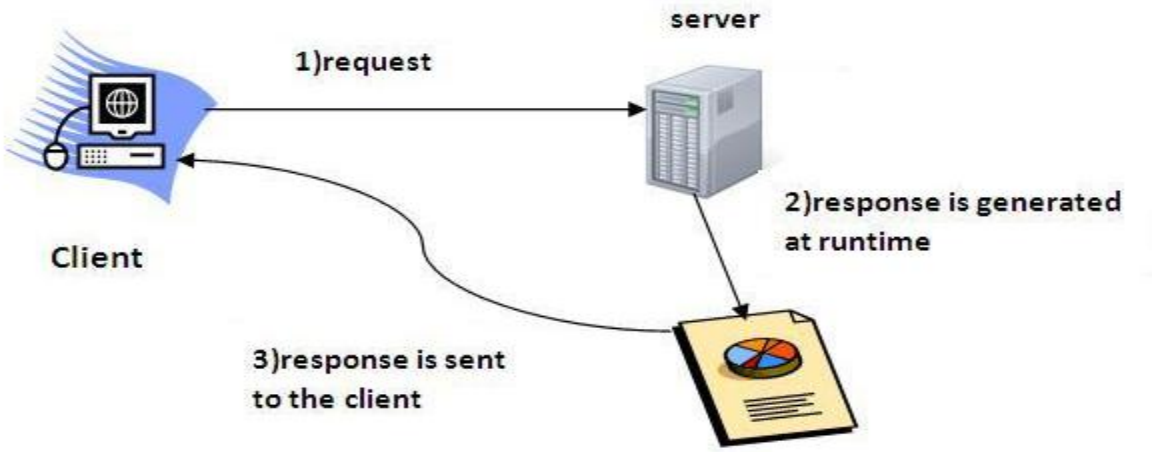
Servlet

What is a Servlet?

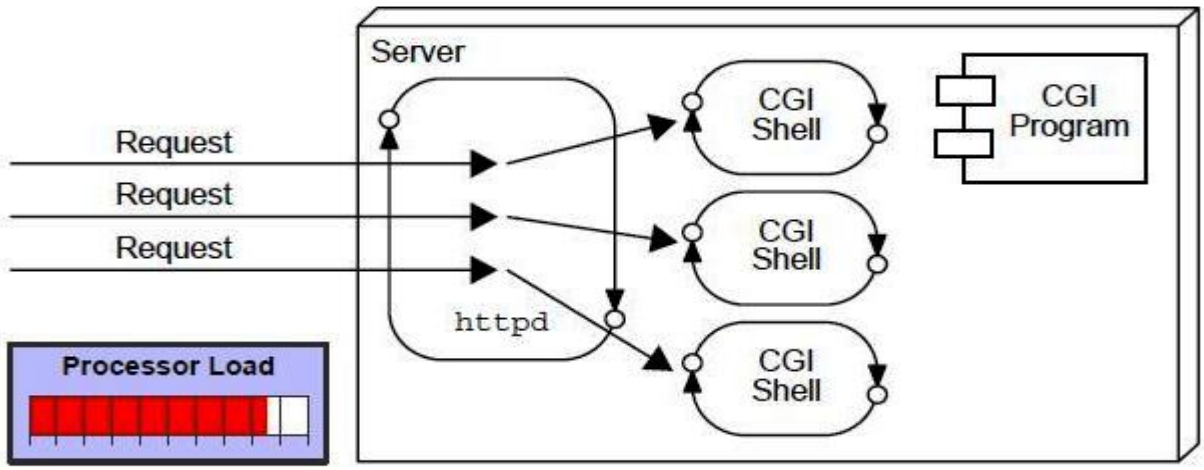
Servlet can be described in many ways, depending on the context.

- 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 extends the capabilities of the servers and responds 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.



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.



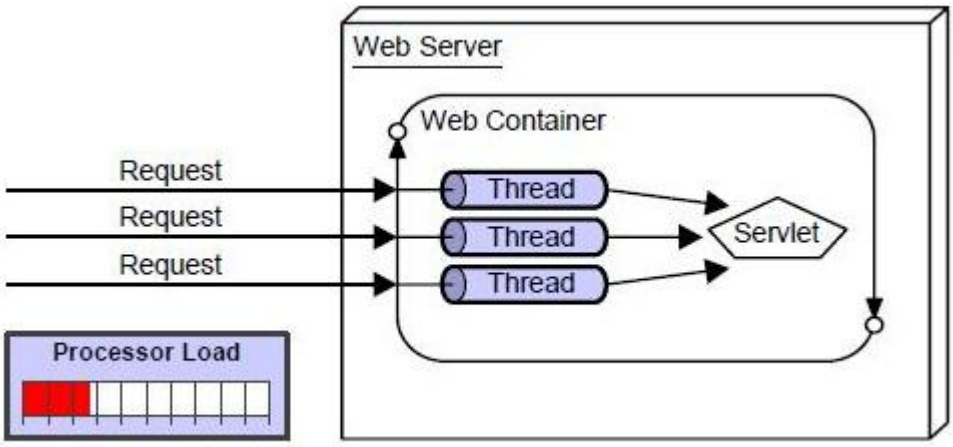
Created By: Vipin Kumar, Assistant Professor, MCA, KIET

