# Servlet

**Azam Amir Reza (**[**azam20104u@gmail.com**](mailto:azam20104u@gmail.com)**) +917278039915**

**Servlet** technology is used to create web application (resides at server side and generates dynamic web page).

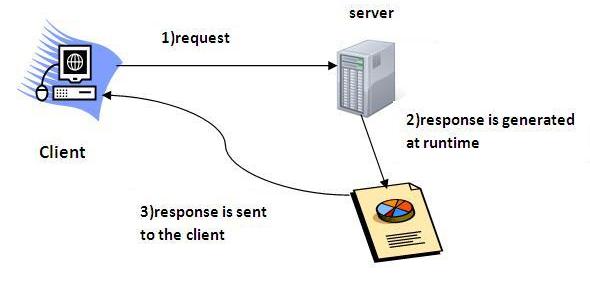
**Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

## What is a Servlet?

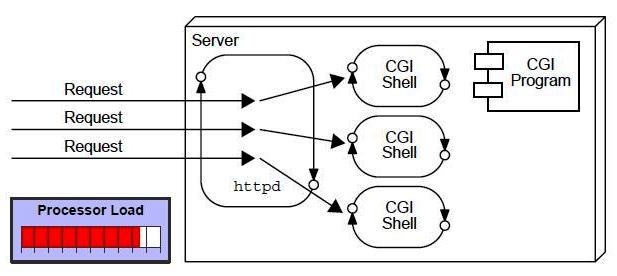
Servlet can be described in many ways, depending on the context.

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



### CGI(Commmon Gateway Interface)

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

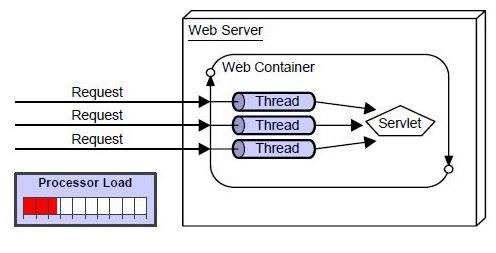


### Disadvantages of CGI

There are many problems in CGI technology:

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

### Advantage of Servlet



There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

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

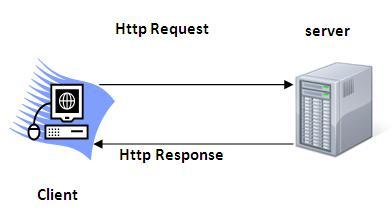
# Servlet Terminology

1. [Basics of Servlet](http://www.javatpoint.com/servlet-terminology)
2. [HTTP](http://www.javatpoint.com/servlet-terminology#http)
3. [Http Request Methods](http://www.javatpoint.com/servlet-terminology#httpreqmethods)
4. [Difference between Get and Post](http://www.javatpoint.com/servlet-terminology#diffgetandpost)
5. [Anatomy of Get Request](http://www.javatpoint.com/servlet-terminology#get)
6. [Anatomy of Post Request](http://www.javatpoint.com/servlet-terminology#post)
7. [Content Type](http://www.javatpoint.com/servlet-terminology#contenttype)

There are some key points that must be known by the servlet programmer like server, container, get request, post request etc

### HTTP (Hyper Text Transfer Protocol)

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



### Http Request Methods

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

The http request methods are:

* GET
* POST
* HEAD
* PUT
* DELETE
* OPTIONS
* TRACE

### What is the difference between Get and Post?

|  |  |
| --- | --- |
| **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. |

### Container

It provides runtime environment for JavaEE (j2ee) applications.

It performs many operations that are given below:

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

### Server

It is a running program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

### Web Server

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

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

### Application Server

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

Example of Application Servers are:

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

### Content Type

Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser.

There are many content types:

* text/html
* text/plain
* application/msword
* application/vnd.ms-excel
* application/jar
* application/pdf
* application/octet-stream
* application/x-zip
* images/jpeg
* video/quicktime etc.

# Servlet API

1. [Servlet API](http://www.javatpoint.com/servlet-api)
2. [Interfaces in javax.servlet package](http://www.javatpoint.com/servlet-api#servletapi1)
3. [Classes in javax.servlet package](http://www.javatpoint.com/servlet-api#servletapi2)
4. [Interfaces in javax.servlet.http package](http://www.javatpoint.com/servlet-api#servletapi3)
5. [Classes in javax.servlet.http package](http://www.javatpoint.com/servlet-api#servletapi4)

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

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

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

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

### Interfaces in javax.servlet package

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

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

### Classes in javax.servlet package

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

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

### Interfaces in javax.servlet.http package

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

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

### Classes in javax.servlet.http package

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

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

**What is Servlet vs GenericServlet vs HttpServlet?**

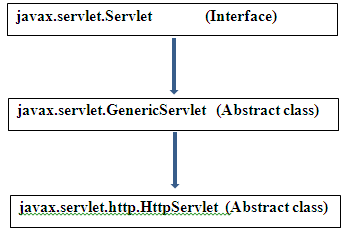
Servlets are **platform-independent** server-side components, being written in Java. Before going for differences, first let us see how the three **Servlet**, **GenericServlet**, **HttpServlet** are related, their signatures and also at the end similarities.

See how to write a Servlet.

**public class Validation extends HttpServlet**

To write a servlet, everyone goes to extend the abstract class **HttpServlet**, like **Frame** is required to extend to create a frame.

**Following figure shows the hierarchy of Servlet vs GenericServlet vs HttpServlet and to know from where HttpServlet comes.**

[](http://way2java.com/wp-content/uploads/2014/02/image1.png)  
Figure on Servlet vs GenericServlet vs HttpServlet

Observe the hierarchy and understand the relationship between the three (involved in multilevel inheritance). With the observation, a conclusion can be arrived, to write a Servlet three ways exist.

**a)** by implementing Servlet (it is interface)  
**b)** by extending GenericServlet (it is abstract class)  
**c)** by extending HttpServlet (it is abstract class)

The minus point of the first way is, all the 5 abstract methods of the interface **Servlet** should be overridden eventhough Programmer is not interested in all (like the interface**WindowListener** to close the frame). A smart approach is inheriting **GenericServlet** (like using**WindowAdapter**) and overriding its only one abstract method **service()**. It is enough to the programmer to override only this method. It is a **callback** method (called implicitly). Still better way is extending HttpServlet and need not to override any methods as HttpServlet contains no abstract methods. Eventhough the HttpServlet does not contain any abstract methods, it is declared as abstract class by the Designers to not to allow the Programmer to create an object directly because a Servlet object is created by the system (here system is Servlet Container).

