**Unit -3**

**SERVLET**

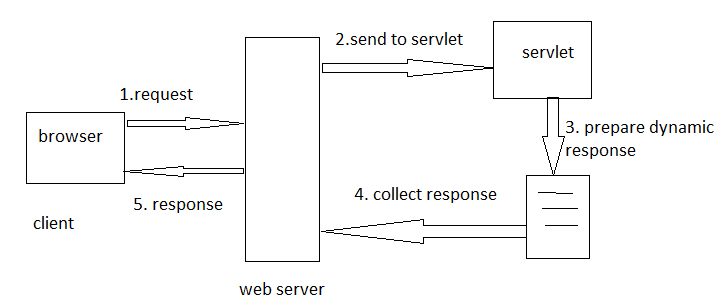
**3.1 Introduction To Servlets**

* servlet technology is used to create a web application
* servlet reside  at the server side and generate the dynamic web pages
* servlet uses HTTP protocol to send  a request from the client
* in servlet the data will be submitted by using the HTML form
* the servlet can process and store the data in the database which is taken from the HTML

**Definition** : Servlet is a class which Run inside the server to produce the dynamic responses to multiple clients across the network

**why servlets are created**:   if you want to produce a dynamic content on the browser we can create the servlet.

**How Sarvlet works**



1. Client send the request by using any Browser
2. the browser send the request to web server .then server send the request to corresponding  servlet present in the Server
3. Servlet  gather relevant information and create a dynamic page
4. this dynamic page is given back to the web server
5. finally the web server send this page to the client

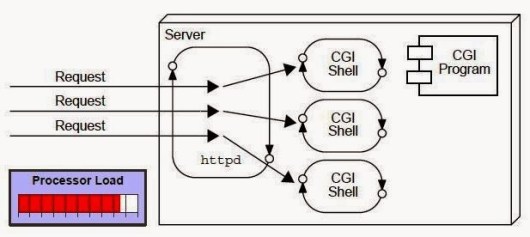
Note:  servlets are platform independent program that can execute on any different web server

3.2 Common Gateway Interface (CGI)

Before servlet CGI programming is used to create webapplications.

The **Common Gateway Interface (CGI)** provides the middleware between WWW servers and external databases and information sources. The World Wide Web Consortium (W3C) defined the Common Gateway Interface (CGI) and also defined how a program interacts with a Hyper Text Transfer Protocol (HTTP) server.

The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This process or convention for passing data back and forth between the server and the application is called the common gateway interface (CGI).



When client sends a request for data on the server  the web server locate the CGi program and pass the request to CGI Program

 the cgi program decode the client information and perform Necessary computation

 output of the CGi program sent back to the client

**steps in CGI program execution**

1. user click the URL link to dynamic pages
2. this URL decide which CGI program to be executed
3. web server run the PGI program in a separate operating system shell. shell includes operating system environment and processes to execute the CGI program
4. CGI  responses is sent back to the web server. it wap the responses in http responses format and send it back to the client

NOTE : for  every request it starts a new process

**Disadvantages of CGI**

1) for every request a new process will be started but the web server has a limited capacity to start the processes

2) it uses the platform dependent languages like c, c++, perl

3) CGI programs are not scalable.i.e  Whenever number of client requests are increased the server not in a position to handle each request

4) CGI is not secure and not a object oriented program

5) programs are written in perl programming language. as a developer every programmer need to learn perl programs to develop the CGI application

6) CGI will increase the burden on the server

|  |  |  |
| --- | --- | --- |
| BASIS FOR COMPARISON | CGI | SERVLET |
| Basic | Programs are written in the native OS. | Programs will be written by using Java. |
| Platform dependency | Platform dependent | Does not rely on the platform |
| Creation of process | Each client request creates its own process. | Processes are created depending on the type of the client request. |
| Conversion of the script | Present in the form of executables (native to the server OS). | Compiled to Java Bytecode. |
| Runs on | Separate process | JVM |
| Security | More vulnerable to attacks. | Can resist attacks. |
| Speed | Slower | Faster |
| Processing of script | Direct | Before running the scripts it is translated and compiled. |
| Portability | Can not be ported | Portable |

**3.3 Deployment of servlet or how to execute servlet**

We use 6 steps to execute or deploy servlet

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. i.e call servlet from browser

### Create a directory structures

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

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



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

### 2)Create a Servlet

|  |
| --- |
| There are three ways to create the servlet.   1. By implementing the Servlet interface 2. By inheriting the GenericServlet class 3. By inheriting the HttpServlet class   The HttpServlet class is widely used to create the servlet because it provides methods to handle  http requests such as doGet(), doPost, doHead() etc. |
| In this example we are going to create a servlet that extends the HttpServlet class. In this example,  we are inheriting the HttpServlet class and providing the implementation of the doGet() method.  Notice that get request is the default request. |

**DemoServlet.java**

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

### 3)Compile the servlet

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

|  |  |
| --- | --- |
| **Jar file** | **Server** |
| 1) servlet-api.jar | Apache 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.

### 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>**
3. **<servlet>**
4. **<servlet-name>**sonoojaiswal**</servlet-name>**
5. **<servlet-class>**DemoServlet**</servlet-class>**
6. **</servlet>**
8. **<servlet-mapping>**
9. **<servlet-name>**sonoojaiswal**</servlet-name>**
10. **<url-pattern>**/welcome**</url-pattern>**
11. **</servlet-mapping>**
13. **</web-app>**

### Description of the elements of web.xml file

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

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

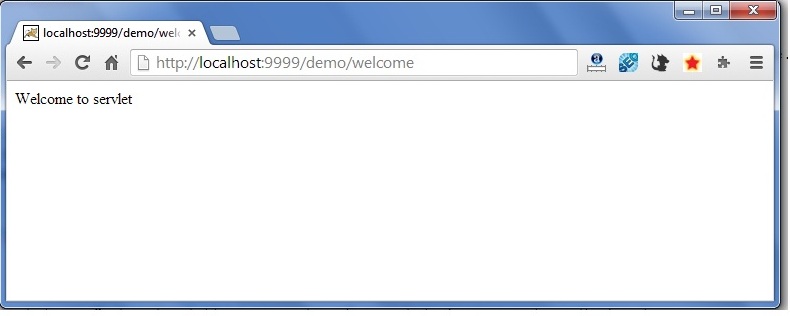
### 5)Start the Server and deploy the project