Disadvantages of CGI

There are many problems in CGI technology:

1. If number of client's 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 memory leak, garbage collection etc.
4. Secure: because it uses java language..

Servlet Terminology

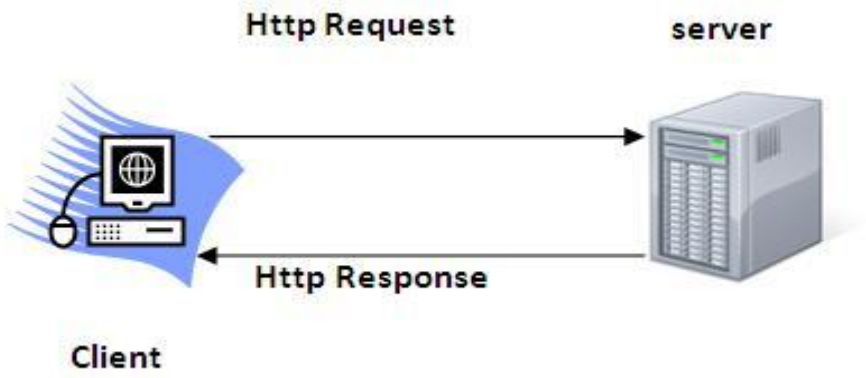
There are some key points that must be known by the servlet programmer like server, container, get request, post request etc. Let's first discuss these points before starting the servlet technology.

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 by default 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 by default.



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	In case of post request, large amount

data can be sent because data is sent in header.	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 information 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

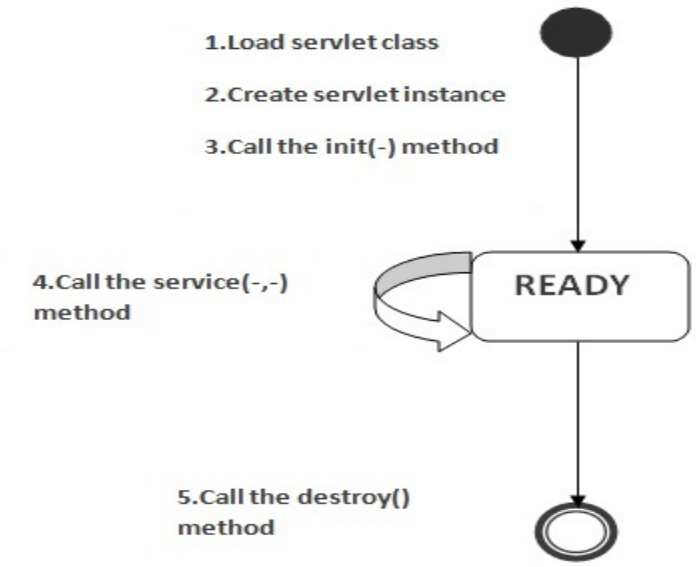
There are many classes in javax.servlet.http package. They are as follows:

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

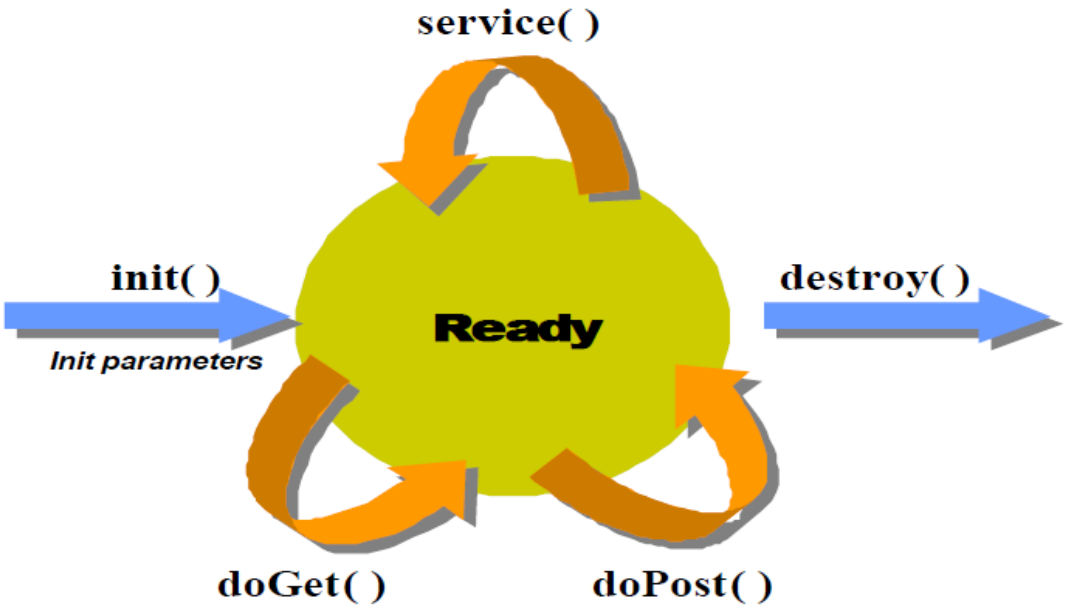
Servlet Life Cycle

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.



Servlet Life Cycle Methods



Steps to Create Servlet

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.

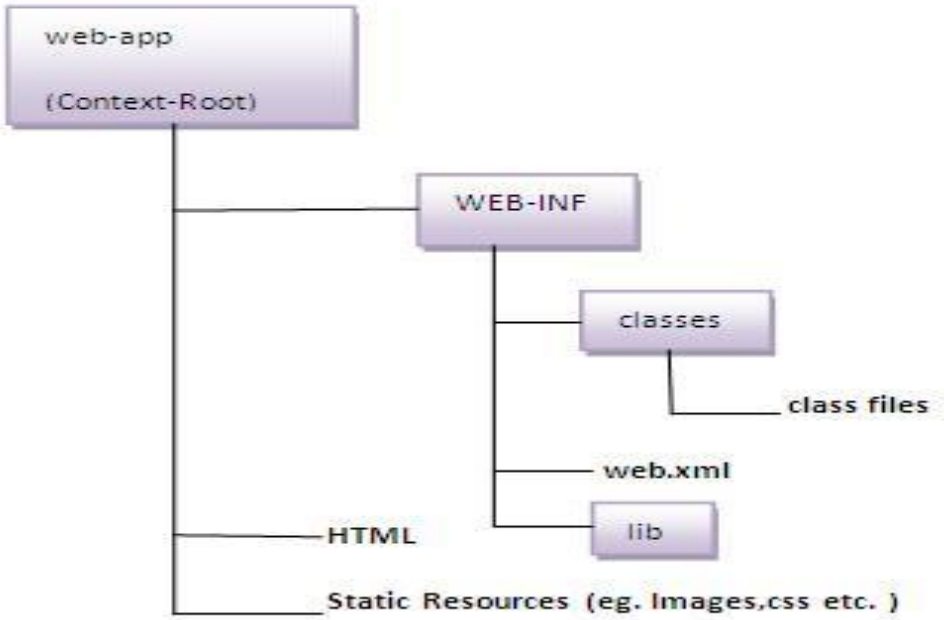
Here, we are going to use apache tomcat server in this example. The steps are as follows:

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

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 responds 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.     //writing html in the stream
11.     pw.println("<html><body>");
12.     pw.println("Welcome to servlet");
13.     pw.println("</body></html>");

14.     pw.close();//closing the stream
15.     }}
```

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

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>myservlet</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7.
8. <servlet-mapping>
9. <servlet-name>myservlet</servlet-name>
10.     <url-pattern>/welcome</url-pattern>
11.     </servlet-mapping>
12.
13.     </web-app>
```

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.

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

There are two methods defined in the RequestDispatcher interface.

Methods of RequestDispatcher interface

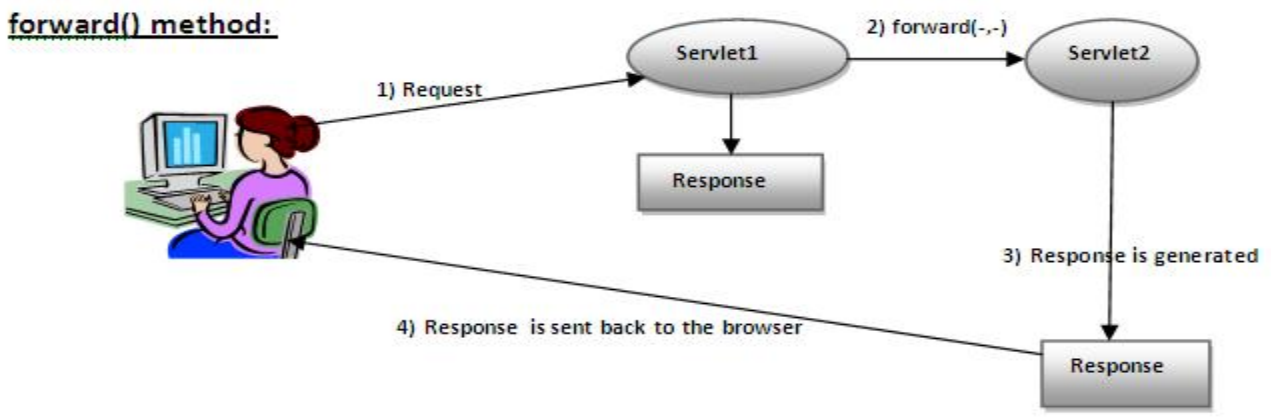
The RequestDispatcher interface provides two methods. They are:

