message)** - writes the given message in the servlet log file.

Example: Program to illustrate GenericServlet class

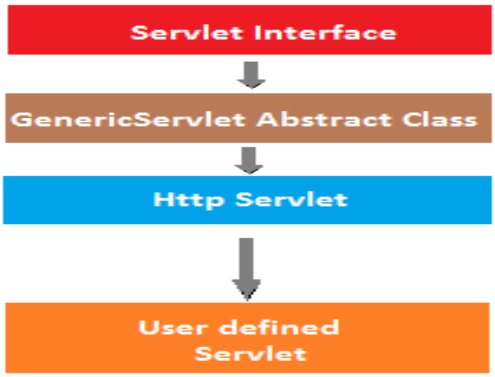
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.GenericServlet;
import javax.servlet.*;
public class GenericServletDemo extends GenericServlet {
    @Override
    public void service(ServletRequest req, ServletResponse res)
            throws IOException, ServletException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.print("<html><body>");
        out.print("<b>hello generic servlet</b>");
        out.print("</body></html>");
    }
}
```

HttpServlet Class

The HttpServlet class extends the GenericServlet class. It provides http specific methods such as doGet, doPost, doHead, doTrace etc. The class provides the following methods which we use generally:

- **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
- **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.

Servlet Hierarchy



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 information, attributes etc.

Method	**Description**
`Object getAttribute(String name)`	Returns value of the given attribute or null if attribute is not present.
`Enumeration getAttributeNames()`	Returns the names of all attributes contained in the request.
`void setAttribute(String name, Object value)`	Sets the named attribute to the given value.
`void removeAttribute(String name)`	Removes the named attribute from the list of attributes.
`String getParameter(String name)`	Returns the value of the named parameter. The name is case sensitive.
`String[] getParameterValues(String name)`	Returns an array of strings containing all of the values of given request parameter has or null if the parameter does not exist.
`Enumeration getParameterNames()`	Returns the list of request parameter names.
`String getServerName()`	Returns the host name of the server that received the request.
`int getServerPort()`	Returns the port number on which this request was received.
`String getContentType()`	Returns the Internet Media Type of the request entity data, or null if not known.

HttpServletRequest Interface

This interface extends ServletRequest and adds methods related to Http protocol.

Method	Description
Cookie[] getCookies()	Returns all cookies passed from client along with the request.
String getHeader(String name)	Returns the values of the given header field.
Enumeration getHeaderNames()	Returns names of all the header fields of the request.
String getMethod()	Returns the type of HTTP request was made.
HttpSession getSession()	Returns the session associated with the request. It creates a new session if one doesn't exist.
HttpSession getSession(boolean)	Returns the session associated with the request. It creates a new session if one doesn't exist when the parameter is true.
StringBuffer getRequestURL()	Reconstructs the URL the client used to make the request.

Example: Display Request Header values using ServletRequest Interface

```
import java.io.*;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.*;
public class DisplayHeader extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        // Set response content type
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "HTTP Header Request Example";
        String docType
                = "<!doctype html public \"-//w3c//dtd html 4.0 "
                + "transitional//en\">\n";
        out.println(docType
                + "<html>\n"
                + "<head><title>" + title + "</title></head>\n"
                + "<body bgcolor=\"#f0f0f0\">\n"
                + "<h1 align=\"center\">" + title + "</h1>\n"
                + "<table width=\"100%\" border=\"1\"
                      align=\"center\">\n"
                + "<tr bgcolor=\"#949494\">\n"
                + "<th>Header Name</th><th>Header Value(s)</th>\n"
                + "</tr>\n");
        Enumeration headerNames = request.getHeaderNames();
        while (headerNames.hasMoreElements()) {
```

```
            String paramName = (String) headerNames.nextElement();
            out.print("<tr><td>" + paramName + "</td>\n");
            String paramValue = request.getHeader(paramName);
            out.println("<td> " + paramValue + "</td></tr>\n");
        }
        out.println("</table>\n");
        out.println("<h1>Method: "+request.getMethod()+"</h1>");
        out.println("<h1>URL: "+request.getRequestURL()+"</h1>");
        out.println("<h1>Session: "+request.getSession()+"</h1>");
        out.println("<h1>Server Name:"+request.getServerName()
                                                    +"</h1>");
        out.println("<h1>Server Port:"+request.getServerPort()
                                                    +"</h1>");
        out.println("<h1>Client Host: "+request.getRemoteHost() +
                                                     "</h1>");
        out.println("<h1>Client Address: "+request.getRemoteAddr()
                                                    + "</h1>");
        out.println("</body></html>");
    }
    // Method to handle POST method request.
    @Override
    public void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Example: Program to Retrieve all parameter values using ServletRequest Interface