**1. Servlet interface**

It is the super interface for the remaining two – GenericServlet and HttpServlet. It contains 5 abstract methods and all inherited by GenericServlet and HttpServlet. Programmers implement Servlet interface who would like to develop their own container.

**2. GenericServlet**

It is the immediate subclass of **Servlet interface**. In this class, only one abstract method**service()** exist. Other 4 abstract methods of Servlet interface are given implementation (given body). Anyone who extends this **GenericServlet** should override **service()** method. It was used by the Programmers when the Web was not standardized to **HTTP** protocol. It is protocol independent; it can be used with any protocol, say, SMTP, FTP, CGI including HTTP etc.

Signature:

**public abstract class GenericServlet extends java.lang.Object implements Servlet, ServletConfig, java.io.Serializable**  
**3. HttpServlet**

When HTTP protocol was developed by W3C people to suit more Web requirements, the Servlet designers introduced **HttpServlet** to suit more for HTTP protocol. HttpServlet is protocol dependent and used specific to HTTP protocol only.

The immediate super class of **HttpServlet** is **GenericServlet**. HttpServlet overrides the**service()** method of GenericServlet. HttpServlet is abstract class but without any abstract methods.

With HttpServlet extension, **service()** method can be replaced by **doGet()** or **doPost()** with the same parameters of service() method.

Signature:

**public abstract class HttpServlet extends GenericServlet implements java.io.Serializable**  
Being subclass of GenericServlet, the HttpServlet inherits all the properties (methods) of GenericServlet. So, if you extend HttpServlet, you can get the functionality of both.

Let us tabulate the differences for easy understanding and remembering.

|  |  |
| --- | --- |
| GENERICSERVLET | HTTPSERVLET |
| Can be used with any protocol (means, can handle any protocol). Protocol independent. | Should be used with HTTP protocol only (can handle HTTP specific protocols) . Protocol dependent. |
| All methods are concrete except service() method. service() method is abstract method. | All methods are concrete (non-abstract). service() is non-abstract method. |
| service() should be overridden being abstract in super interface. | service() method need not be overridden. |
| It is a must to use service() method as it is a callback method. | Being service() is non-abstract, it can be replaced by doGet() or doPost() methods. |
| Extends Object and implements interfaces Servlet, ServletConfig and Serializable. | Extends GenericServlet and implements interface Serializable |
| Direct subclass of Servet interface. | Direct subclass of GenericServlet. |
| Defined javax.servlet package. | Defined javax.servlet.http package. |
| All the classes and interfaces belonging to javax.servlet package are protocol independent. | All the classes and interfaces present in javax.servlet.http package are protocol dependent (specific to HTTP). |
| Not used now-a-days. | Used always. |

# Servlet Interface

1. [Servlet Interface](http://www.javatpoint.com/Servlet-interface)
2. [Methods of Servlet interface](http://www.javatpoint.com/Servlet-interface#servletmethods)

**Servlet interface** provides common behaviour to all the servlets.

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

### Methods of Servlet interface

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

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

### Servlet Example by implementing Servlet interface

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

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

*File: First.java*

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
4. **public** **class** First **implements** Servlet{
5. ServletConfig config=**null**;
7. **public** **void** init(ServletConfig config){
8. **this**.config=config;
9. System.out.println("servlet is initialized");
10. }
12. **public** **void** service(ServletRequest req,ServletResponse res)
13. **throws** IOException,ServletException{
15. res.setContentType("text/html");
17. PrintWriter out=res.getWriter();
18. out.print("<html><body>");
19. out.print("<b>hello simple servlet</b>");
20. out.print("</body></html>");
21. }
22. **public** **void** destroy(){System.out.println("servlet is destroyed");}
23. **public** ServletConfig getServletConfig(){**return** config;}
24. **public** String getServletInfo(){**return** "copyright 2007-1010";}
25. }

# GenericServlet class

1. [GenericServlet class](http://www.javatpoint.com/GenericServlet-class)
2. [Methods of GenericServlet class](http://www.javatpoint.com/GenericServlet-class#genericmethods)
3. [Example of GenericServlet class](http://www.javatpoint.com/GenericServlet-class)

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

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

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

### Methods of GenericServlet class

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

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

### Servlet Example by inheriting the GenericServlet class

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

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

*File: First.java*

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

# HttpServlet class

1. [HttpServlet class](http://www.javatpoint.com/HttpServlet-class)
2. [Methods of HttpServlet class](http://www.javatpoint.com/HttpServlet-class#httpservletmethods)

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

### Methods of HttpServlet class

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

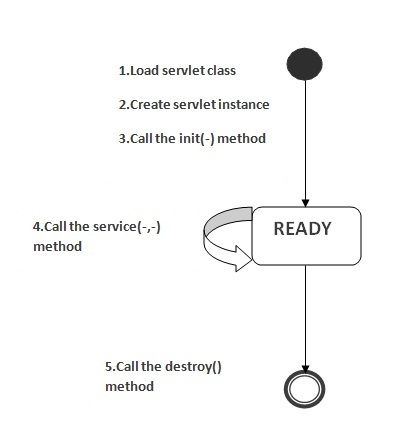
1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service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

# Life Cycle of a Servlet (Servlet Life Cycle)

1. [Life Cycle of a Servlet](http://www.javatpoint.com/life-cycle-of-a-servlet)
   1. [Servlet class is loaded](http://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle1)
   2. [Servlet instance is created](http://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle2)
   3. [init method is invoked](http://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle3)
   4. [service method is invoked](http://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle4)
   5. [destroy method is invoked](http://www.javatpoint.com/life-cycle-of-a-servlet#servletlifecycle5)

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

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



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

### 1) Servlet class is loaded

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

### 2) Servlet instance is created

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

### 3) init method is invoked

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

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

### 4) service method is invoked

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

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

### 5) destroy method is invoked

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

1. **public** **void** destroy()

# Steps to create a servlet example

1. [Steps to create the servlet using Tomcat server](http://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server)
   1. [Create a directory structure](http://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep1)
   2. [Create a Servlet](http://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep2)
   3. [Compile the Servlet](http://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep3)
   4. [Create a deployment descriptor](http://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep4)
   5. [Start the server and deploy the application](http://www.javatpoint.com/steps-to-create-a-servlet-using-tomcat-server#servletstep5)