public void forward(ServletRequest request,ServletResponse response)throws ServletException, java.io.IOException

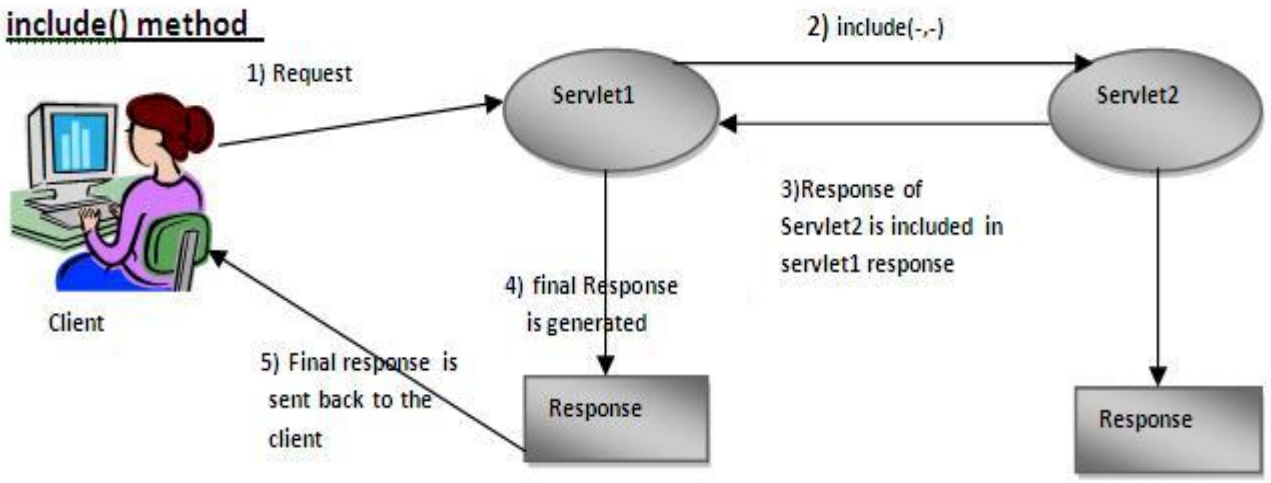
Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