form.html

```
<html>
    <head>
        <title>Retrieving All Parameters</title>
    </head>
    <body>
        <h4>Order your Pizza Here</h4>
        <form action="RetrivingParams" method="post">
            <b>Name</b> <input type="text" name="name"><br><br>
            <b>Select the Crust:</b>
            <select name = "crust">
                <option value="pan">Pan</option>
                <option value="thin">Thin Crust</option>
                <option value="deep">Deep Crust</option>
                <option value="cheese">Cheese Burst</option>
            </select> <br><br>
            <b>Toppings: </b><br>
            <input type="checkbox" name="toppings" value="peas">
             Peas<br>
            <input type="checkbox" name="toppings" value="paneer">
             Paneer<br>
            <input type="checkbox" name="toppings" value="redpeppers">
             Red Peppers<br>
```

```
            <input type="checkbox" name="toppings" value="pineapple">
             Pineapple<br>
            <input type="checkbox" name="toppings" value="onion">
             Onion<br>
            <input type="checkbox" name="toppings" value="tomato">
             Tomato<br><br>

            <b>Select 1 FREE Appetizer</b>
            <input type="radio" name="appetizer" value="Garlic Bread">
                Garlic Bread
            <input type="radio" name="appetizer" value="Cheese Garlic
                Bread">Cheese Garlic Bread
            <input type="radio" name="appetizer" value="Veg Soup">
                Veg Soup
            <input type="radio" name="appetizer" value="Veg Sandwich">
                Veg Sandwich<br><br>

            <b>Address</b><br>
             <textarea name="address" rows=3 cols=40></textarea>
            <br><br>
            <b>Credit Card:</b><br>
            <input type="radio" name="cardType" value="Visa">Visa
            <input type="radio" name="cardType" value="MasterCard">
                MasterCard
            <input type="radio" name="cardType" value="Amex">
                American Express
            <br><br>
            <b>Credit Card Number:</b>
                <input type="password" name="cardNum">
            <b>Repeat Credit Card Number: </b>
                <input type="password" name="cardNum"><br><br>
            <input type="submit" name="submit" value="Order Pizza">
        </form>
    </body>
</html>
```

RetrivingParams.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class RetrivingParams extends HttpServlet {
    @Override
    public void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        Enumeration paramNames = request.getParameterNames();
        //Enumeration e = request.getHeaderNames();
        PrintWriter out = response.getWriter();
        response.setContentType("text/pdf");
```

```
response.setHeader("Content-
Disposition","attachment;filename=sample.pdf");

        out.print("<html><body>");
        out.print("<h1> Your Order...</h1>");
        out.println("<table border=\"1\" cellpadding =
\"5\"cellspacing = \"5\">");
        out.println("<tr> <th>Parameter Name</th>"
                + "<th>Parameter Value</th></tr>");
        while (paramNames.hasMoreElements()) {
            String paramName = (String) paramNames.nextElement();
            out.print("<tr><td>" + paramName + "\n<td>");
            String[] paramValues =
request.getParameterValues(paramName);
            if (paramValues.length == 1) {
                String paramValue = paramValues[0];
                if (paramValue.length() == 0) {
                    out.println("No Value");
                } else {
                    out.println(paramValue);
                }
            } else {
                out.println("<ul>");
                for (int i = 0; i < paramValues.length; i++) {
                    out.println("<li>" + paramValues[i] + "</li>");
                }
                out.println("</ul>");
            }
        }
        out.println("</table></body></html>");
    }
}
```

ServletResponse Interface

- Provides means for servlet to send response to client.

Methods	**Description**
PrintWriter getWriter()	returns a PrintWriter object that can send character text to the client.
void setBufferSize(int size)	Sets the preferred buffer size for the body of the response
void setContentLength (int len)	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header
void setContentType (String type)	sets the content type of the response being sent to the client before sending the respond.
void setBufferSize(int size)	sets the preferred buffer size for the body of the response.
boolean isCommitted()	returns a boolean indicating if the response has

	been committed
void setLocale(Locale loc)	sets the locale of the response, if the response has not been committed yet.

Response Headers

Header	Description
Allow	This header specifies the request methods (GET, POST, etc.) that the server supports.
Content-Disposition	This header lets you request that the browser ask the user to save the response to disk in a file of the given name.
Content-Encoding	This header specifies the way in which the page was encoded during transmission.
Content-Language	This header signifies the language in which the document is written. For example en, en-us, ru, etc.
Content-Length	This header indicates the number of bytes in the response.
Content-Type	This header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
Expires	This header specifies the time at which the content should be considered out-of-date and thus no longer be cached.
Refresh	This header specifies how soon the browser should ask for an updated page. You can specify time in number of seconds after which a page would be refreshed.
Retry-After	This header can be used in conjunction with a 503 (Service Unavailable) response to tell the client how soon it can repeat its request.

Content Types

```
response.setContentType("text/html"); // the most popular one
response.setContentType("text/plain");
response.setContentType("text/css"); // Cascading Style Sheet
response.setContentType("application/html");
response.setContentType("image/gif");
response.setContentType("application/zip");
response.setContentType("application/pdf");
response.setContentType("application/vnd.ms-excel");  //Excel Sheet
response.setContentType("application/vnd.ms-word");  //Word File
response.setContentType("text/plain");
//below line helps to download the file to the browser
response.setHeader("Content-Disposition","attachment;filename=simple.txt");
```

<meta tag>

The <meta> tag provides metadata about the HTML document. Metadata will not be displayed on the page, but will be machine parsable. <meta> tags always goes inside the <head> element. Metadata is always passed as name/value pairs.

Attribute	Value	Description
content	*text*	Gives the value associated with the http-equiv or name attribute <meta name="description" content="Free Web tutorials"> <meta name="keywords" content="HTML,CSS,JavaScript"> <meta name="author" content="SRIKANTH">
http-equiv	content-type default-style refresh	Provides an HTTP header for the information/value of the content attribute <meta http-equiv="content-type" content="text/html; charset=UTF-8"> <meta http-equiv="refresh" content="30"> <meta http-equiv="default-style" content="*the document's preferred stylesheet*">
name	application-name author description generator keywords	Specifies a name for the metadata

Example: Program to refresh the page for every 10 seconds using ServletResponse

Using HTML