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

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

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

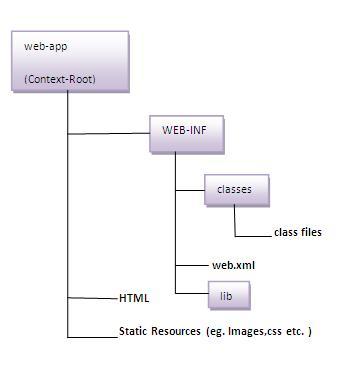
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

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

### 1)Create a directory structures

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

### 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. |

# ServletRequest Interface

1. [ServletRequest Interface](http://www.javatpoint.com/servletrequest)
2. [Methods of ServletRequest interface](http://www.javatpoint.com/servletrequest#methods)
3. [Example of ServletRequest interface](http://www.javatpoint.com/servletrequest#example)
4. [Displaying all the header information](http://www.javatpoint.com/displaying-all-the-header-information-in-the-servlet)

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

### Methods of ServletRequest interface

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

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

### Example of ServletRequest to display the name of the user

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

**index.html**

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

**DemoServ.java**

1. **import** javax.servlet.http.\*;
2. **import** javax.servlet.\*;
3. **import** java.io.\*;
4. **public** **class** DemoServ **extends** HttpServlet{
5. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
6. **throws** ServletException,IOException
7. {
8. res.setContentType("text/html");
9. PrintWriter pw=res.getWriter();
11. String name=req.getParameter("name");//will return value
12. pw.println("Welcome "+name);
14. pw.close();
15. }}

# RequestDispatcher in Servlet

1. [RequestDispatcher Interface](http://www.javatpoint.com/requestdispatcher-in-servlet)
2. [Methods of RequestDispatcher interface](http://www.javatpoint.com/requestdispatcher-in-servlet#rdmethod)
   1. [forward method](http://www.javatpoint.com/requestdispatcher-in-servlet#rdforward)
   2. [include method](http://www.javatpoint.com/requestdispatcher-in-servlet#rdinclude)
3. [How to get the object of RequestDispatcher](http://www.javatpoint.com/requestdispatcher-in-servlet#rdhow)
4. [Example of RequestDispatcher interface](http://www.javatpoint.com/requestdispatcher-in-servlet#rdex)

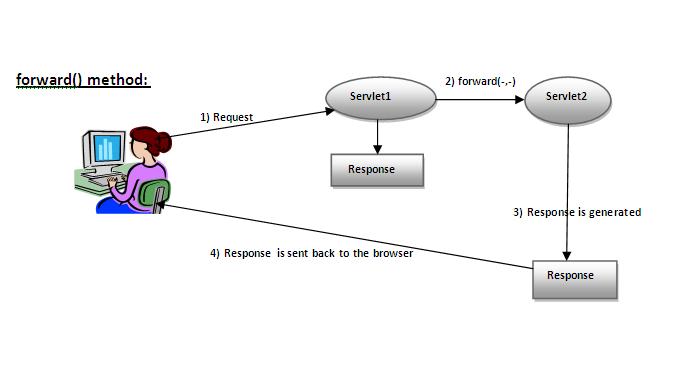
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

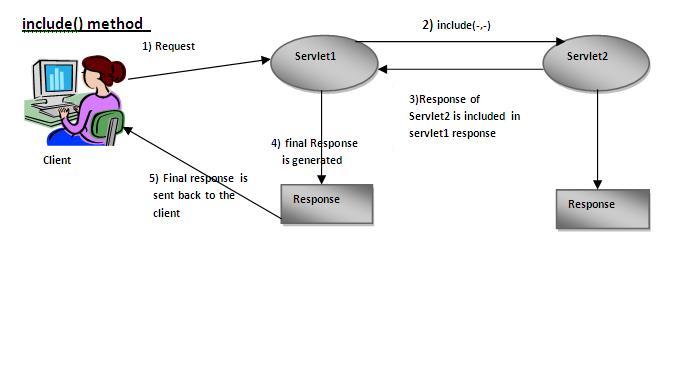
### Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**Includes the content of a resource (servlet, JSP page, or HTML file) in the response.



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



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

### How to get the object of RequestDispatcher

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

#### Syntax of getRequestDispatcher method

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

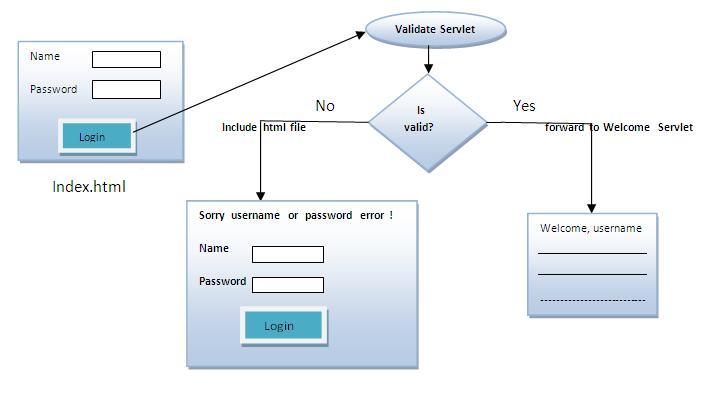
#### Example of using getRequestDispatcher method

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

### Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

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



**index.html**

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

**Login.java**

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

6. **public** **class** Login **extends** HttpServlet {
8. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)
9. **throws** ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. String p=request.getParameter("userPass");
17. **if**(p.equals("servlet"){
18. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
19. rd.forward(request, response);
20. }
21. **else**{
22. out.print("Sorry UserName or Password Error!");
23. RequestDispatcher rd=request.getRequestDispatcher("/index.html");
24. rd.include(request, response);
26. }
27. }
29. }

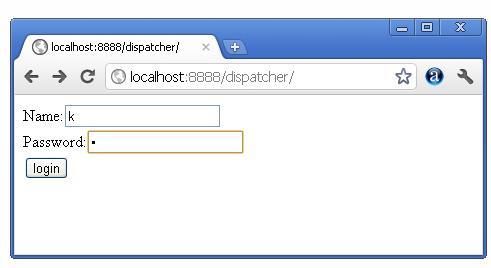
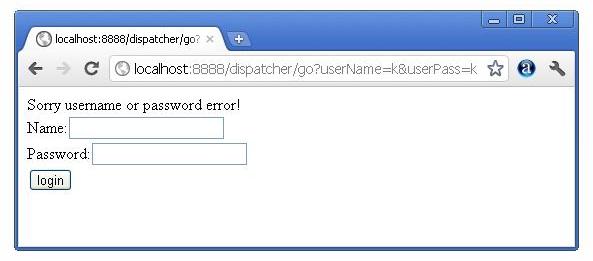
**WelcomeServlet.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** WelcomeServlet **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. out.print("Welcome "+n);
15. }
17. }