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

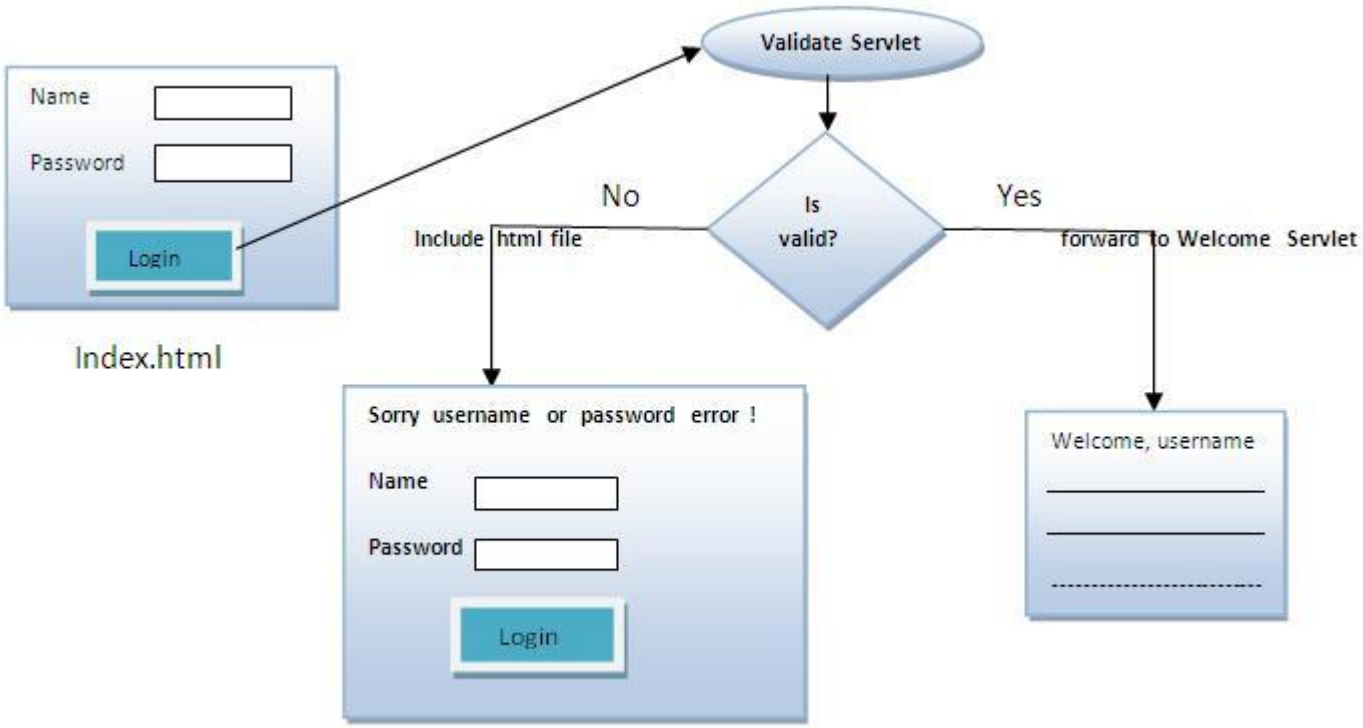
1. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
2. //servlet2 is the url-pattern of the second servlet
3. rd.forward(request, response);//method may be include or forward

Example of RequestDispatcher interface

. 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 servlet, 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

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class Login extends HttpServlet {
6.
```

```
7. public void doPost(HttpServletRequest request, HttpServletResponse respon
   se)
8.       throws ServletException, IOException {
9.
10.         response.setContentType("text/html");
11.         PrintWriter out = response.getWriter();
12.
13.         String n=request.getParameter("userName");
14.         String p=request.getParameter("userPass");
15.
16.         if(p.equals("servlet"){
17.            RequestDispatcher rd=request.getRequestDispatcher("servlet2");
18.            rd.forward(request, response);
19.         }
20.         else{
21.            out.print("Sorry UserName or Password Error!");
22.            RequestDispatcher rd=request.getRequestDispatcher("/index.html
   ");
23.            rd.include(request, response);
24.
25.            }
26.         }
27.
28.      }
```

WelcomeServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class WelcomeServlet extends HttpServlet {
6.
7.    public void doPost(HttpServletRequest request, HttpServletResponse resp
   onse)
8.       throws ServletException, IOException {
9.
10.         response.setContentType("text/html");
11.         PrintWriter out = response.getWriter();
12.
13.         String n=request.getParameter("userName");
14.         out.print("Welcome "+n);
15.         }
```

```
16.
17.     }
```

---

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>
```

SendRedirect Method

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 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");

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, which 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

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset="ISO-8859-1">
5. <title>sendRedirect example</title>
6. </head>
7. <body>
8.
9.
10.     <form action="MySearcher">
11.     <input type="text" name="name">
12.     <input type="submit" value="Google Search">
13.     </form>
14.
15.     </body>
16.     </html>
```

MySearcher.java

```
1. import java.io.IOException;
2. import javax.servlet.ServletException;
3. import javax.servlet.http.HttpServlet;
4. import javax.servlet.http.HttpServletRequest;
```

```
5. import javax.servlet.http.HttpServletResponse;
6.
7. public class MySearcher extends HttpServlet {
8.     protected void doGet(HttpServletRequest request, HttpServletResponse r
   esponse)
9.           throws ServletException, IOException {
10.
11.              String name=request.getParameter("name");
12.              response.sendRedirect("https://www.google.co.in/#q="+name);
13.          }
14.      }
```