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

### 6) How to access the servlet

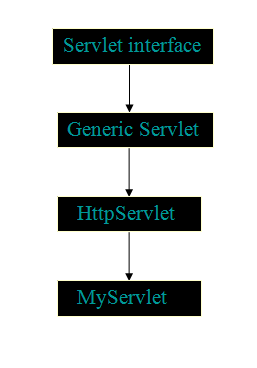
Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

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



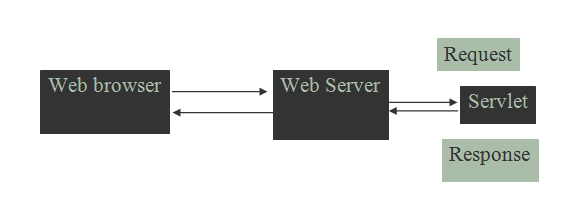
# 3.3 Servlet Architecture

A Servlet is a class, which implements the javax.servlet.Servlet interface. However instead of directly implementing the javax.servlet.Servlet interface we extend a class that has implemented the interface like javax.servlet.GenericServlet or javax.servlet.http.HttpServlet.



#### Servlet Exceution

This is how a servlet execution takes place when client (browser) makes a request to the webserver.



### Servlet architecture includes:

a) **Servlet Interface**  
To write a servlet we need to implement Servlet interface. Servlet interface can be implemented directly or indirectly by extending **GenericServlet** or **HttpServlet** class.

b) **Request handling methods**  
There are 3 methods defined in Servlet interface: **init(), service() and destroy()**.

The first time a servlet is invoked, the init method is called. It is called only once during the lifetime of a servlet. So, we can put all your initialization code here.

The Service method is used for handling the client request. As the client request reaches to the container it creates a thread of the servlet object, and request and response object are also created.

These request and response object are then passed as parameter to the service method,( for Servlet interface and GenericServlet) which then process the client request. The service method again in turn calls the doGet or doPost methods if the user has extended the class from HttpServlet .

### Basic Structure of a Servlet using HttpServlet

public class firstServlet extends HttpServlet {

public void init() {

/\* Put your initialization code in this method,

\* as this method is called only once \*/

}

public void service() {

// Service request for Servlet

}

public void destroy() {

// For taking the servlet out of service, this method is called only once

}

}

Service method call doGet() or doPost() methods

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Creating Http Servlet by Extending HttpServlet class

public class ExampleHttpServlet extends HttpServlet

{

private String mymsg;

public void init() throws ServletException

{

mymsg = "Http Servlet Demo";

}

public void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException

{

// Setting up the content type of web page

response.setContentType("text/html");

// Writing the message on the web page

PrintWriter out = response.getWriter();

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

out.println("<p>" + "Hello Friends!" + "</p>");

}

public void destroy()

{

// Leaving empty. Use this if you want to perform

//something at the end of Servlet life cycle.

}

}

3.5 **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.

**Stages of the Servlet Life Cycle**: The Servlet life cycle mainly goes through four stages,

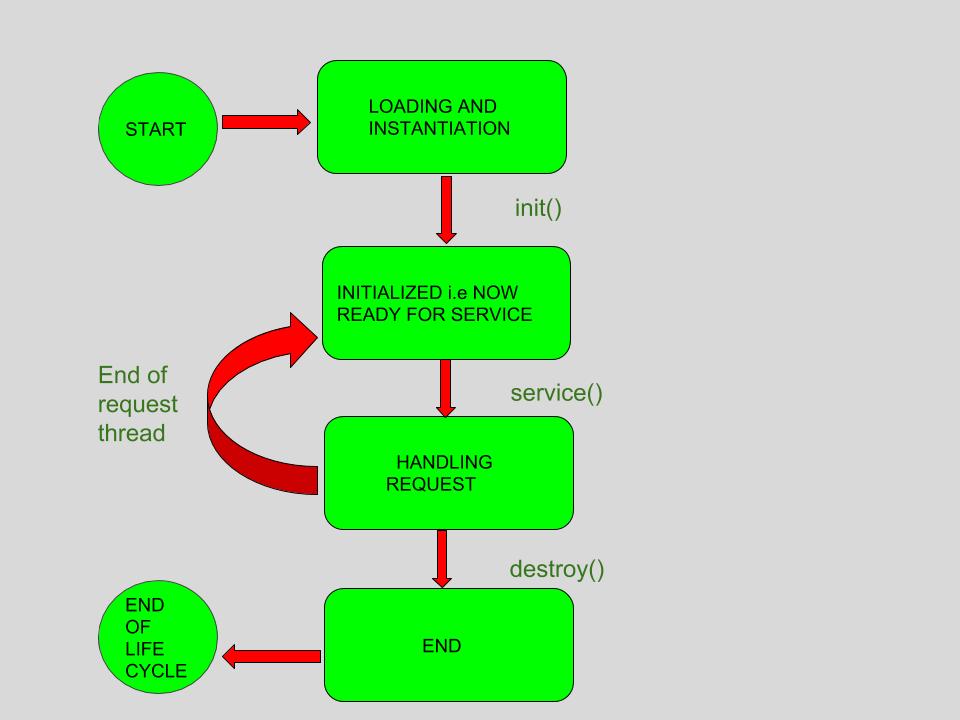
* Loading a Servlet.
* Initializing the Servlet.
* Request handling.
* Destroying the Servlet.

Let’s look at each of these stages in details:

1. **Loading a Servlet**: The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :
   * Initializing the context, on configuring the Servlet with a zero or positive integer value.
   * If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

* + **Loading :** Loads the Servlet class.
  + **Instantiation :** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.



