Unit 6
Servlets

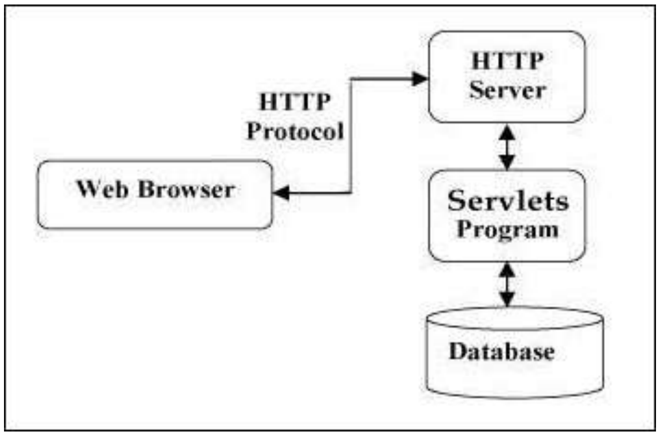
Servlets are the **Java programs** that run on the **Java-enabled web server** or **application server**. They are used to handle the **request obtained** from the **webserver**, **process the request**, **produce the response**, then **send a response back to the webserver**.
Using Servlets, you can **collect input** from **users** through **web page forms**, **present records** from a **database** or **another source**, and **create web pages dynamically**.
Properties of Servlets are as follows:

- **Servlets work on the server-side.**
- **Servlets are capable of handling complex requests obtained from the webserver.**

Advantages of Servlet

- **Performance** is significantly better.
- **Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.**
- Servlets are **platform-independent** because they are written in **Java.**
- **Java security manager** on the **server enforces** a **set of restrictions** to **protect the resources on a server machine.** So **servlets are trusted.**
- The full functionality of the **Java class libraries** is available to a **servlet.** It can communicate with **applets, databases, or other software** via the sockets.

How Servlets work



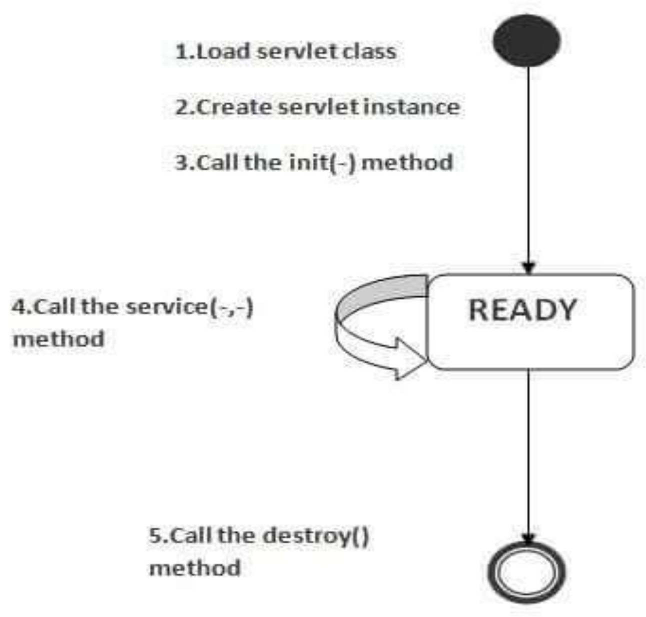
Execution of Servlets basically involves six basic steps:

1. The **clients** send the request to the webserver.
2. The **web server** receives the **request**.
3. The web server passes the request to the corresponding servlet.
4. The **servlet processes the request** and **generates the response in the form of output**.
5. **The servlet sends the response back to the webserver.**
6. The **web server sends the response back to the client** and the **client browser displays it on the screen.**

Life cycle of a servlet

The entire life cycle of a Servlet is managed by the **Servlet container** which uses the **javax.servlet.Servlet** interface to understand the **Servlet object** and **manage** it. So, before creating a Servlet object, let's first understand the life cycle of the Servlet object which is actually understanding how the Servlet container manages the Servlet object.

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.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

1) Servlet class is loaded

The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

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

3) init() method is invoked

After the servlet is instantiated successfully, the servlet container **initializes** the instantiated servlet object.