```
<!DOCTYPE html>
<html>
	<head>
		<meta http-equiv="refresh" content="10">
	</head>
	<body>
		<p id="demo"></p>
		<script>
			d = new Date();
			document.getElementById("demo").innerHTML = d;
		</script>
	</body>
</html>
```

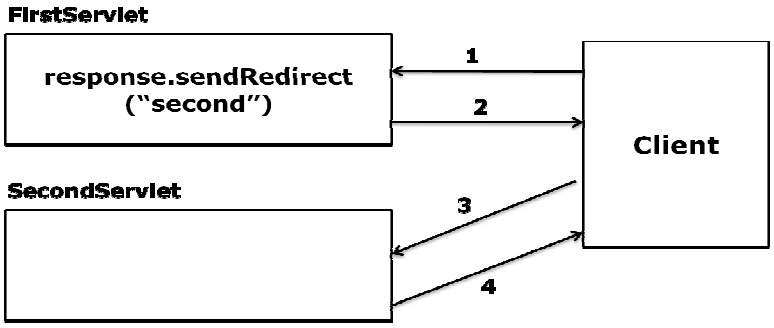
Using Servlet

```
import java.io.*;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetTime extends HttpServlet {
```

```
    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        Date cd = new Date();
        response.setHeader("refresh", "10");
        response.setContentType("text/html");
        PrintWriter pw = response.getWriter();
        String st = "<html><body><h1>"+cd.toString()+"</h1></body>"
        st = st  + "</html>";
        pw.println(st);
    }
}
```

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



1. Client requests for FirstServlet.
2. FirstServlet sends redirect message to client.
3. Client makes separate request for SecondServlet.
4. Response from SecondServlet is sent to client.

Example: Program using sendRedirect method to send request to Google server with the request data.

```
<!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, HttpServletRespon
se response)
            throws ServletException, IOException {
        String name=request.getParameter("name");
        response.sendRedirect("https://www.google.co.in/#q="+name);
    }
}
```

RequestDispatcher

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 another resource also. It is one of the ways of servlet collaboration.

The following are possible scenarios:

- A servlet processed the request but a JSP is to send output to client.
- A servlet processed the requet partially and remaining is to be done by another servlet.

We obtain an object or RequestDispatcher in either of the ways:

- Using **getRequestDispatcher**(String path) method or **getNamedDispatcher**(string) method of **ServletContext.** The path is relative to the root of the ServletContext.
- Using **getRequestDispatcher**(string path) method of **ServletRequest.** Path is relative to the current request.

The following methods of this interface are used to send control from one servlet/jsp to another

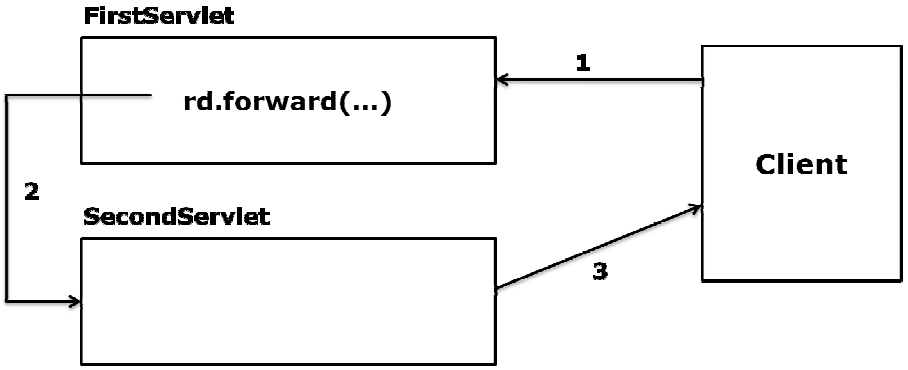
forward (ServletRequest, ServletResponse)	Sends control from current servlet/jsp to the sevlet/jsp to which RequestDispatcher points.
include((ServletRequest, ServletResponse)	Calls the resource and includes the content produced by the resource in the current response.