**web.xml**

1. <web-app>
2. <servlet>
3. <servlet-name>Login</servlet-name>
4. <servlet-**class**>Login</servlet-**class**>
5. </servlet>
6. <servlet>
7. <servlet-name>WelcomeServlet</servlet-name>
8. <servlet-**class**>WelcomeServlet</servlet-**class**>
9. </servlet>

12. <servlet-mapping>
13. <servlet-name>Login</servlet-name>
14. <url-pattern>/servlet1</url-pattern>
15. </servlet-mapping>
16. <servlet-mapping>
17. <servlet-name>WelcomeServlet</servlet-name>
18. <url-pattern>/servlet2</url-pattern>
19. </servlet-mapping>
21. <welcome-file-list>
22. <welcome-file>index.html</welcome-file>
23. </welcome-file-list>
24. </web-app>

# SendRedirect in servlet

1. [sendRedirect method](http://www.javatpoint.com/sendRedirect()-method)
2. [Syntax of sendRedirect() method](http://www.javatpoint.com/sendRedirect()-method#redirectsyn)
3. [Example of RequestDispatcher interface](http://www.javatpoint.com/sendRedirect()-method#redirectex)

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

## Difference between forward() and sendRedirect() method

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

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

### Syntax of sendRedirect() method

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

### Example of sendRedirect() method

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

### Full example of sendRedirect method in servlet

|  |
| --- |
| In this example, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, that is why we can our request to anywhere. We can send our request within and outside the server. |

*DemoServlet.java*

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

### Creating custom google search using sendRedirect

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

*index.html*

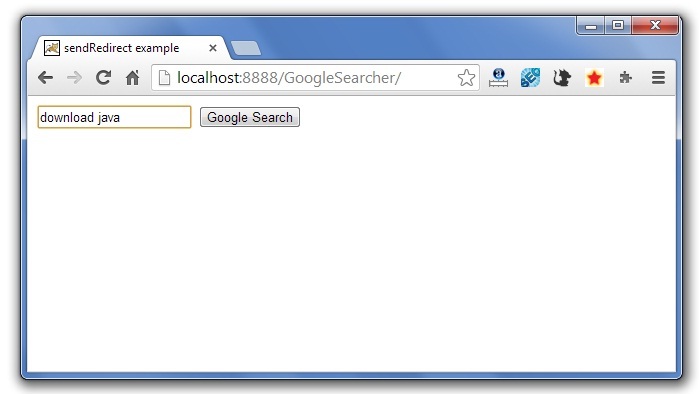
1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="ISO-8859-1"**>**
5. **<title>**sendRedirect example**</title>**
6. **</head>**
7. **<body>**

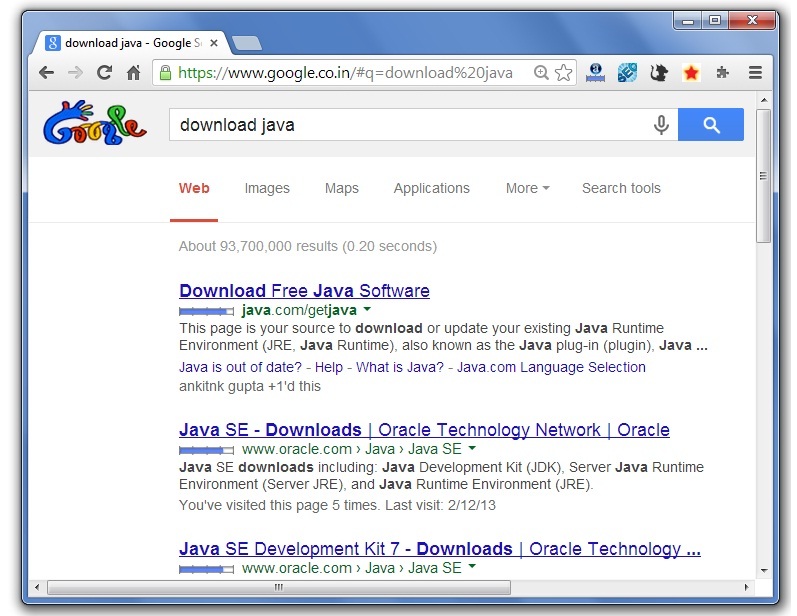
10. **<form** action="MySearcher"**>**
11. **<input** type="text" name="name"**>**
12. **<input** type="submit" value="Google Search"**>**
13. **</form>**
15. **</body>**
16. **</html>**

*MySearcher.java*

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

#### Output





# ServletConfig Interface

1. [ServletConfig Interface](http://www.javatpoint.com/servletconfig)
2. [Methods of ServletConfig interface](http://www.javatpoint.com/servletconfig#configmethod)
3. [How to get the object of ServletConfig](http://www.javatpoint.com/servletconfig#configobject)
4. [Syntax to provide the initialization parameter for a servlet](http://www.javatpoint.com/servletconfig#configsyntax)
5. [Example of ServletConfig to get initialization parameter](http://www.javatpoint.com/servletconfig#configex1)
6. [Example of ServletConfig to get all the initialization parameter](http://www.javatpoint.com/servletconfig#configex2)

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

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

### Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

### Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

### How to get the object of ServletConfig

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

#### Syntax of getServletConfig() method

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

#### Example of getServletConfig() method

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

### Syntax to provide the initialization parameter for a servlet

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

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

### Example of ServletConfig to get initialization parameter

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

**DemoServlet.java**

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

**web.xml**

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

### Example of ServletConfig to get all the initialization parameters

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

**DemoServlet.java**

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
3. **import** java.util.Enumeration;
5. **import** javax.servlet.ServletConfig;
6. **import** javax.servlet.ServletException;
7. **import** javax.servlet.http.HttpServlet;
8. **import** javax.servlet.http.HttpServletRequest;
9. **import** javax.servlet.http.HttpServletResponse;

12. **public** **class** DemoServlet **extends** HttpServlet {
13. **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)
14. **throws** ServletException, IOException {
16. response.setContentType("text/html");
17. PrintWriter out = response.getWriter();
19. ServletConfig config=getServletConfig();
20. Enumeration<String> e=config.getInitParameterNames();
22. String str="";
23. **while**(e.hasMoreElements()){
24. str=e.nextElement();
25. out.print("<br>Name: "+str);
26. out.print(" value: "+config.getInitParameter(str));
27. }
29. out.close();
30. }
32. }