The container initializes the servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts **ServletConfig** object reference as parameter.

The Servlet container invokes **init( )** method only once, immediately after the object is instantiated successfully.

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

Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the **ServletException** or **UnavailableException**.

4) service() method is invoked

After initialization, the Servlet instance is **ready** to **serve** the **client requests**.

The **Servlet container** performs the following operations when the Servlet instance is located to service a request :

- It creates the ServletRequest and ServletResponse objects.
  In this case, if this is a HTTP request, then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively.
- After creating the **request** and **response** objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.
- The **service()** method while processing the request may throw the **ServletException** or **UnavailableException** or **IOException**.

5) destroy() method is invoked

When a Servlet container decides to destroy the Servlet, it performs the following operations,

- It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
- After currently running threads have completed their jobs, the Servlet container calls the **destroy()** method on the Servlet instance.

After the **destroy()** method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

```
public void destroy()
```

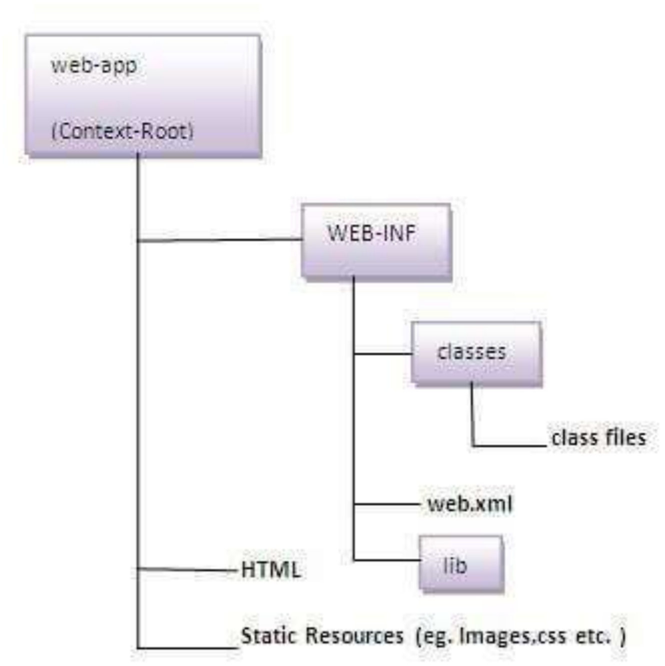
Steps to create a servlet example

There are given 6 steps to create a **servlet example**. These steps are required for all the servers. 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 (web.xml)
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 respond to the client.
The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



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

2)Create a Servlet

There are three ways to create the servlet.

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

The **HttpServlet** class is **widely used to create the servlet** because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.
In this example we are going to create a servlet that extends the HttpServlet class.

My.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class My extends HttpServlet
{
public void service(HttpServletRequest req, HttpServletResponse res)  throws
ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data
pw.println("Hello Servlet"); //writing html in the stream
pw.close();//closing the stream
}
}
```

The **javax.servlet** and **javax.servlet.http** are important packages containing the classes and interfaces that are required for the operation of servlets.
The most commonly used **interface** from **javax.servlet** package is **Servlet.**
Similarly most commonly used **class** in this package is **GenericServlet**.
The **ServletRequest** and **ServletResponse** are another **two commonly used interfaces defined javax.servlet package.**
In the **javax.servlet.http** package **HttpServletRequest** and **HttpServletResponse** are two commonly used interfaces.
The **HttpServletRequest** enables the **servlet** to **read data from the HTTP request** and **HttpServletResponse enables the servlet to write data to HTTP response**.
We have given class name **My** which should be **derived** from the class **HttpServlet.**
(Sometimes we can derive our class from GenericServlet).
Then we have defined **service**() method to which the **HTTP request** and **response** are passed as parameters.
The commonly used basic exceptions for the servlets are **IOException** and **ServletException**.
The **response type** is specified using the **setContentType()** method.
PrintWriter is used **to create object for output stream**.
getWriter() method is **used to for obtaining the output stream from response object**.
Anything written to this stream , is **sent** to **client** as **a response.**
Hence using object of outout stream **pw**, we can write the HTML source code in **println** method as **HTTP response.**

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 Tomcat
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.

web.xml file