```
RequestDispatcher r = request.getRequestDispatcher(“/second”);
r.forward(request,response);
r.include(request,response);
```

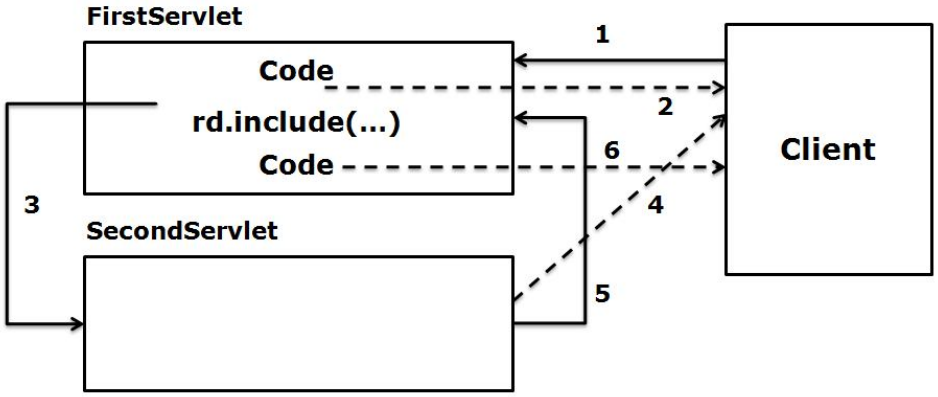
forward() vs. include()

- The forward() method of the RequestDispatcher interface may be called only by the calling servlet if no output has been committed to the client.

- In case of forward(), if no output exists in the response buffer that has not been committed, the buffer must be cleared before the target servlet's service method is called.
- In case of include(), the included servlet cannot set headers or call any method that affects the header of the response.



1. Client Request for FirstServlet
2. FirstServlet forwards request to SecondServlet
3. Response of SecondServlet is sent to client.



1. Client Request for FirstServlet
2. FirstServlet sends some response to client
3. FirstServlet calls SecondServlet
4. Secondservlet response goes to client
5. Control returns to FirstServlet
6. Response from FirstServlet goes to client.

First.java

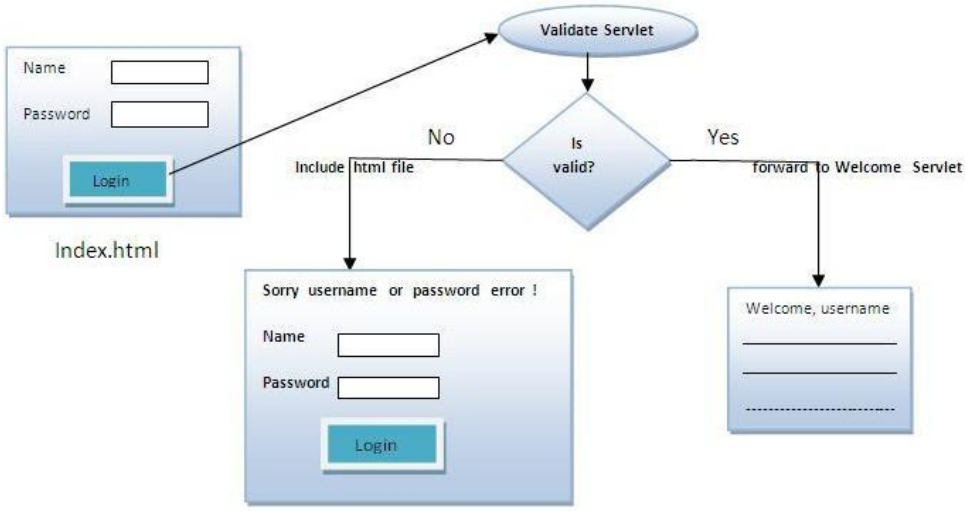
```
package servlets;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class First extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><head><title>First Servlet”);
            out.println(“</title></head>");
            out.println("<body><h1>First Servlet</h1>");
            out.println("<h1>Hello Welcome</h1>");
            request.setAttribute("name","DEPT IT");
            //RequestDispatcher rd =
            request.getRequestDispatcher("/color.html");
            RequestDispatcher rd =
            request.getRequestDispatcher("/second");
            //rd.include(request,response);
            rd.forward(request,response);
            out.println("</body>");
            out.println("</html>");
        }
    }
    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

Second.java

```
package servlets;
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 Second extends HttpServlet {
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<html><head><title>Second Servlet”);
```

```
            out.println("</title></head>");
            out.println("<body><h1>Second Servlet</h1>");
            String name = (String) request.getAttribute("name");
            out.println("<h1>"+name+"</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
    @Override
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
}
```