1. **Initializing a Servlet**: 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 the **Servlet.init(ServletConfig)** method only once, immediately after the **Servlet.init(ServletConfig)** object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

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

1. **Handling request**: 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**.

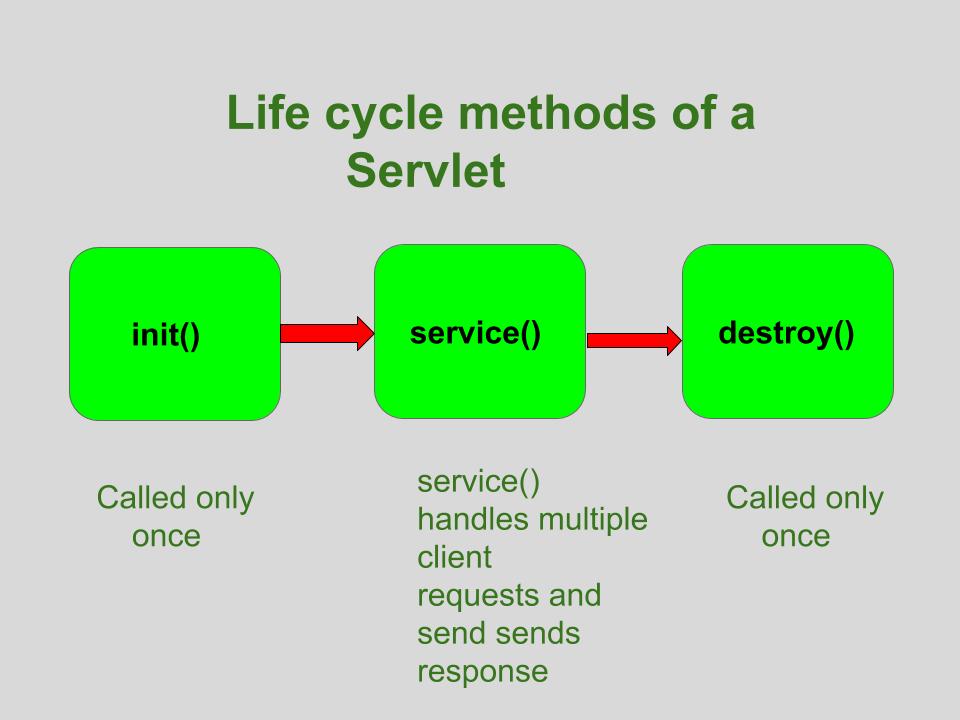
1. **Destroying a Servlet**: When a Servlet container decides to destroy the Servlet, it performs the following operations,
   * It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
   * After currently running threads have 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.

**Servlet Life Cycle Methods**

There are three life cycle methods of a Servlet :

* init()
* service()
* destroy()



Let’s look at each of these methods in details:

1. **init() method**: The **Servlet init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.
2. //init() method
3. **public class MyServlet implements Servlet**
4. **{**
5. **public void init(ServletConfig config) throws ServletException**
6. **{**
7. **//initialization code**
8. **}**
9. **//rest of code**
10. **}**

**service() method**: The **service()** method of the Servlet is invoked to inform the Servlet about the client requests.

* + This method uses **ServletRequest** object to collect the data requested by the client.
  + This method uses **ServletResponse** object to generate the output content.

// service() method

1. **public class MyServlet implements Servlet**
2. **{**
3. **public void service(ServletRequest res, ServletResponse res)**
4. **throws ServletException, IOException**
5. **{**
6. **// request handling code**
7. **}**
8. **// rest of code**
9. **}**
10. **destroy() method**: The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.
11. //destroy() method
12. **public void destroy()**

As soon as the **destroy()** method is activated, the Servlet container releases the Servlet instance.

**Servlet Life Cycle in depth knowledge (OUT OF SYLLABUS ) :**  
Servlet life cycle can be defined as the stages through which the servlet passes from its creation to its destruction.  
The servlet life cycle consists these stages:

* Servlet is borned
* Servlet is initialized
* Servlet is ready to service
* Servlet is servicing
* Servlet is not ready to service
* Servlet is destroyed

**Life cycle methods:**  
Life cycle methods are those methods which are used to control the life cycle of the servlet. These methods are called in specific order during the servlets’s entire life cycle.

The class **Servlet** provides the methods to control and supervise the life cycle of servlet. There are three life cycle methods in the Servlet interface. There are as follows:

* **init() method :**
  1. A servlet’s life begins here .
  2. This method is called only once to load the servlet.Since it is called only once in it’s lifetime,therefore “connected architecture” code is written inside it because we only want once to get connected with the database.
  3. **Now Question Arises is that:-**  
     Q.Why can’t we write connected architecture code( i.e data base connection code ) inside the constructor, since constructor also run only once in it’s entire life?

Ans. Suppose if the connection doesn’t get established, then we can throw an exception from init() and the rest of the steps stop executing. But in the constructor we can’t use, throw in it’s prototype otherwise it is an error.

* 1. This method receives only one parameter, i.e **ServletConfig** object.
  2. This method has the possibility to throw the ServletException.
  3. Once the servlet is initialized, it is ready to handle the client request.
  4. The prototype for the init() method:

public void init(ServletConfig con)throws ServletException{ }

where **con** is ServletConfig object

* **NOTE:-** In programs of servlet,we use non parameterized version of init().

* **Now,Question Arises is that:-**  
  Q. Why it is recommended to use the non parameterized version of init() instead of parameterized version as seen above?

Ans. To answer this, we have to go into detail. Think like developers,i.e there must be some valid reason for this and the answer will blow your mind. Coming to answer:

* **APPROACH 1**  
  Whenever the lifecycle method of a servlet starts executing,i.e when public void init(ServletConfig con) throws ServletException gets call then our class public void init(ServletConfig con) throws ServletException gets called but we have to run the code which initializes servlet config object which is written inside “HttpServlet” method

public void init(ServletConfig con) throws ServletException,i.e:  
Coding of HttpServlet class be like:

public void init(ServletConfig con) throws ServletException

{

//code to initialise ServletConfig object

init(); //This HttpServlet has 2 init() one which is parameterized and the other one is

//non parameterized.But this non parameterized version of init() has a blank body.