```
<web-app>
<servlet>
<servlet-name>My</servlet-name>
<servlet-class>My</servlet-class>
</servlet>

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

Description of the elements of web.xml file

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

<web-app> represents the whole application.
<servlet> is sub element of <web-app> and represents the servlet.
<servlet-name> is sub element of <servlet> represents the name of the servlet.
<servlet-class> is sub element of <servlet> represents the class of the servlet.
<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.
<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5) How to deploy the servlet project

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

6)Start the Server

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

One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

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

1) How to set JAVA_HOME in environment variable?

To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.
Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.
There must not be semicolon (;) at the end of the path.
After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.
Now server is started successfully.

2) How to change port number of apache tomcat

Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.
Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9494 and save this file.

6) How to access the servlet

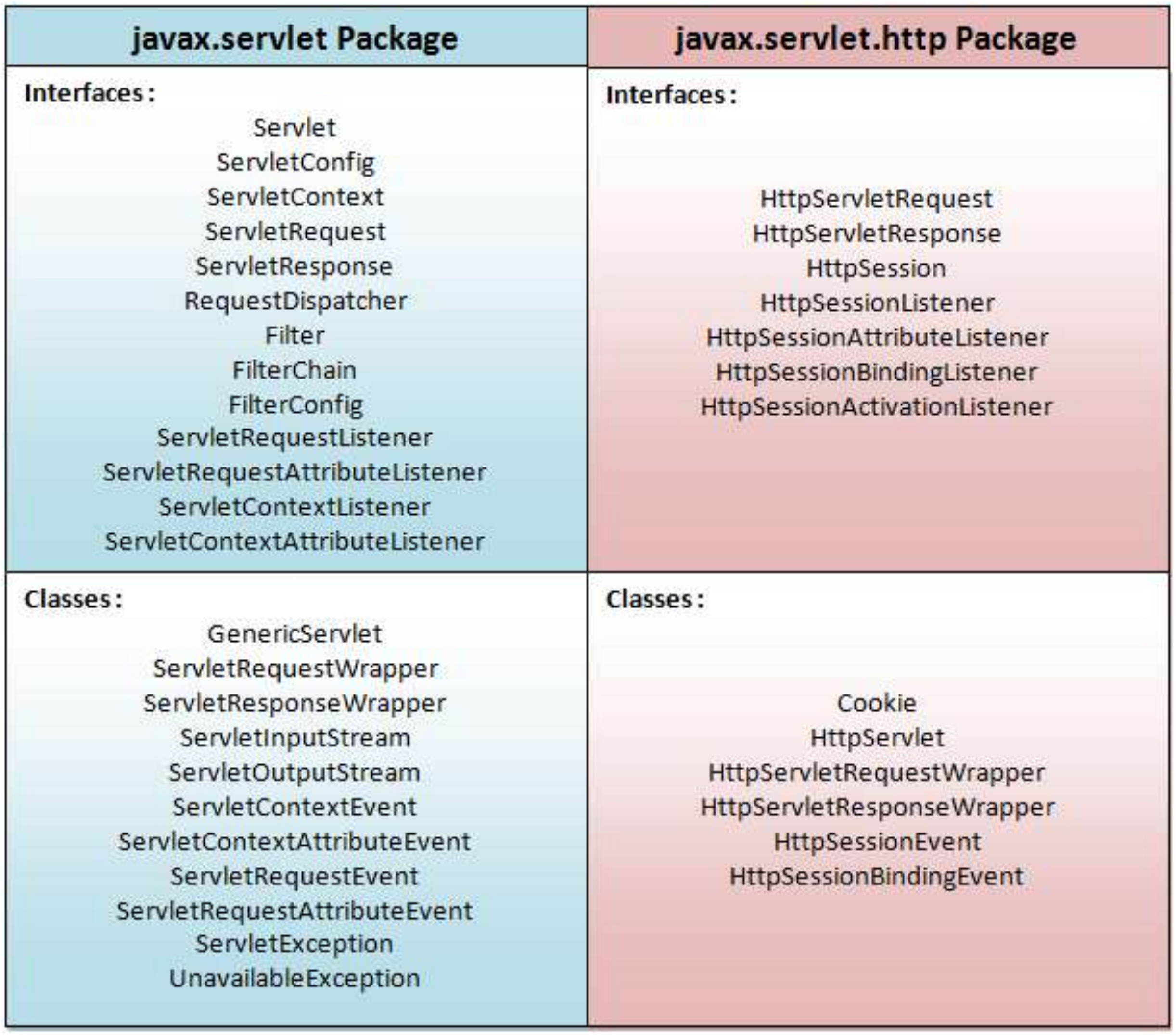
Open broser and write http://hostname:portno/contextroot/urlpatternofservlet.
For example:
http://localhost:9494/demo/welcome