index.html

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

Login.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 {
```

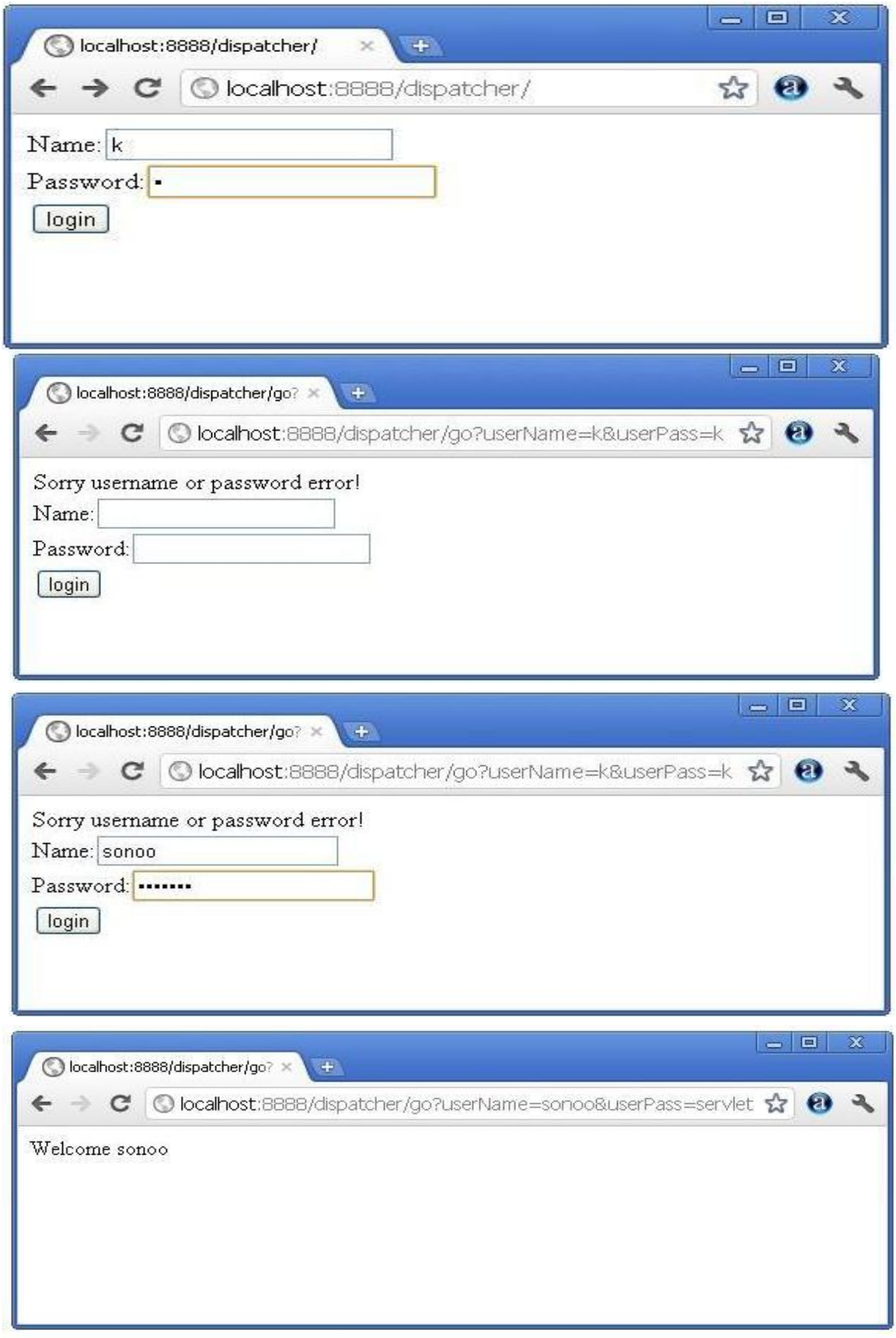
```
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    String p=request.getParameter("userPass");
    if(p.equals("servlet"){
        RequestDispatcher rd=request.getRequestDispatcher("servlet2");
        rd.forward(request, response);
    }
    else{
        out.print("Sorry UserName or Password Error!");
        RequestDispatcher rd=request.getRequestDispatcher("/index.html");
        rd.include(request, response);
        }
    }
}
```

WelcomeServlet.java

```
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

```
        <web-app>
         <servlet>
           <servlet-name>Login</servlet-name>
           <servlet-class>Login</servlet-class>
          </servlet>
          <servlet>
           <servlet-name>WelcomeServlet</servlet-name>
           <servlet-class>WelcomeServlet</servlet-class>
          </servlet>
          <servlet-mapping>
           <servlet-name>Login</servlet-name>
           <url-pattern>/servlet1</url-pattern>
          </servlet-mapping>
          <servlet-mapping>
           <servlet-name>WelcomeServlet</servlet-name>
           <url-pattern>/servlet2</url-pattern>
          </servlet-mapping>
          <welcome-file-list>
           <welcome-file>index.html</welcome-file>
          </welcome-file-list>
        </web-app>
```


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
localhost:8888/dispatcher/go?
localhost:8888/dispatcher/go?userName=sonoo&userPass=servlet
Welcome sonoo

Difference between forward() and sendRedirect() method

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2"). forward(request,response);	Example: response.sendRedirect("servlet2");

ServletConfig Interface

It contains methods that provide information about Servlet Configuration such as intialisation parameters. This interface is implemented by **GenericServlet.** So methods of this interface can be called directly from servlet or we can obtain an object of ServletConfig using **getServletConfig()** method of servlet interface.

Method	Meaning
String getInitParameter(string name)	Returns the value of the initialization parameter whose name is supplied.
Enumeration getInitParameterNames()	Returns the names of all Initialization parameters.
ServletContext getServletContext()	Returns ServletContext object associated with the web application. Each servlet belongs to an application and each application is associated with a Servlet Context.
String getServletName()	Returns the name assigned to a servlet in its deployment descriptor. If no name is specifird then the servlet name is returned instead.

Example: Program 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

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
public class DemoServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   ServletConfig config=getServletConfig();
   Enumeration<String> e=config.getInitParameterNames();
   String str="";
   while(e.hasMoreElements()){
   str=e.nextElement();
   out.print("<br>Name: "+str);
   out.print(" value: "+config.getInitParameter(str));
   }
   out.close();
}
}
```

web.xml