//So this call is useless.

}

Now see the coding of our class

public void init(ServletConfig con) throws ServletException

{

super.init(con); //Since,our class init() will run first,but to run HttpServlet init() we

// have used super keyword.And Database connectivity code will be their

}

**NOTE:-** As we can see, total 3 init() calls we have to make.First init() gets called of our class then of HttpServlet class then non parameterized version of HttpServlet class.

But now, we will achieve the same thing with less number of calls:

**APPROACH 2**  
Coding of HttpServlet parametrized and non parameterized versions of init() will remain the same. **But in our class instead of overriding parameterized version of init(), we will override non parameterized version of init()**.

Let’s see the coding of our class non parameterized version of init():

public void init() throws ServletException

{

//database connectivity code

}

**NOTE:** Since this method public void init() throws ServletException ,we have override from HttpServlet class whose coding is like:

public void init() throws ServletException

{

//empty body

}

Since it’s body is blank, therefore it is known as **“Helper method”** as it is used for overriding purpose.

Now, as the servlet starts executing its methods, it will call the parameterized version of init(). Since we have not to override the parameterized version, therefore it will give a call to the HttpServlet parameterized version of init().

Since coding of a parameterized version of init() of HttpServlet is as same as above, therefore, from there on it will call init() (i.e non parameterized version of init). It will give a call to our class non parameterized version of init() and the code continues.

**Now, as you can see, total number of init() calls are 2 which is less than the first approach. Therefore, execution time is less in 2nd approach and less headache for CPU for maintaining stack and it’s speed increases as compared to 1st approach.**

Therefore, it is highly recommended to override non parameterized version of init().Although both will run but due to efficiency first approach is rarely used and also in first approach we have to use super keyword too.

Therefore in below mentioned program,we have override non parameterized version of init().

**service() method :**

* 1. The service() method is the most important method to perform that provides the connection between client and server.
  2. The web server calls the service() method to handle requests coming from the client( web browsers) and to send response back to the client.
  3. This method determines the type of Http request (GET, POST, PUT, DELETE, etc.) .
  4. This method also calls various other methods such as doGet(), doPost(), doPut(), doDelete(), etc. as required.
  5. This method accepts two parameters.
  6. The prototype for this method:
  7. public void service(ServletRequest req, ServletResponse resp)

throws ServletException, IOException { }

where

* + - **req** is the ServletRequest object which encapsulates the connection from client to server
    - **resp** is the ServletResponse object which encapsulates the connection from server back to the client

**destroy() method :**

* 1. The destroy() method is called only once.
  2. It is called at the end of the life cycle of the servlet.
  3. This method performs various tasks such as closing connection with the database, releasing memory allocated to the servlet, releasing resources that are allocated to the servlet and other cleanup activities.
  4. When this method is called, the garbage collector comes into action.
  5. The prototype for this method is:

public void destroy() { // Finalization code...}

**3.6 Servlet API**

You need to use Servlet API to create servlets. There are two packages that you must remember while using API, the javax.servlet package that contains the classes to support generic servlet (protocol-independent servlet) and the javax.servlet.http package that contains classes to support http servlet. You may be wondering what is generic and http servlet, I have explained them later in this post.

**Let’s see the hierarchy of packages:**

java.lang.Object

|\_extended byjavax.servlet.GenericServlet

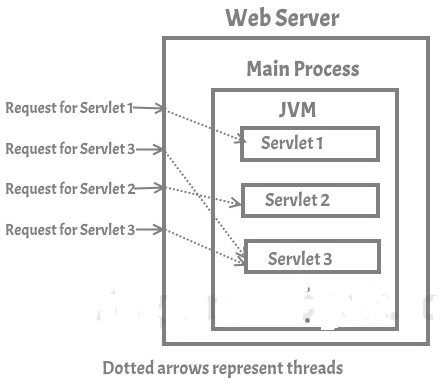
|\_extended byjavax.servlet.http.HttpServlet

Every Servlet must implement the java.servlet.Servlet interface, you can do it by extending one of the following two classes:

 javax.servlet.GenericServlet or

javax.servlet.http.HttpServlet.

The first one is for protocol independent Servlet and the second one for http Servlet.

**How servlet works?**  


## Generic Servlet

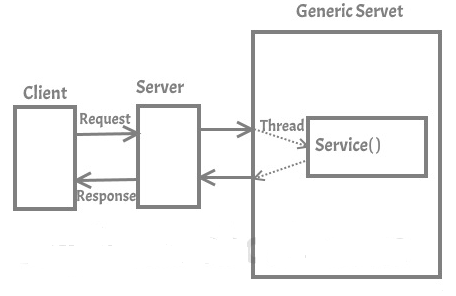
As I mentioned above, if you are creating a Generic Servlet then you must extend javax.servlet.GenericServlet class. GenericServlet class has an abstract service() method. Which means the subclass of GenericServlet should always override the service() method.

**Signature of service() method:**

public abstract void service(ServletRequest request, ServletResponse response)

throws ServletException, java.io.IOException

The service() method accepts two arguments ServletRequest object and ServletResponse object. The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

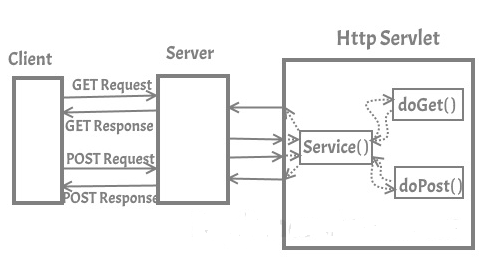


## HTTP Servlet

If you creating Http Servlet you must extend javax.servlet.http.HttpServlet class, which is an abstract class. Unlike Generic Servlet, the HTTP Servlet doesn’t override the service() method. Instead it overrides one or more of the following methods. It must override at least one method from the list below:

* **doGet()** – This method is called by servlet service method to handle the HTTP GET request from client. The Get method is used for getting information from the server
* **doPost()** – Used for posting information to the Server
* **doPut()** – This method is similar to doPost method but unlike doPost method where we send information to the server, this method sends file to the server, this is similar to the FTP operation from client to server
* **doDelete()** – allows a client to delete a document, webpage or information from the server
* **init() and destroy()** – Used for managing resources that are held for the life of the servlet
* **getServletInfo()** – Returns information about the servlet, such as author, version, and copyright.