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.



javax.servlet Package	javax.servlet.http Package
Interfaces :	**Interfaces :**
Servlet	HttpServletRequest
ServletConfig	HttpServletResponse
ServletContext	HttpSession
ServletRequest	HttpSessionListener
ServletResponse	HttpSessionAttributeListener
RequestDispatcher	HttpSessionBindingListener
Filter	HttpSessionActivationListener
FilterChain	
FilterConfig	
ServletRequestListener	
ServletRequestAttributeListener	
ServletContextListener	
ServletContextAttributeListener	
Classes :	**Classes :**
GenericServlet	Cookie
ServletRequestWrapper	HttpServlet
ServletResponseWrapper	HttpServletRequestWrapper
ServletInputStream	HttpServletResponseWrapper
ServletOutputStream	HttpSessionEvent
ServletContextEvent	HttpSessionBindingEvent
ServletContextAttributeEvent	
ServletRequestEvent	
ServletRequestAttributeEvent	
ServletException	
UnavailableException	

1] javax.servlet package

Interfaces in javax.servlet package

A] Servlet Interface

Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement.

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

Methods of Servlet interface

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

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

B] 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.

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.

C] 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.

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.

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 void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

D] ServletRequest Interface

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

Methods of ServletRequest interface

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

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.

public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream()	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

E] ServletResponse Interface

The **servlet container** is connected to the **web server** that receives **Http Requests** from **client** on a **certain port**.
When **client sends** a **request** to **web server**, the **servlet container** creates **HttpServletRequest** and **HttpServletResponse objects** and passes them as an argument to the servlet service() method.
The **response object** allows you to **format** and **send** the **response** back to the **client.**

1) String getCharacterEncoding(): It returns the name of the MIME charset used in body of the response sent to the client.
2) String getContentType(): It returns the response content type. e.g. text, html etc.
3) ServletOutputStream getOutputStream(): Returns a ServletOutputStream suitable for writing binary data in the response.
4) java.io.PrintWriter getWriter(): Returns the PrintWriter object.
5) void setCharacterEncoding(java.lang.String charset): Set the MIME charset (character encoding) of the response.
6) void setContentLength(int len): It sets the length of the response body.
7) void setContentType(java.lang.String type): Sets the type of the response data.

2] Classes in javax.servlet package

A] GenericServlet class

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

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

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

Methods of GenericServlet class

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public String getServletName()** returns the name of the servlet object.
7. **public void log(String msg)** writes the given message in the servlet log file.

2] javax.servlet.http package

The javax.servlet.http package supports the development of servlets that use the HTTP protocol. The classes in this package extend the basic servlet functionality to support various HTTP specific features, including request and response headers, different request methods, and cookies. The abstract HttpServlet class extends javax.servlet.GenericServlet and serves as the base class for HTTP servlets.

Interfaces in javax.servlet.http package

A] HttpServletRequest Interface

HttpServletRequest is an **interface** and **extends** the **ServletRequest interface**. By extending the ServletRequest this interface is able to allow request information for HTTP Servlets. Object of the HttpServletRequest is created by the Servlet container and, then, it is passed to the service method (doGet(), doPost(), etc.) of the Servlet.

getAuthType() Returns the name of the authentication scheme used to protect the servlet.
getHeader(java.lang.String name) Returns the value of the specified request header as a String.
getCookies() Returns an array containing all of the Cookie objects the client sent with this request.
getMethod() Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
getPathInfo() Returns any extra path information associated with the URL the client sent when it made this request.
getSession() Returns the current session associated with this request, or if the request does not have a session, creates one.