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 respons
   e)
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>
```

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.

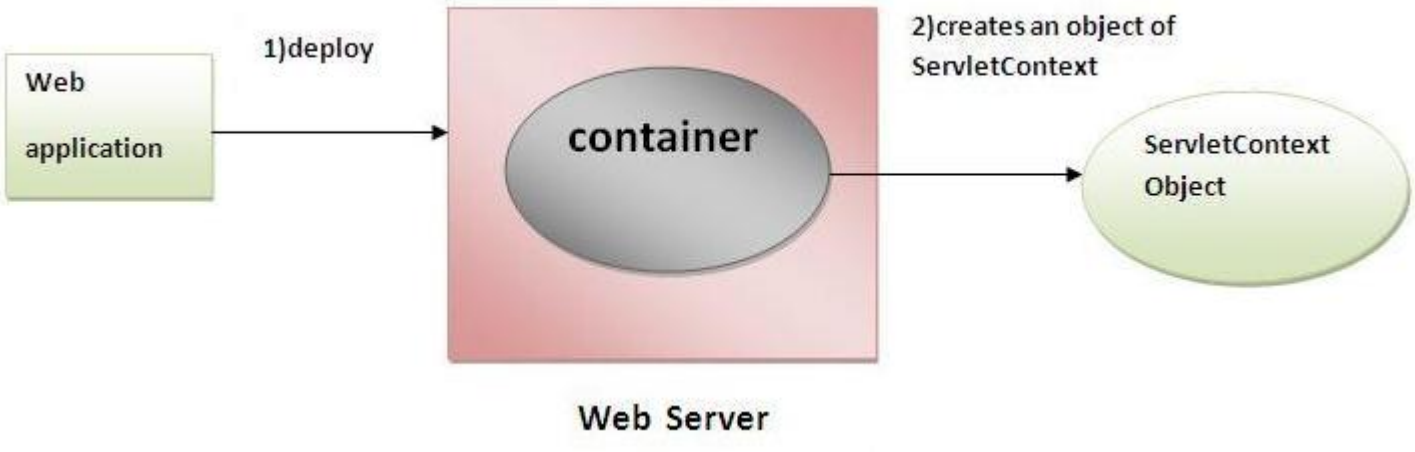
Advantage of ServletContext

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.

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.

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

  <context-param>
    <param-name>parametername</param-name>
    <param-value>parametervalue</param-value>
  </context-param>
 ......
</web-app>
```

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();
```

```
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>
```

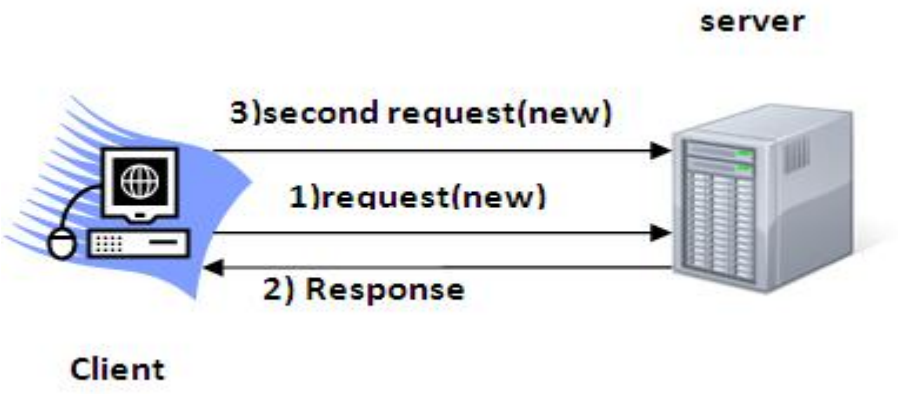
Session Tracking in Servlets

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of a 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

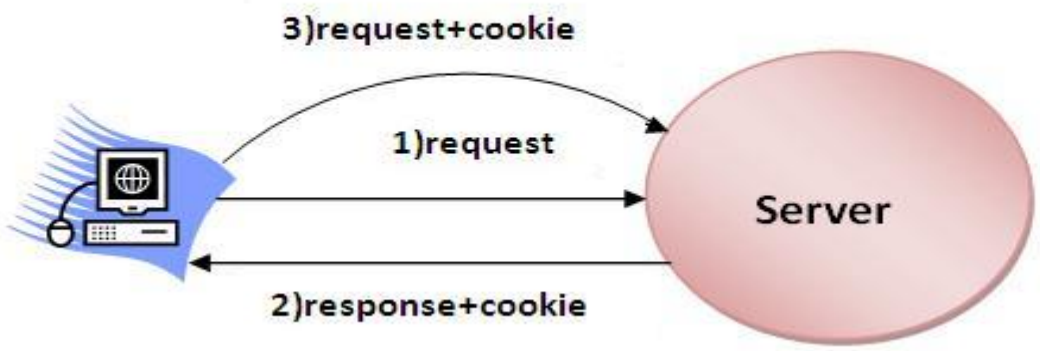
Cookies in Servlet

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 sessions: 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.

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
```