**In Http Servlet there is no need to override the service() method** as this method dispatches the Http Requests to the correct method handler, for example if it receives HTTP GET Request it dispatches the request to the doGet() method.



#### Interfaces in javax.servlet package

* Servlet-----it define all life cycle methods
* ServletRequest----reading data from client
* ServletResponse-----write data to client response
* ServletConfig-----it is used to obtain initialization parameters
* ServletContext----
* SingleThreadModel
* RequestDispatcher
* ServletRequestListener
* ServletRequestAttributeListener
* ServletContextListener
* ServletContextAttributeListener
* Filter
* FilterConfig
* FilterChain

#### Classes in javax.servlet package

* GenericServlet-----it implements servlet and servletconfig interfaces
* ServletInputStream---------provide inputstream for reading client request
* ServletOutputStream----provide outputstream to writeclient response.
* ServletException----raise exception when error occurred.
* ServletRequestWrapper
* ServletRequestEvent
* ServletResponseWrapper
* ServletContextEvent
* ServletRequestAttributeEvent
* ServletContextAttributeEvent
* UnavailableException

#### Interfaces in javax.servlet.http package

* HttpSession----session data can read or write using this interface
* HttpServletRequest-----read information from http request
* HttpServletResponse---write data to http response using this interface
* HttpSessionAttributeListener
* HttpSessionListener
* HttpSessionBindingListener
* HttpSessionActivationListener
* HttpSessionContext

#### Classes in javax.servlet.http package

* HttpServlet---used to creat servlet
* Cookie-----used to create cookies
* HttpSessionEvent---used to handle session events
* HttpSessionBindingEvent
* HttpServletRequestWrapper
* HttpServletResponseWrapper
* HttpUtils

**Reading servlet parameters :**

* ServletRequest interface object is used to store client form data.
* Client information is stored in ServletRequest object.
* ServletRequest interface provide some methods to receive client information.

## Methods of ServletRequest interface

**String getParameter(String name)**: It returns the value of the given parameter as String or null if the given parameter does not exist.

**Enumeration getParameterNames()**: It returns an Enumeration of Strings objects containing the names of parameters in the request.

**String[] getParameterValues(String name)**: It returns an array of Strings containing the all the values, the parameters has, returns null if parameter doesn’t have any value.

**String getCharacterEncoding()**: Returns the name of the character encoding used in the body of this request. This method returns null if the request does not specify a character encoding.

**void setCharacterEncoding(String env)**: Overrides the character encoding in the body of the request.

**int getContentLength()**: Returns the length of the request content in bytes.

**String getContentType()**: Returns the MIME type of the body of the request, or null if the type is not known.

**Reading initialization parameters in servlet:**

Servlet Container creates ServletConfig object for each Servlet during initialization, to pass information to the Servlet. This object can be used to get configuration information such as parameter name and values from deployment descriptor file(web.xml).

## Methods of ServletConfig interface

**public String getInitParameter(String name)**: Returns the value of given parameter as String, or null if the given parameter doesn’t exist in web.xml.  
**public Enumeration getInitParameterNames()**: Returns an enumeration of all the parameter names.  
**public String getServletName()**: Returns the name of the servlet instance.  
**public ServletContext getServletContext()**: Returns an object of ServletContext.

## Example:

In this example, we will use two methods getInitParameter() and getInitParameterNames() to get all the parameters from web.xml along with their values.  
The getInitParameterNames() method returns an enumeration of all parameters names and by passing those names during the call of getInitParameter() method, we can get the corresponding parameter value from web.xml.

**DemoServlet.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.util.Enumeration;

public class DemoServlet extends HttpServlet {

protected void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

response.setContentType("text/html;charset=UTF-8");

PrintWriter pwriter = response.getWriter();

ServletConfig sc=getServletConfig();

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

String str;

while(e.hasMoreElements()) {

str=e.nextElement();

pwriter.println("<br>Param Name: "+str);

pwriter.println(" value: "+sc.getInitParameter(str));

}

}

}

**web.xml**

<web-app>

<display-name>BeginnersBookDemo</display-name>

<welcome-file-list>

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

</welcome-file-list>

<servlet>

<servlet-name>MyServlet</servlet-name>

<servlet-class>DemoServlet</servlet-class>

<init-param>

<param-name>MyName</param-name>

<param-value>Chaitanya</param-value>

</init-param>

<init-param>

<param-name>MyWebsite</param-name>

<param-value>Beginnersbook.com</param-value>

</init-param>

</servlet>

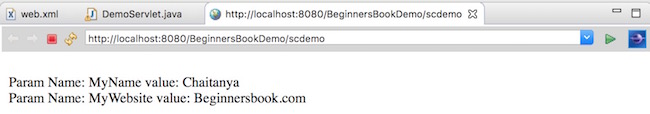
<servlet-mapping>

<servlet-name>MyServlet</servlet-name>

<url-pattern>/scdemo</url-pattern>

</servlet-mapping>

</web-app>

**Output:**  


## ServletRequest Interface

The **ServletRequest** Interface is used to handle client request to access a servlet. It provides the information of a servlet like content type, content length, parameter names and values etc.