**web.xml**

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

### 3) init method is invoked

|  |
| --- |
| The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. |

# ServletContext Interface

1. [ServletContext Interface](http://www.javatpoint.com/servletcontext)
2. [Usage of ServletContext Interface](http://www.javatpoint.com/servletcontext#contextusage)
3. [Methods of ServletContext interface](http://www.javatpoint.com/servletcontext#contextmethods)
4. [How to get the object of ServletContext](http://www.javatpoint.com/servletcontext#contextobject)
5. [Syntax to provide the initialization parameter in Context scope](http://www.javatpoint.com/servletcontext#contextsyn)
6. [Example of ServletContext to get initialization parameter](http://www.javatpoint.com/servletcontext#contextex1)
7. [Example of ServletContext to get all the initialization parameter](http://www.javatpoint.com/servletcontext#contextex2)

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

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

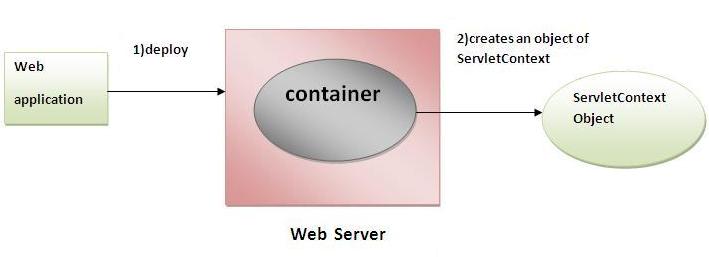
### Advantage of ServletContext

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

### Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.



### Commonly used methods of ServletContext interface

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

### How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

#### Syntax of getServletContext() method

1. **public** ServletContext getServletContext()

#### Example of getServletContext() method

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

### Syntax to provide the initialization parameter in Context scope

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

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

### Example of ServletContext to get the initialization parameter

|  |
| --- |
| In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example: |

**DemoServlet.java**

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

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

**web.xml**

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

### Example of ServletContext to get all the initialization parameters

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

**DemoServlet.java**

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

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

**web.xml**

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-**class**>DemoServlet</servlet-**class**>
6. </servlet>
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
13. <context-param>
14. <param-name>username</param-name>
15. <param-value>system</param-value>
16. </context-param>
18. <context-param>
19. <param-name>password</param-name>
20. <param-value>oracle</param-value>
21. </context-param>
23. <servlet-mapping>
24. <servlet-name>sonoojaiswal</servlet-name>
25. <url-pattern>/context</url-pattern>
26. </servlet-mapping>
28. </web-app>

# Attribute in Servlet

1. [Attribute in Servlet](http://www.javatpoint.com/attribute)
2. [Attribute specific methods](http://www.javatpoint.com/attribute#attributemethod)
3. [Example of ServletContext to set and get attribute](http://www.javatpoint.com/attribute#attributeex)
4. [Difference between ServletConfig and ServletContext](http://www.javatpoint.com/attribute#diffcontext)

An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

1. request scope
2. session scope
3. application scope

The servlet programmer can pass informations from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

### Attribute specific methods of ServletRequest, HttpSession and ServletContext interface

|  |
| --- |
| There are following 4 attribute specific methods. They are as follows:   1. **public void setAttribute(String name,Object object):**sets the given object in the application scope. 2. **public Object getAttribute(String name):**Returns the attribute for the specified name. 3. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects. 4. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context. |

### Example of ServletContext to set and get attribute

|  |
| --- |
| In this example, we are setting the attribute in the application scope and getting that value from another servlet. |

### DemoServlet1.java

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

6. **public** **class** DemoServlet1 **extends** HttpServlet{
7. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
8. {
9. **try**{
11. res.setContentType("text/html");
12. PrintWriter out=res.getWriter();
14. ServletContext context=getServletContext();
15. context.setAttribute("company","IBM");
17. out.println("Welcome to first servlet");
18. out.println("<a href='servlet2'>visit</a>");
19. out.close();
21. }**catch**(Exception e){out.println(e);}
23. }}

### DemoServlet2.java

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

6. **public** **class** DemoServlet2 **extends** HttpServlet{
7. **public** **void** doGet(HttpServletRequest req,HttpServletResponse res)
8. {
9. **try**{
11. res.setContentType("text/html");
12. PrintWriter out=res.getWriter();
14. ServletContext context=getServletContext();
15. String n=(String)context.getAttribute("company");
17. out.println("Welcome to "+n);
18. out.close();
20. }**catch**(Exception e){out.println(e);}
22. }}

### web.xml

1. <web-app>
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-**class**>DemoServlet1</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**>DemoServlet2</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>

### Difference between ServletConfig and ServletContext

|  |
| --- |
| The servletconfig object refers to the single servlet whereas servletcontext object refers to the whole web application. |

# Session Tracking in Servlets

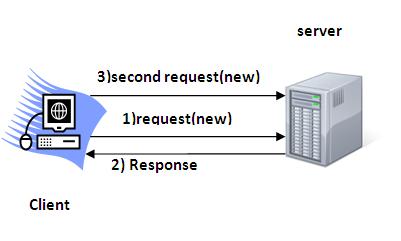
1. [Session Tracking](http://www.javatpoint.com/session-tracking-in-servlets#session1)
2. [Session Tracking Techniques](http://www.javatpoint.com/session-tracking-in-servlets#session1tech)

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

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

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

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



### Why use Session Tracking?

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

### Session Tracking Techniques

There are four techniques used in Session tracking:

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

# Cookies in Servlet

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

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

### How Cookie works

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



### Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

### Non-persistent cookie

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

### Persistent cookie

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

### Advantage of Cookies

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

### Disadvantage of Cookies

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

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

### Cookie class

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

### Constructor of Cookie class

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

### Useful Methods of Cookie class

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

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

### Other methods required for using Cookies

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

### How to create Cookie?

Let's see the simple code to create cookie.

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

### How to delete Cookie?

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

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

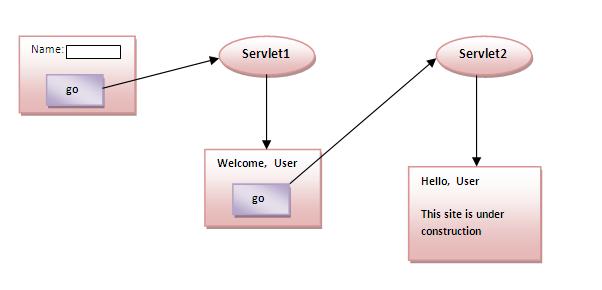
### How to get Cookies?