B] HttpServletResponse

HttpServletResponse is a predefined interface present in javax.servlet.http package. It can be said that it is a mirror image of request object. The response object is where the servlet can write information about the data it will send back. Whereas the majority of the methods in the request object start with GET, indicating that they get a value, many of the important methods in the response object start with SET, indicating that they change some property.

addCookie(Cookie cookie)
Adds the specified cookie to the response.
encodeURL(java.lang.String url)
Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
Boolean containsHeader(java.lang.String name)
Returns a boolean indicating whether the named response header has already been set.
sendError(int sc)
Sends an error response to the client using the specified status code and clearing the buffer.

Classes of javax.servlet.http package

HttpServlet class

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

Methods of HttpServlet class

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

1. **public void service(ServletRequest req, ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** This method is invoked for processing Http Request and response.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
7. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
8. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.

Handling HTTP Request and Response

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** (question mark) symbol as follows −

http://www.test.com/hello?key1 = value1&key2 = value2
Example:
http://localhost:9494/HTTP_PG1/sample?name=Arrow

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string. Servlet handles this type of requests using **doGet()** method.

POST Method

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL **it sends it as a separate message**. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this **type of requests** using **doPost()** method.

Reading Form Data using Servlet

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

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

Example of ServletRequest to display the name of the user

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

Program 1

index.html

```
<html>
<body>
<form name= "HTTP Servlet Request" method=Get action= "http:// localhost:9494/ HTTP_PG1/ sample" >
Enter your name
<input type="text" name="name"><br>
<input type="submit" value="login">
</form>
</body>
</html>
```

sample.java file

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class sample extends HttpServlet{
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
String name=req.getParameter("name");//will return value
pw.println("Welcome "+name);
pw.close();
}
}
```

Example 2:

Index.html

```
<html>
<body>
<h1>The select element</h1>
<p>The select element is used to create a drop-down list.</p>
<form name= "form1" method=Get action="http://localhost:9494/HTTP_PG2/sample">
 <label>Choose a car:</label>
 <select name="cars" id="cars">
  <option value="Maruti">Maruti</option>
  <option value="BMW">BMW</option>
  <option value="Lexus">Lexus</option>
  <option value="Audi">Audi</option>
 </select>
 <br><br>
 <input type="submit" value="Submit">
</form>
</body>
</html>
```

sample.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class sample extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
String name=req.getParameter("cars");//will return value
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
pw.println("Selected car is  "+name);
pw.close();
}}
```

Example 3:- Display the Greatest of the two numbers entered by User

Index.html

```
<html>
<body>
<h1>The select element</h1>
<form name= "form1" method=Get action="http://localhost:9494/HTTP_PG3/sample">
  <label>Enter First Number</label>
  <input type="text" name="Number1" size="5"><br><br>
  <label>Enter Second Number</label>
  <input type="text" name="Number2" size="5">
  <br><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

sample.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class sample extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
int a=Integer.parseInt(req.getParameter("Number1"));
int b=Integer.parseInt(req.getParameter("Number2"));
if (a>b)
        pw.println("Greatest Number is "+a);
else
        pw.println("Greatest Number is "+b);
pw.close();
}}
```

Session Tracking in Servlets

Session simply means a **particular interval of time**.
Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet.
Http protocol is a **stateless** so **we need to maintain state** using **session tracking techniques**. Each time user requests to the server, server treats the request as the **new request**. So we **need to maintain the state of an user** to **recognize to particular user.**

Why use Session Tracking?