## ServletRequest Interface Methods

|  |  |
| --- | --- |
| **Methods** | **Description** |
| public Object getAttribute(String name) | Returns the value of the named attribute as an Object. |
| public int getContentLength( ) | Returns the length of the request body in bytes. |
| public String getContentType( ) | Returns the MIME type of the request body. |
| public String getParameter(String name) | Returns the value of a request parameter as a string. |
| public String getProtocol( ) | Returns the name and version of the request protocol. |
| public int getRemotePort( ) | Returns the Internet Protocol (IP) port of the client. |
| public String getServerName( ) | Returns the host name of the server to which the request was sent. |

**Following are the important methods of ServletRequest Interface:**

## ServletResponse Interface

The **ServletResponse** interface defines an object to help a Servlet in sending a response to the client. It has various methods that help a servlet to respond to the client requests.

## ServletResponse Interface Methods

|  |  |
| --- | --- |
| **Methods** | **Description** |
| public void flushBuffer( ) | It forces the content in the buffer to be written to the client. |
| public int getBufferSize( ) | Returns the actual buffer size. |
| public String getContentType( ) | Returns the content type used to send this response. |
| public PrintWriter getWriter( ) | Returns the PrintWriter object that can be used to send character text to the client. |
| public void reset( ) | Clears the buffer as well as status code and header data. |
| public void setBufferSize(int size) | Sets the specified buffer size for the body of the response. |
| public void setContentType(String type) | Sets the content type of the response being sent to the client. |

**3.7 Handling Http Request and Respnse**

HttpServlet class provide some methods to handle http request

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

### Methods of HttpServlet class

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

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. For an example,  It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. you can use it to post an image file or a large file to the server. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.

#### Example : Request to take username & password. Display username.

**//index.html**  
  
<form action="test" method="post">  
     User Name: <input type="text" name="uname"><br>  
     Password: <input type = "password name = "password"><br>  
     <input type="submit" value="Log In">  
</form>  
  
**//web.xml**  
<web-app>  
<servlet>  
    <servlet-name>test</servlet-name>  
    <servlet-class>ServletDemo</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>test</servlet-name>  
    <url-pattern>/test</url-pattern>  
</servlet-mapping>  
</web-app>  
**//ServletDemo.java**  
  
import java.io.\*;  
import javax.servlet.\*;  
import javax.servlet.http.\*;  
public class ServletDemo extends HttpServlet  
{  
      protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException  
      {  
          response.setContentType("text/html");  
          PrintWriter pw = response.getWriter();  
          try  
          {  
              String username = req.getParameter("username");  
              String password = req.getParameter("password");  
              pw.println("<h1> Hello"+username+"<h1>");  
          }  
          finally  
          {  
              pw.close();  
          }  
      }  
}

## 3.7.1 Get vs. Post

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

|  |  |
| --- | --- |
| **GET** | **POST** |
| 1) In case of Get request, only **limited amount of data**can be sent because data is sent in header. | In case of post request, **large amount of data**can be sent because data is sent in body. |
| 2) Get request is **not secured**because data is exposed in URL bar. | Post request is **secured**because data is not exposed in URL bar. |
| 3) Get request **can be bookmarked.** | Post request **cannot be bookmarked.** |
| 4) Get request is **idempotent**. It means second request will be ignored until response of first request is delivered | Post request is **non-idempotent.** |
| 5) Get request is **more efficient**and used more than Post. | Post request is **less efficient**and used less than get. |
| 6)get request can be cached | 6) post request can’t be cashed |
| 7)get request remain in browser history | 7) post request not store information in browser history |
| 8) get request is never used when dealing with sensitive information | 8)post always used when dealing with sensitive data |
| 9) GET is mostly used for view purpose (e.g. SQL SELECT) | 9) POST is mainly use for update purpose (e.g. SQL INSERT or UPDATE). |

# 3.8 Session Tracking in Servlets

**By default http protocol is state less protocol.i.e it did not remember the first request (made by same client) information while sending the second request by same client.**

**So we need to keep the same client information among multiple request and response cycles**

**By using session tracking mechanism we can make http protocol as state protocol. i.e it remember the first request (made by same client) information while sending the second request by same client .**

However, in Http protocol, the state of the communication is not maintained. Hence, the web applications that work on http protocol use several different technologies that comprise **Session Tracking**, which means maintaining the state (data) of the user, in order to recognize him/her.

In order to achieve session tracking in servlets, cookies have been one of the most commonly used tech. However, they have the following disadvantages:

* They can only keep textual information.
* They’re browser dependent. Hence, if the client disables them, your web application can’t make use of them
* Individual cookie can contain not more than 4kb of information

**Session** simply means a particular interval of time.

**Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

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

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



### Why use Session Tracking?

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

### Session Tracking Techniques

There are four techniques used in Session tracking:

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

**3.8.1 Session tracking using cookies:**

# Cookies in Servlet

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

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

### How Cookie works

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



### Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

### Non-persistent cookie

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

### Persistent cookie

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

### Advantage of Cookies

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

### Disadvantage of Cookies

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

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

### Cookie class

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

### Constructor of Cookie class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| Cookie() | constructs a cookie. |
| Cookie(String name, String value) | constructs a cookie with a specified name and value. |

### Useful Methods of Cookie class

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

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

### Other methods required for using Cookies

|  |
| --- |
| For adding cookie or getting the value from the cookie, we need some methods provided by other  interfaces. They are:   1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to   add cookie in response object.   1. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return 2. 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.\*;

6. **public** **class** FirstServlet **extends** HttpServlet {
8. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){
9. **try**{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. Cookie ck=**new** Cookie("uname",n);//creating cookie object
18. response.addCookie(ck);//adding cookie in the response
20. //creating submit button
21. out.print("<form action='servlet2'>");
22. out.print("<input type='submit' value='go'>");
23. out.print("</form>");
25. out.close();
27. }**catch**(Exception e){System.out.println(e);}
28. }
29. }