```
<web-app>
<servlet>
<servlet-name>DemoServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>
<init-param>
<param-name>username</param-name>
<param-value>system</param-value>
</init-param>
<init-param>
<param-name>password</param-name>
<param-value>oracle</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>DemoServlet</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
</web-app>
```

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 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.

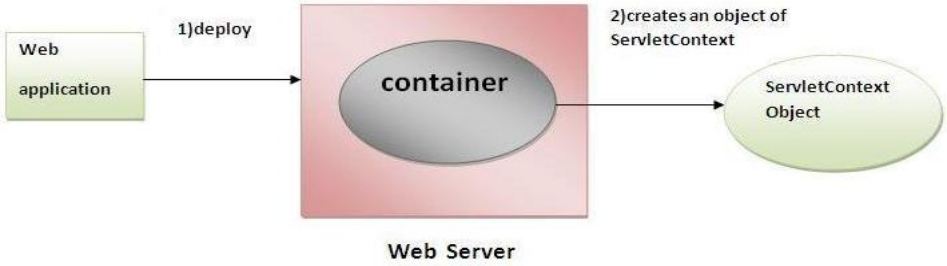
Advantage of ServletContext

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.

- **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
- **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
- **public void setAttribute(String name,Object object):**sets the given object in the application scope.
- **public Object getAttribute(String name):**Returns the attribute for the specified name.
- **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
- **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

public ServletContext getServletContext()

Example of getServletContext() method

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

Example of ServletContext to get the initialization parameter

DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
      res.setContentType("text/html");
      PrintWriter pw=res.getWriter();
      //creating ServletContext object
      ServletContext context=getServletContext();
//Getting the value of the initialization parameter and printing it
      String driverName=context.getInitParameter("dname");
      pw.println("driver name is="+driverName);
      pw.close();
}}
```

web.xml

```
      <web-app>
      <servlet>
      <servlet-name>sonoojaiswal</servlet-name>
      <servlet-class>DemoServlet</servlet-class>
      </servlet>
      <context-param>
      <param-name>dname</param-name>
      <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
      </context-param>
      <servlet-mapping>
      <servlet-name>sonoojaiswal</servlet-name>
      <url-pattern>/context</url-pattern>
      </servlet-mapping>
      </web-app>
```

Session Tracking

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 a user to recognize to particular user.

Why use Session Tracking? It is used to recognize the particular user.

There are four techniques used in Session tracking:

- ***Cookies***
- ***Hidden Form Field***
- ***URL Rewriting***
- ***HttpSession***

Cookies

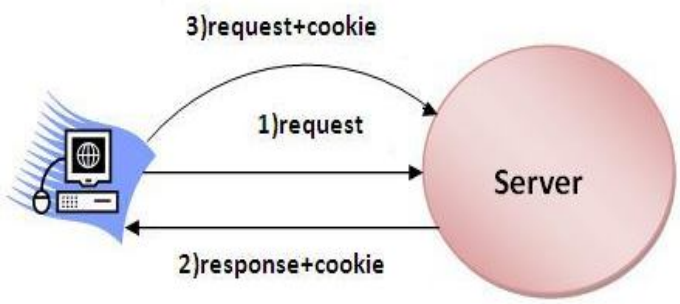
A **cookie** is a small piece of information that is persisted between the multiple client requests i.e., ***information saved by the browser and later sent back to the server***. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

```
Cookie c = new Cookie(String name, String value);
```

Keep in mind that, neither the name nor the value should contain white space or any of the following characters: [ ] ( ) = , " / ? @ : ;

How does Cookie work?

- 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 Cookies

- There are 2 types of cookies in servlets.
  - Non-persistent cookie
  - 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 sessions**. It is not removed each time when user closes the browser. It is removed only if user logout or sign-out.

Methods of 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.

Setting the maximum age: You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

```
cookie.setMaxAge(60*60*24);
```

How to create Cookie?

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

How to delete Cookie? It is mainly used for signout of user.

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

How to get Cookies?

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

Advantage of Cookies

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

Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.

- Only textual information can be set in Cookie object.

addcookie.html

```
<html>
   <head>
      <title>Cookie Demo</title>
   </head>
   <body bgcolor="lightblue">
   <center>
      <form action="addcookie" method="post">
         <table>
            <tr><th>Name:</th><td><input type="text" name="cname"/></td></tr>
            <tr><th>Value:</th><td><input type="text" name="cvalue"/></td></tr>
            <tr><th colspan="2" align="left">
       <input type="checkbox" name="durable" value="y"> Durable Cookie</th></tr>
            <tr><th align="center"><input type="submit" value="LOGIN"/> </th></tr>
         </table>
      </form><br/>
      <h1><a href="listcookies">LIST OF COOKIES</a></h1>
   </center>
   </body>
</html>
```

AddCookie.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class AddCookie extends HttpServlet {
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter pw = response.getWriter();
      String name =request.getParameter("cname");
      String value =request.getParameter("cvalue");
      String durable =request.getParameter("durable");
      Cookie c = new Cookie(name,value);
      if(durable != null) {
         c.setMaxAge(7*24*60*60); // persistent for 7 days
      }
      pw.println(c.getMaxAge());
      response.addCookie(c);
      pw.println("<h2>Cookie Created...</h2>");
      RequestDispatcher rd = request.getRequestDispatcher("/addcookie.html");
      rd.include(request, response);
   }
   @Override
   public String getServletInfo() {
      return "Short description";
   }// </editor-fold>
}
```

ListCookies.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
public class ListCookies extends HttpServlet {
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter pw = response.getWriter();
      Cookie cookies[] = request.getCookies();
      String st = "<body><table border=1><tr><th>Name</th><th>Value</th>";
      St = st + "<th>MaxAge</th></tr>";
      for(Cookie c: cookies) {
         st = st + "<tr><td>" + c.getName() + "<td>" + c.getValue() + "<td>" +
c.getMaxAge() + "</tr>";
      }
      st = st + "</table></body>";
      pw.println(st);
      pw.println("<h1><a href=addcookie.html>ADD COOKIE</a></h1>");
   }
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      doGet(request, response);
   }
   @Override
   public String getServletInfo() {
      return "Short description";
   }// </editor-fold>
}
```