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

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

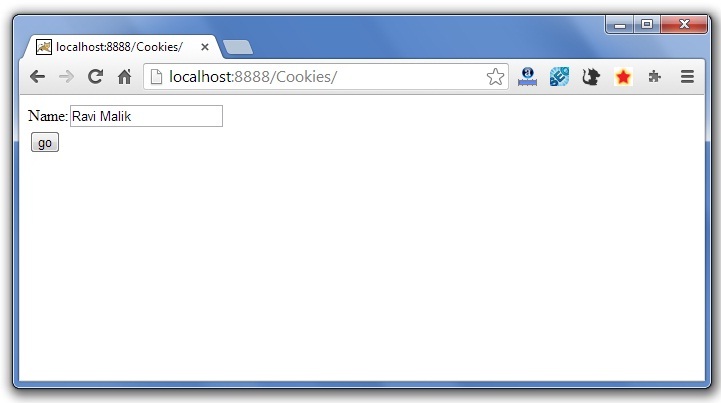
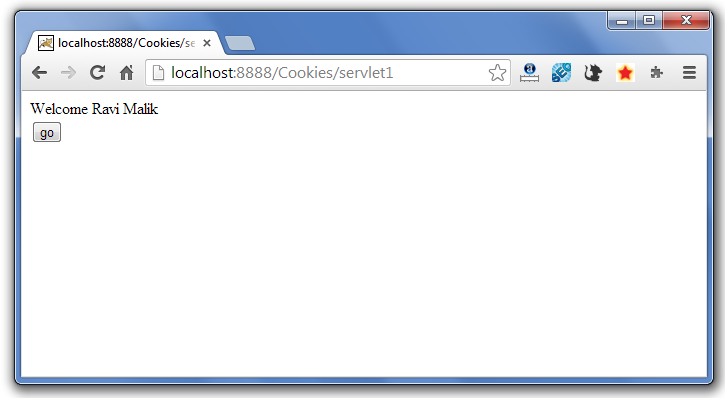
1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
5. **public** **class** SecondServlet **extends** HttpServlet {
7. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){
8. **try**{
10. response.setContentType("text/html");
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. }

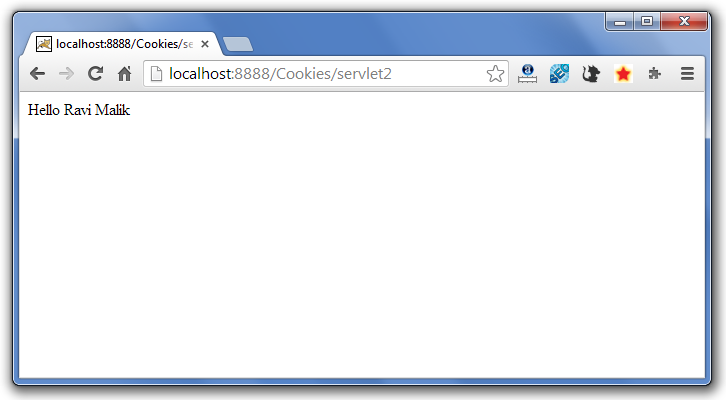
22. }

### 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



# Servlet Login and Logout Example using Cookies

A **cookie** is a kind of information that is stored at client side.

In the previous page, we learned a lot about cookie e.g. how to create cookie, how to delete cookie, how to get cookie etc.

Here, we are going to create a login and logout example using servlet cookies.

In this example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

*File: index.html*

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="ISO-8859-1"**>**
5. **<title>**Servlet Login Example**</title>**
6. **</head>**
7. **<body>**
9. **<h1>**Welcome to Login App by Cookie**</h1>**
10. **<a** href="login.html"**>**Login**</a>**|
11. **<a** href="LogoutServlet"**>**Logout**</a>**|
12. **<a** href="ProfileServlet"**>**Profile**</a>**
14. **</body>**
15. **</html>**

*File: link.html*

1. **<a** href="login.html"**>**Login**</a>** |
2. **<a** href="LogoutServlet"**>**Logout**</a>** |
3. **<a** href="ProfileServlet"**>**Profile**</a>**
4. **<hr>**

*File: login.html*

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

*File: LoginServlet.java*

1. **package** com.javatpoint;
3. **import** java.io.IOException;
4. **import** java.io.PrintWriter;
5. **import** javax.servlet.ServletException;
6. **import** javax.servlet.http.Cookie;
7. **import** javax.servlet.http.HttpServlet;
8. **import** javax.servlet.http.HttpServletRequest;
9. **import** javax.servlet.http.HttpServletResponse;
10. **public** **class** LoginServlet **extends** HttpServlet {
11. **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)
12. **throws** ServletException, IOException {
13. response.setContentType("text/html");
14. PrintWriter out=response.getWriter();
16. request.getRequestDispatcher("link.html").include(request, response);
18. String name=request.getParameter("name");
19. String password=request.getParameter("password");
21. **if**(password.equals("admin123")){
22. out.print("You are successfully logged in!");
23. out.print("<br>Welcome, "+name);
25. Cookie ck=**new** Cookie("name",name);
26. response.addCookie(ck);
27. }**else**{
28. out.print("sorry, username or password error!");
29. request.getRequestDispatcher("login.html").include(request, response);
30. }
32. out.close();
33. }
35. }

*File: LogoutServlet.java*

1. **package** com.javatpoint;
3. **import** java.io.IOException;
4. **import** java.io.PrintWriter;
5. **import** javax.servlet.ServletException;
6. **import** javax.servlet.http.Cookie;
7. **import** javax.servlet.http.HttpServlet;
8. **import** javax.servlet.http.HttpServletRequest;
9. **import** javax.servlet.http.HttpServletResponse;
10. **public** **class** LogoutServlet **extends** HttpServlet {
11. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)
12. **throws** ServletException, IOException {
13. response.setContentType("text/html");
14. PrintWriter out=response.getWriter();

17. request.getRequestDispatcher("link.html").include(request, response);
19. Cookie ck=**new** Cookie("name","");
20. ck.setMaxAge(0);
21. response.addCookie(ck);
23. out.print("you are successfully logged out!");
24. }
25. }