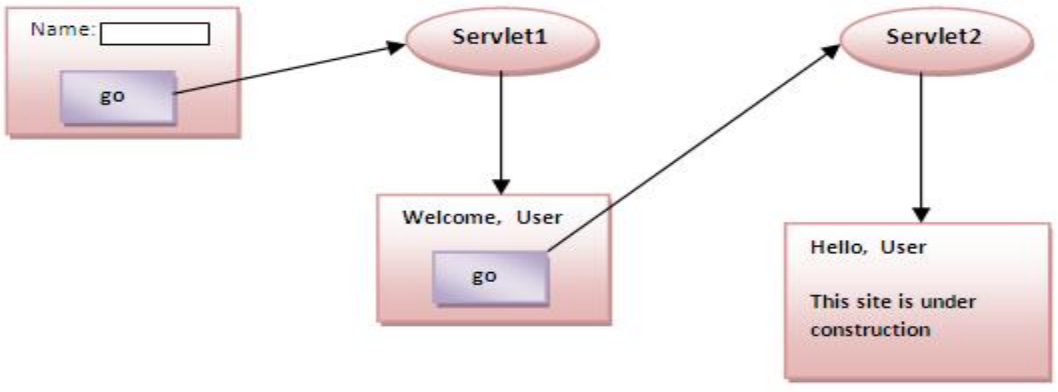
How to get Cookies?

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 cookie
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 respo
   nse){
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.
25.          out.close();
26.
27.             }catch(Exception e){System.out.println(e);}
28.        }
29.      }
```

SecondServlet.java

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

```
4.
5. public class SecondServlet extends HttpServlet {
6.
7. public void doPost(HttpServletRequest request, HttpServletResponse respon
   se){
8.    try{
9.
10.         response.setContentType("text/html");
11.         PrintWriter out = response.getWriter();
12.
13.         Cookie ck[]=request.getCookies();
14.         out.print("Hello "+ck[0].getValue());
15.
16.         out.close();
17.
18.             }catch(Exception e){System.out.println(e);}
19.         }
20.
21.
22.      }
```

web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-class>FirstServlet</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.      <servlet>
14.      <servlet-name>s2</servlet-name>
15.      <servlet-class>SecondServlet</servlet-class>
16.      </servlet>
17.
18.      <servlet-mapping>
19.      <servlet-name>s2</servlet-name>
20.      <url-pattern>/servlet2</url-pattern>
21.      </servlet-mapping>
```

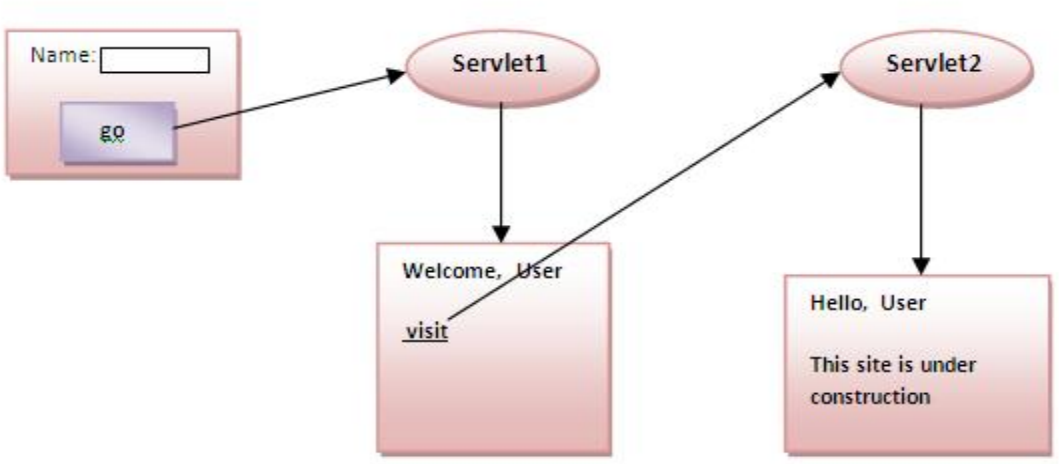
```
22.
23.      </web-app>
```

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&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



Advantage of URL Rewriting

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

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

Example of using URL Rewriting

In this example, 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

```
1. <form action="servlet1">
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. public class FirstServlet extends HttpServlet {
6.
7. public void doGet(HttpServletRequest request, HttpServletResponse respons
   e){
8.       try{
9.
10.            response.setContentType("text/html");
11.            PrintWriter out = response.getWriter();
12.
13.            String n=request.getParameter("userName");
14.            out.print("Welcome "+n);
15.
16.            //appending the username in the query string
17.            out.print("<a href='servlet2?uname="+n+"'>visit</a>");
18.
19.            out.close();
20.
21.                  }catch(Exception e){System.out.println(e);}
22.         }
23.
24.      }
```

SecondServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class SecondServlet extends HttpServlet {
```