Hidden form fields

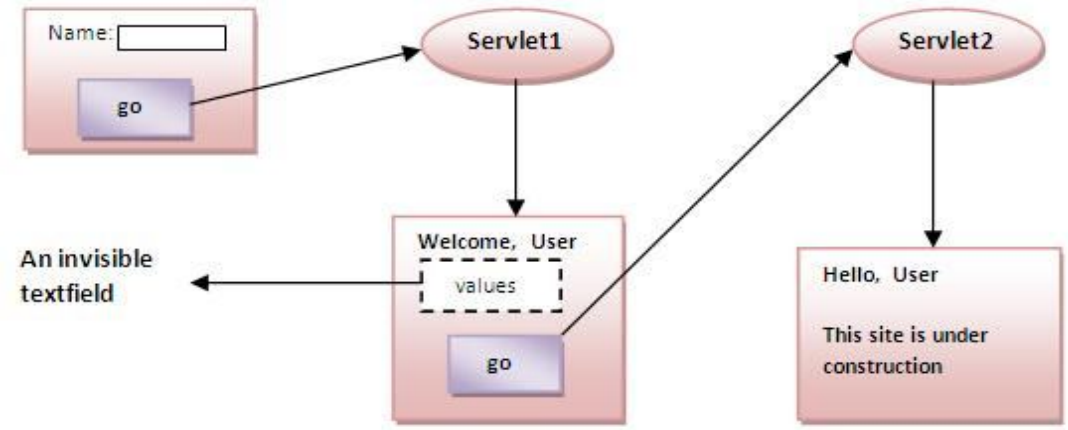
- In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of a user.
- In such case, we store the information in the hidden field and get it from another servlet.
- This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Advantage of Hidden Form Field

- It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.



index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String n=request.getParameter("userName");
      out.print("Welcome "+n);
      //creating form that have invisible textfield
      out.print("<form action='servlet2'>");
      out.print("<input type='hidden' name='uname' value='"+n+"'>");
      out.print("<input type='submit' value='go'>");
      out.print("</form>");
      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 doGet(HttpServletRequest request, HttpServletResponse response)
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      //Getting the value from the hidden field
      String n=request.getParameter("uname");
      out.print("Hello "+n);
```

```
        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>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
</web-app>
```

URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

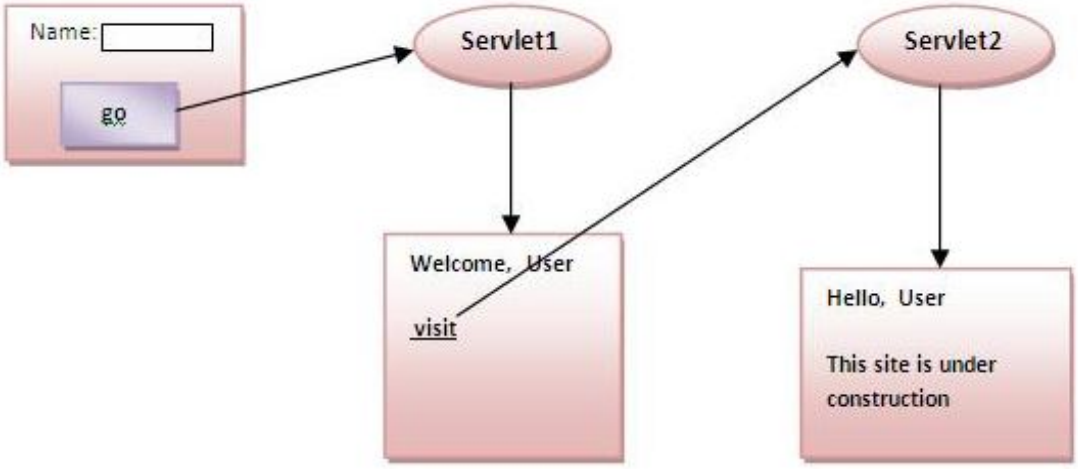
```
url?name1=value1&name2=value2&??
```

Advantage of URL Rewriting

- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

- It will work only with links.
- It can send only textual information.



We are maintaining the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String n=request.getParameter("userName");
      out.print("Welcome "+n);
      //appending the username in the query string
      out.print("<a href='servlet2?uname="+n+"'>visit</a>");
      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 doGet(HttpServletRequest request, HttpServletResponse response)
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      //getting value from the query string
      String n=request.getParameter("uname");
      out.print("Hello "+n);
      out.close();
      } catch(Exception e) { System.out.println(e); }
   }
}
```

web.xml

```
      <web-app>
      <servlet>
      <servlet-name>s1</servlet-name>
      <servlet-class>FirstServlet</servlet-class>
```