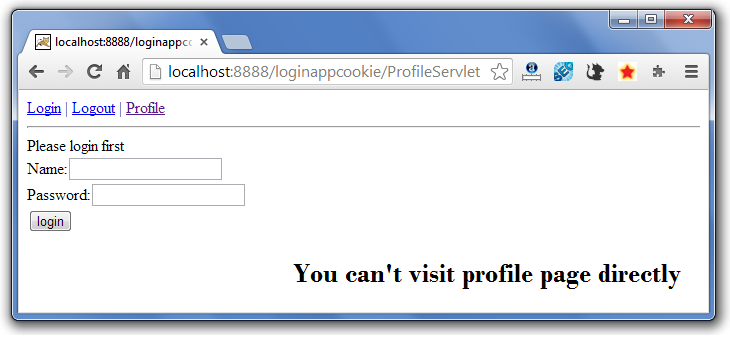
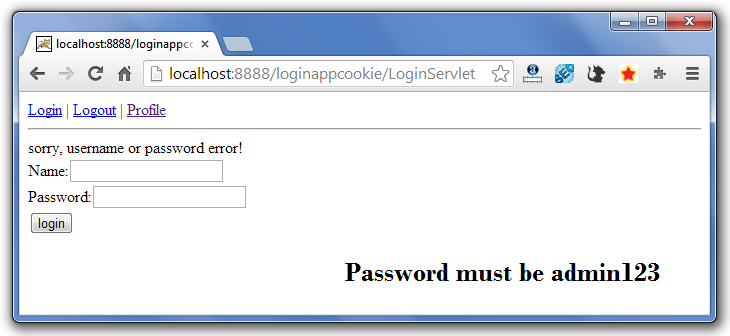
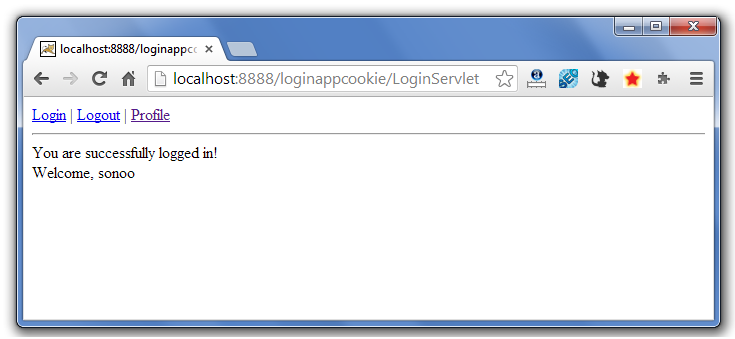
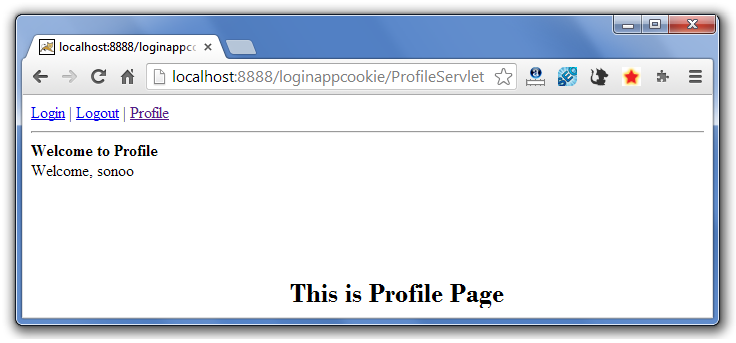
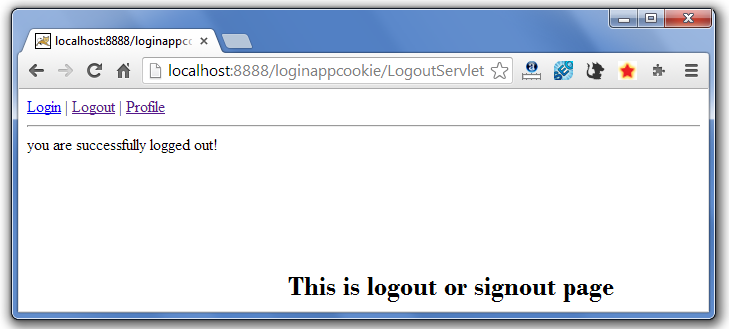
*File: ProfileServlet.java*

1. **package** com.javatpoint;
3. **import** java.io.IOException;
4. **import** java.io.PrintWriter;
5. **import** javax.servlet.ServletException;
6. **import** javax.servlet.http.Cookie;
7. **import** javax.servlet.http.HttpServlet;
8. **import** javax.servlet.http.HttpServletRequest;
9. **import** javax.servlet.http.HttpServletResponse;
10. **public** **class** ProfileServlet **extends** HttpServlet {
11. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)
12. **throws** ServletException, IOException {
13. response.setContentType("text/html");
14. PrintWriter out=response.getWriter();
16. request.getRequestDispatcher("link.html").include(request, response);
18. Cookie ck[]=request.getCookies();
19. **if**(ck!=**null**){
20. String name=ck[0].getValue();
21. **if**(!name.equals("")||name!=**null**){
22. out.print("<b>Welcome to Profile</b>");
23. out.print("<br>Welcome, "+name);
24. }
25. }**else**{
26. out.print("Please login first");
27. request.getRequestDispatcher("login.html").include(request, response);
28. }
29. out.close();
30. }
31. }

*File: web.xml*

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
4. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5"**>**
6. **<servlet>**
7. **<description></description>**
8. **<display-name>**LoginServlet**</display-name>**
9. **<servlet-name>**LoginServlet**</servlet-name>**
10. **<servlet-class>**com.javatpoint.LoginServlet**</servlet-class>**
11. **</servlet>**
12. **<servlet-mapping>**
13. **<servlet-name>**LoginServlet**</servlet-name>**
14. **<url-pattern>**/LoginServlet**</url-pattern>**
15. **</servlet-mapping>**
16. **<servlet>**
17. **<description></description>**
18. **<display-name>**ProfileServlet**</display-name>**
19. **<servlet-name>**ProfileServlet**</servlet-name>**
20. **<servlet-class>**com.javatpoint.ProfileServlet**</servlet-class>**
21. **</servlet>**
22. **<servlet-mapping>**
23. **<servlet-name>**ProfileServlet**</servlet-name>**
24. **<url-pattern>**/ProfileServlet**</url-pattern>**
25. **</servlet-mapping>**
26. **<servlet>**
27. **<description></description>**
28. **<display-name>**LogoutServlet**</display-name>**
29. **<servlet-name>**LogoutServlet**</servlet-name>**
30. **<servlet-class>**com.javatpoint.LogoutServlet**</servlet-class>**
31. **</servlet>**
32. **<servlet-mapping>**
33. **<servlet-name>**LogoutServlet**</servlet-name>**
34. **<url-pattern>**/LogoutServlet**</url-pattern>**
35. **</servlet-mapping>**
36. **</web-app>**

#### Output

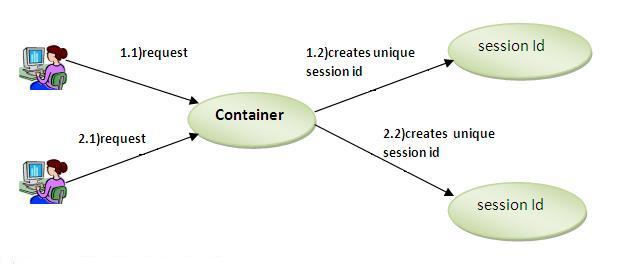
If again you click on the profile link, you need to login first.