```
6.
7. public void doGet(HttpServletRequest request, HttpServletResponse respons
   e)
8.       try{
9.
10.           response.setContentType("text/html");
11.           PrintWriter out = response.getWriter();
12.
13.           //getting value from the query string
14.           String n=request.getParameter("uname");
15.           out.print("Hello "+n);
16.
17.           out.close();
18.
19.                }catch(Exception e){System.out.println(e);}
20.        }
21.
22.
23.     }
```

web.xml

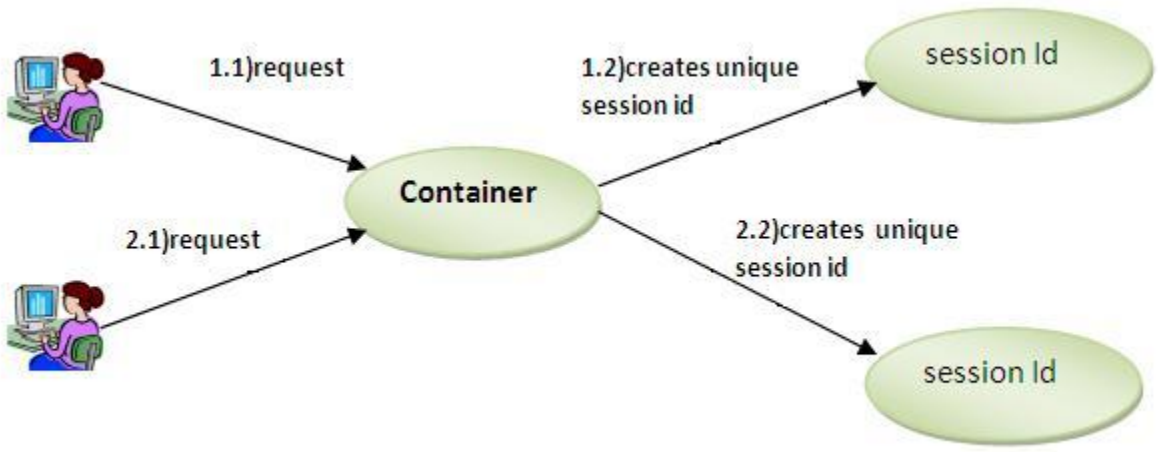
```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-class>FirstServlet</servlet-class>
6. </servlet>
7.
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10.     <url-pattern>/servlet1</url-pattern>
11.     </servlet-mapping>
12.
13.     <servlet>
14.     <servlet-name>s2</servlet-name>
15.     <servlet-class>SecondServlet</servlet-class>
16.     </servlet>
17.
18.     <servlet-mapping>
19.     <servlet-name>s2</servlet-name>
20.     <url-pattern>/servlet2</url-pattern>
21.     </servlet-mapping>
22.
```

```
23.     </web-app>
```

HttpSession Interface

In such case, 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:

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



How to get the HttpSession object ?

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

1. public HttpSession getSession():Returns the current session associated with this request, or if the request does not have a session, creates one.
2. public HttpSession getSession(boolean create):Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. public String getId():Returns a string containing the unique identifier value.
2. public long getCreationTime():Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. public long getLastAccessedTime():Returns the last time the client sent a request associated with this session, as the number of milliseconds since

midnight January 1, 1970 GMT.

4. public void invalidate():Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

index.html

```
1. <form action="servlet1">
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 doGet(HttpServletRequest request, HttpServletResponse respons
   e){
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.           HttpSession session=request.getSession();
18.           session.setAttribute("uname",n);
19.
20.           out.print("<a href='servlet2'>visit</a>");
```

```
21.
22.             out.close();
23.
24.                 }catch(Exception e){System.out.println(e);}
25.         }
26.
27.     }
```

SecondServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class SecondServlet extends HttpServlet {
6.
7. public void doGet(HttpServletRequest request, HttpServletResponse respons
   e)
8.       try{
9.
10.             response.setContentType("text/html");
11.             PrintWriter out = response.getWriter();
12.
13.             HttpSession session=request.getSession(false);
14.             String n=(String)session.getAttribute("uname");
15.             out.print("Hello "+n);
16.
17.             out.close();
18.
19.                 }catch(Exception e){System.out.println(e);}
20.         }
21.
22.
23.     }
```

web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-class>FirstServlet</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.     <servlet>
14.     <servlet-name>s2</servlet-name>
15.     <servlet-class>SecondServlet</servlet-class>
16.     </servlet>
17.
18.     <servlet-mapping>
19.     <servlet-name>s2</servlet-name>
20.     <url-pattern>/servlet2</url-pattern>
21.     </servlet-mapping>
22.
23.     </web-app>
```