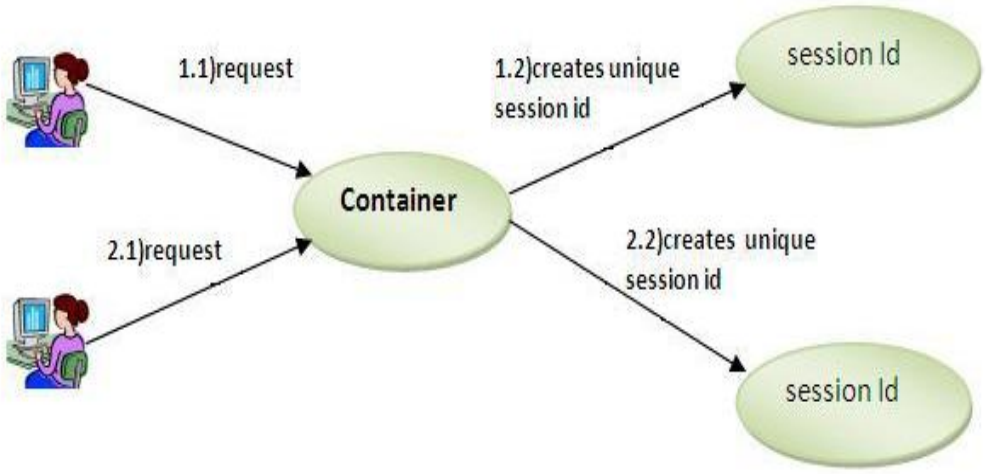
```
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
</web-app>
```

HttpSession Interface

The HttpSession object represents a user session. A user session contains information about the user across multiple HTTP requests. When a user enters your site for the first time, the user is given a unique ID to identify his session by. This ID is typically stored in a cookie or in a request parameter.

Container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

- bind objects
- view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



S.N.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.

3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public boolean isNew(** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

session.html

```
<html>
   <head>
      <title>Session Demo</title>
   </head>
   <body bgcolor="#eeeeee">
      <h1>Session Variables Demo</h1>
      <form action="sessionservlet" method="post">
         <table>
            <tr><th>Name:</th><td><input type="text" name="key"/></td></tr>
            <tr><th>Value:</th><td><input type="text" name="value"/></td></tr>
            <tr><th align="center" colspan="2">
                  <input type="submit" value="Add" name="act"/>
                  <input type="submit" value="Remove" name="act"/>
               </th>
            </tr>
         </table>
      </form>
   </body>
</html>
```

SessionServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class SessionServlet extends HttpServlet {
```

```
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String act = request.getParameter("act");
        String key, value;
        key = request.getParameter("key");
        HttpSession s = request.getSession();
        if(act.equals("Add")) {
            value = request.getParameter("value");
            s.setAttribute(key,value);
            out.println("Key was added to Session");
        }
        else {
            s.removeAttribute(key);
            out.println(key + " was removed");
        }
        //Display Session varibles
        out.println("<h3>Available Session variables</h3><ul>");
        Enumeration e = s.getAttributeNames();
        while(e.hasMoreElements()) {
            key = (String)e.nextElement();
            value = (String) s.getAttribute(key);
            out.println("<li>"+ key + ":" + value  + "</li>");
        }
        out.println("</ul>");
    }
}
```

Deleting Session Data:

When you are done with a user's session data, you have several options:

- **Remove a particular attribute:** You can call *public void **removeAttribute(String name)*** method to delete the value associated with a particular key.
- **Delete the whole session:** You can call *public void **invalidate()*** method to discard an entire session.
- **Setting Session timeout:** You can call *public void **setMaxInactiveInterval(int interval)*** method to set the timeout for a session individually.
- **Log the user out:** The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.

LOGIN DEMO USING SESSIONS

appllogin.html

```
<html>
    <head>
        <title>Login Session Demo</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
   </head>
   <body>
   <center>
      <form action="applogin" method="post">
         <table>
            <tr><th>Username:</th><td>
               <input type="text" name="userName"/></td></tr>
            <tr><th>Password:</th><td>
               <input type="password" name="userPass"/></td></tr>
          <tr><th  align="center"  colspan="2"><input  type="submit"  value="LOGIN"/>
          </th></tr>
         </table>
      </form>
   </center>
</body>
</html>
```

MemLogin.java (/applogin)

```
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;
import javax.servlet.http.HttpSession;
public class MemLogin extends HttpServlet {
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      try {
         String uname = request.getParameter("userName");
         String upass = request.getParameter("userPass");
         if (upass.equals("admin123")) {
            HttpSession session = request.getSession();
            session.setAttribute("name", uname);
            request.getRequestDispatcher("profile").forward(request, response);
         } else {
            out.print("Sorry, username or password error!");
            request.getRequestDispatcher("appllogin.html").include(request, response);
         }
      } catch (NullPointerException e) {

      }
   }
}
```

UserProfile.java (/profile)

```
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;
import javax.servlet.http.HttpSession;
public class UserProfile extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         HttpSession s = request.getSession(false);
         if (s != null) {
            String name = (String) s.getAttribute("name");
            out.print("<h1>Hello, " + name + " Welcome to Profile</h1>");
            out.println("<h1>"+new java.util.Date()+"</h1>");
            out.println("<a href='logout'>LOGOUT</a>");
         } else {
            out.print("Please login first");
            request.getRequestDispatcher("login.html").include(request, response);
         }
      } }
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException { processRequest(request, response); }
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException { processRequest(request, response); }
}
```

Logout.java

```
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;
import javax.servlet.http.HttpSession;
public class Logout extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         HttpSession session=request.getSession();
         session.removeAttribute("name");
         session.invalidate();
         out.print("<h4>You are successfully logged out!</h4>");
         request.getRequestDispatcher("appllogin.html").include(request, response);
      }
   }
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException { processRequest(request, response); }
}
```