### SecondServlet.java

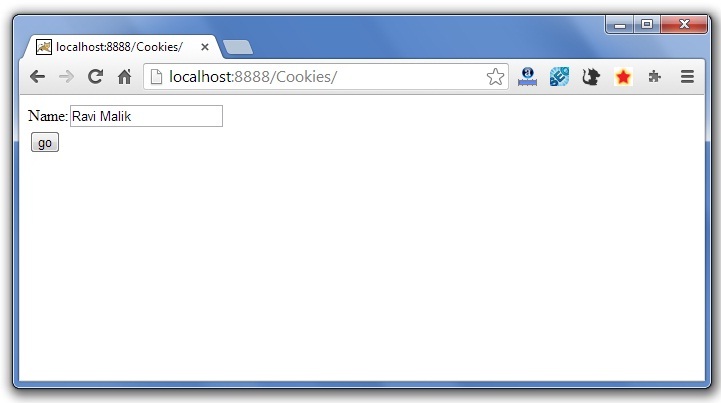
1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** SecondServlet **extends** HttpServlet {
7. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){
8. **try**{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. Cookie ck[]=request.getCookies();
14. out.print("Hello "+ck[0].getValue());
16. out.close();
18. }**catch**(Exception e){System.out.println(e);}
19. }

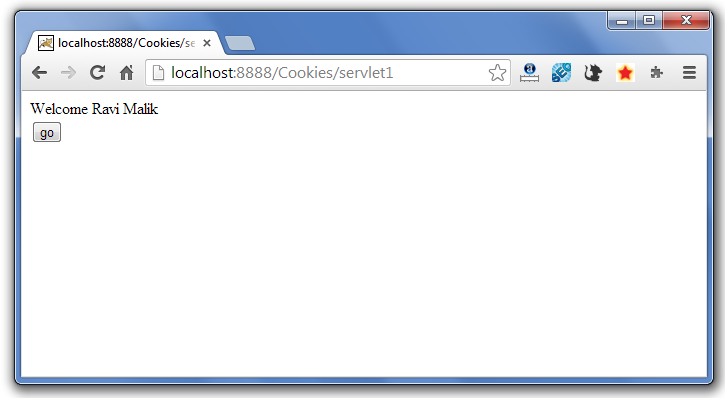
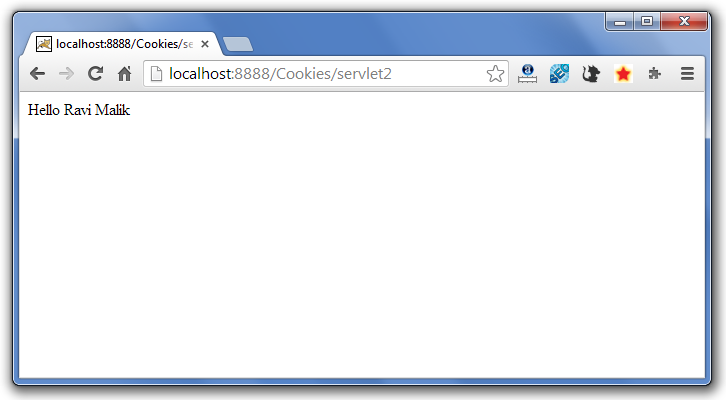
22. }

### web.xml

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

Output:



**3.8.12 Session tracking using HttpSession:**

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.

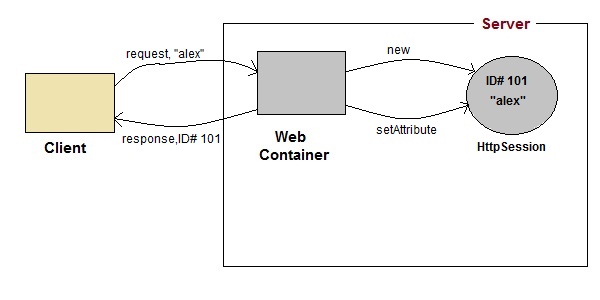


### How to get the HttpSession object ?

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.

## How HttpSession works



1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

## Servlet: HttpSession Interface



When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session. This unique ID can be stored into a cookie or in a request parameter.

The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file( web.xml).

The default timeout value is 30 minutes, this is used if you don’t specify the value in tag. This means that when the user doesn’t visit web application time specified, the session is destroyed by servlet container. The subsequent request will not be served from this session anymore, the servlet container will create a new session.

This is how you create a HttpSession object.

protected void doPost(HttpServletRequest req,

HttpServletResponse res)

throws ServletException, IOException {

HttpSession session = req.getSession();

}

You can store the user information into the session object by using setAttribute() method and later when needed this information can be fetched from the session. This is how you store info in session. Here we are storing username, emailid and userage in session with the attribute name uName, uemailId and uAge respectively.

session.setAttribute("uName", "ChaitanyaSingh");

session.setAttribute("uemailId", "myemailid@gmail.com");

session.setAttribute("uAge", "30");

This First parameter is the attribute name and second is the attribute value. For e.g. uName is the attribute name and ChaitanyaSingh is the attribute value in the code above.

TO get the value from session we use the getAttribute() method of HttpSession interface. Here we are fetching the attribute values using attribute names.

String userName = (String) session.getAttribute("uName");

String userEmailId = (String) session.getAttribute("uemailId");

String userAge = (String) session.getAttribute("uAge");

## Methods of HttpSession

**public void setAttribute(String name, Object value)**: Binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.

**public Object getAttribute(String name)**: Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.

**public Enumeration getAttributeNames()**: Returns an Enumeration that contains the name of all the objects that are bound as attributes to the session object.

**public void removeAttribute(String name)**: Removes the given attribute from session.

**setMaxInactiveInterval(int interval)**: Sets the session inactivity time in seconds. This is the time in seconds that specifies how long a sessions remains active since last request received from client.