It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

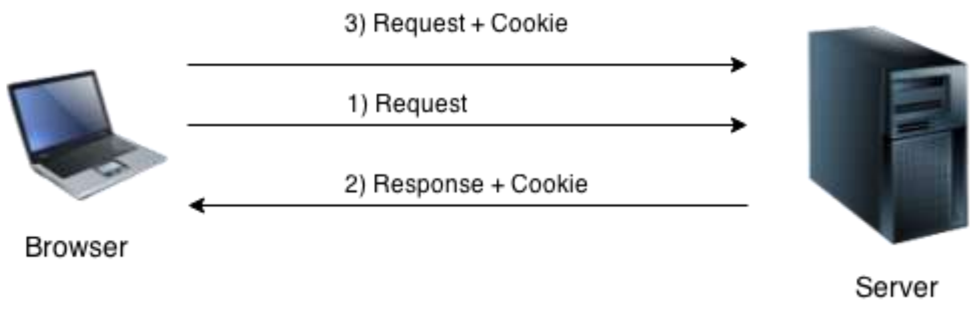
1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

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

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

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

Persistent cookie

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

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.

```
Cookie ck=new Cookie("user","Arrow");//creating cookie object
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.

```
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?

Let's see the simple code to get all the 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
}
```

Program: Set the cookie from the value entered by user.

Index.html

```
<html>
<body>
<form name= "form1" method=post action="http://localhost:9494/Cookie_PG1/sample1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
</body>
</html>
```

sample1.java

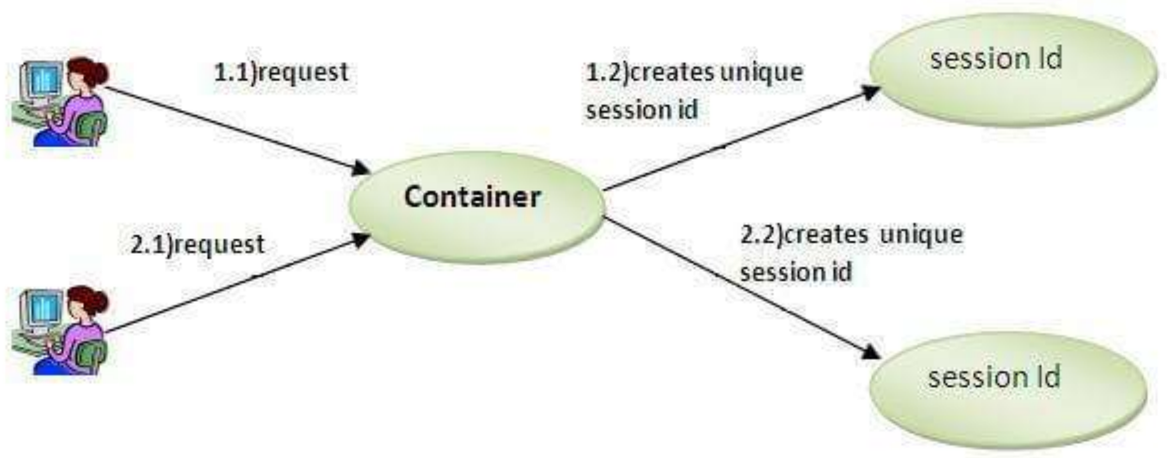
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class sample1 extends HttpServlet
{
  public void doPost(HttpServletRequest request, HttpServletResponse response)
{
    try
    {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    Cookie ck=new Cookie("uname",n);        //creating cookie object
    response.addCookie(ck);                 //adding cookie in the response
    out.println("Your Cookie Added Successfully!!!");
    out.println("Welcome "+n);
    out.close();

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

HttpSession interface

Here 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 method to get the object of HttpSession:

public HttpSession getSession():
A non parameterized getSession() method returns a session, if it already exists or creates a new session if it does not.

```
HttpSession session = request.getSession();
```

We create a new session using the code below:

```
HttpSession session = request.getSession(true);
```

Fetching a pre-existing session

getSession(false) fetches a pre-existing session.

```
HttpSession session = request.getSession(false);
```

Destroy existing session

invalidate() method destroys a session object.

```
session.invalidate();
```

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

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

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);

    HttpSession session=request.getSession();
    session.setAttribute("uname",n);
```

```
        out.print("<a href='SecondServlet'>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();

        HttpSession session=request.getSession(false);
        String n=(String)session.getAttribute("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>
```