For the complete list of methods, refer the

## Session Example

index.html

<form action="login">

User Name:<input type="text" name="userName"/><br/>

Password:<input type="password" name="userPassword"/><br/>

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

</form>

MyServlet1.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MyServlet1 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){

try{

response.setContentType("text/html");

PrintWriter pwriter = response.getWriter();

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

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

pwriter.print("Hello "+name);

pwriter.print("Your Password is: "+password);

HttpSession session=request.getSession();

session.setAttribute("uname",name);

session.setAttribute("upass",password);

pwriter.print("<a href='welcome'>view details</a>");//response.sendRedirect(Welcome) #its url pattern value

pwriter.close();

}catch(Exception exp){

System.out.println(exp);

}

}

}

MyServlet2.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MyServlet2 extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){

try{

response.setContentType("text/html");

PrintWriter pwriter = response.getWriter();

HttpSession session=request.getSession(false);

String myName=(String)session.getAttribute("uname");

String myPass=(String)session.getAttribute("upass");

pwriter.print("Name: "+myName+" Pass: "+myPass);

pwriter.close();

}catch(Exception exp){

System.out.println(exp);

}

}

}

web.xml

<web-app>

<servlet>

<servlet-name>Servlet1</servlet-name>

<servlet-class>MyServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Servlet1</servlet-name>

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

</servlet-mapping>

<servlet>

<servlet-name>Servlet2</servlet-name>

<servlet-class>MyServlet2</servlet-class>

</servlet>

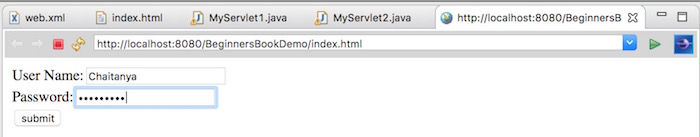
<servlet-mapping>

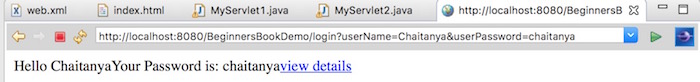
<servlet-name>Servlet2</servlet-name>

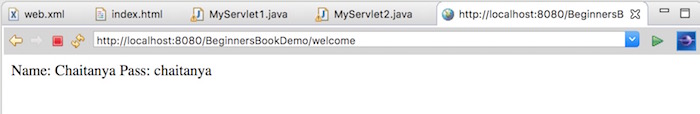
<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

**Output:**  
**First Screen:**  


**After clicking Submit:**  


**After clicking view details:**  


**3.10 connecting to data base using JDBC in servlet:**

|  |
| --- |
| There are 5 steps to connect any java application with the database using JDBC.  These steps are as follows:   * Register the Driver class * Create connection * Create statement * Execute queries * Close connection |

### 1) Register the driver class

|  |
| --- |
| The **forName()** method of Class class is used to register the driver class. This method is used  to dynamically load the driver class. |

### Syntax of forName() method

1. **public** **static** **void** forName(String className)**throws** ClassNotFoundException

### Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

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

### 2) Create the connection object

|  |
| --- |
| The **getConnection()** method of DriverManager class is used to establish connection with  the database. |

### Syntax of getConnection() method

1. 1) **public** **static** Connection getConnection(String url)**throws** SQLException
2. 2) **public** **static** Connection getConnection(String url,String name,String password)
3. **throws** SQLException

### Example to establish connection with the Oracle database

1. Connection con=DriverManager.getConnection(
2. "jdbc:oracle:thin:@localhost:1521:xe","system","password");

### 3) Create the Statement object

|  |
| --- |
| The createStatement() method of Connection interface is used to create statement. The  object of statement is responsible to execute queries with the database. |

### Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

### Example to create the statement object

1. Statement stmt=con.createStatement();

### 4) Execute the query

|  |
| --- |
| The executeQuery() method of Statement interface is used to execute queries to the database.  This method returns the object of ResultSet that can be used to get all the records of a table. |

### Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

### Example to execute query

1. ResultSet rs=stmt.executeQuery("select \* from emp");
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

### 5) Close the connection object

|  |
| --- |
| By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection. |

### Syntax of close() method

1. **public** **void** close()**throws** SQLException

### Example to close connection

1. con.close();

**Example program:**

1. CREATE TABLE  "REGISTERUSER"
2. (    "NAME" VARCHAR2(4000),
3. "PASS" VARCHAR2(4000),
4. "EMAIL" VARCHAR2(4000),
5. "COUNTRY" VARCHAR2(4000)
6. )

**register.html**

1. <html>
2. <body>
3. <form action="servlet/Register" method="post">
5. Name:<input type="text" name="userName"/><br/><br/>
6. Password:<input type="password" name="userPass"/><br/><br/>
7. Email Id:<input type="text" name="userEmail"/><br/><br/>
8. Country:
9. <select name="userCountry">
10. <option>India</option>
11. <option>Pakistan</option>
12. <option>other</option>
13. </select>
15. <br/><br/>
16. <input type="submit" value="register"/>
18. </form>
19. </body>
20. </html>

**Register.java**

1. **import** java.io.\*;
2. **import** java.sql.\*;
3. **import** javax.servlet.ServletException;
4. **import** javax.servlet.http.\*;
6. **public** **class** Register **extends** HttpServlet {
7. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)
8. **throws** ServletException, IOException {
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. String n=request.getParameter("userName");
14. String p=request.getParameter("userPass");
15. String e=request.getParameter("userEmail");
16. String c=request.getParameter("userCountry");
18. **try**{
19. Class.forName("oracle.jdbc.driver.OracleDriver");
20. Connection con=DriverManager.getConnection(
21. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
23. PreparedStatement ps=con.prepareStatement(
24. "insert into registeruser values(?,?,?,?)");
26. ps.setString(1,n);
27. ps.setString(2,p);
28. ps.setString(3,e);
29. ps.setString(4,c);
31. **int** i=ps.executeUpdate();
32. **if**(i>0)
33. out.print("You are successfully registered...");

36. }**catch** (Exception e2) {System.out.println(e2);}
38. out.close();
39. }
41. }

**web.xml file**

The is the configuration file, providing information about the servlet.

1. <web-app>
3. <servlet>
4. <servlet-name>Register</servlet-name>
5. <servlet-**class**>Register</servlet-**class**>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>Register</servlet-name>
10. <url-pattern>/servlet/Register</url-pattern>
11. </servlet-mapping>
13. <welcome-file-list>
14. <welcome-file>register.html</welcome-file>
15. </welcome-file-list>
17. </web-app>