# 4) HttpSession interface

1. [HttpSession interface](http://www.javatpoint.com/http-session-in-session-tracking)
2. [How to get the HttpSession object](http://www.javatpoint.com/http-session-in-session-tracking#httpsessionhow)
3. [Commonly used methods of HttpSession interface](http://www.javatpoint.com/http-session-in-session-tracking#httpsessionmethod)
4. [Example of using HttpSession](http://www.javatpoint.com/http-session-in-session-tracking#httpsessionex)

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

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



### How to get the HttpSession object ?

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

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

### Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

### Example of using HttpSession

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

### index.html

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

### FirstServlet.java

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

6. **public** **class** FirstServlet **extends** HttpServlet {
8. **public** **void** doGet(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. HttpSession session=request.getSession();
18. session.setAttribute("uname",n);
20. out.print("<a href='servlet2'>visit</a>");
22. out.close();
24. }**catch**(Exception e){System.out.println(e);}
25. }
27. }

### SecondServlet.java

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

23. }

### 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>

# Servlet HttpSession Login and Logout Example

We can bind the objects on HttpSession instance and get the objects by using setAttribute and getAttribute methods.

In the previous page, we have learnt about what is HttpSession, How to store and get data from session object etc.

Here, we are going to create a real world login and logout application without using database code. We are assuming that password is admin123.

Visit here for login and logout application using cookies only [servlet login and logout example using cookies](http://www.javatpoint.com/servlet-login-and-logout-example-using-cookies)

In this example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

*File: index.html*

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="ISO-8859-1"**>**
5. **<title>**Servlet Login Example**</title>**
6. **</head>**
7. **<body>**
9. **<h1>**Login App using HttpSession**</h1>**
10. **<a** href="login.html"**>**Login**</a>**|
11. **<a** href="LogoutServlet"**>**Logout**</a>**|
12. **<a** href="ProfileServlet"**>**Profile**</a>**
14. **</body>**
15. **</html>**

*File: link.html*

1. **<a** href="login.html"**>**Login**</a>** |
2. **<a** href="LogoutServlet"**>**Logout**</a>** |
3. **<a** href="ProfileServlet"**>**Profile**</a>**
4. **<hr>**

*File: login.html*

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

*File: LoginServlet.java*

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
4. **import** javax.servlet.ServletException;
5. **import** javax.servlet.http.HttpServlet;
6. **import** javax.servlet.http.HttpServletRequest;
7. **import** javax.servlet.http.HttpServletResponse;
8. **import** javax.servlet.http.HttpSession;
9. **public** **class** LoginServlet **extends** HttpServlet {
10. **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)
11. **throws** ServletException, IOException {
12. response.setContentType("text/html");
13. PrintWriter out=response.getWriter();
14. request.getRequestDispatcher("link.html").include(request, response);
16. String name=request.getParameter("name");
17. String password=request.getParameter("password");
19. **if**(password.equals("admin123")){
20. out.print("Welcome, "+name);
21. HttpSession session=request.getSession();
22. session.setAttribute("name",name);
23. }
24. **else**{
25. out.print("Sorry, username or password error!");
26. request.getRequestDispatcher("login.html").include(request, response);
27. }
28. out.close();
29. }
30. }

*File: LogoutServlet.java*

1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
4. **import** javax.servlet.ServletException;
5. **import** javax.servlet.http.HttpServlet;
6. **import** javax.servlet.http.HttpServletRequest;
7. **import** javax.servlet.http.HttpServletResponse;
8. **import** javax.servlet.http.HttpSession;
9. **public** **class** LogoutServlet **extends** HttpServlet {
10. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)
11. **throws** ServletException, IOException {
12. response.setContentType("text/html");
13. PrintWriter out=response.getWriter();
15. request.getRequestDispatcher("link.html").include(request, response);
17. HttpSession session=request.getSession();
18. session.invalidate();
20. out.print("You are successfully logged out!");
22. out.close();
23. }
24. }

*File: ProfileServlet.java*

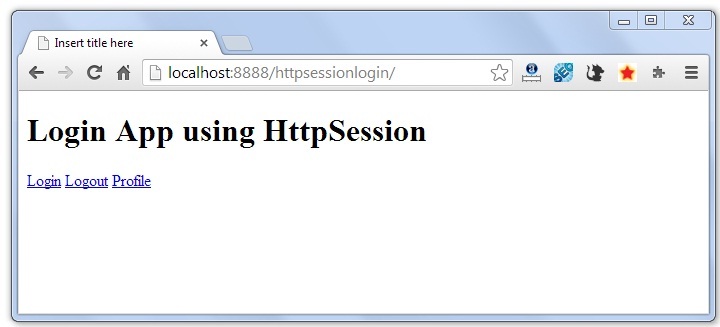
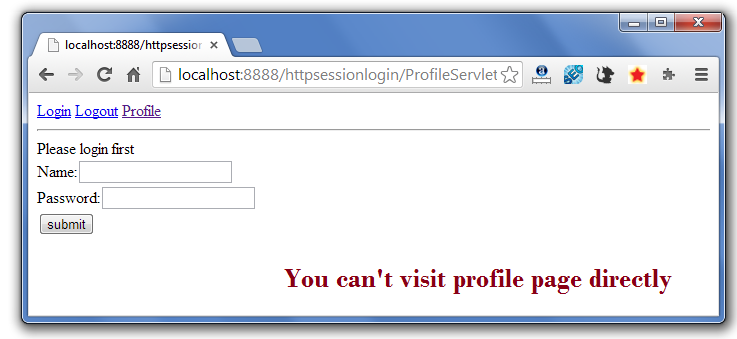
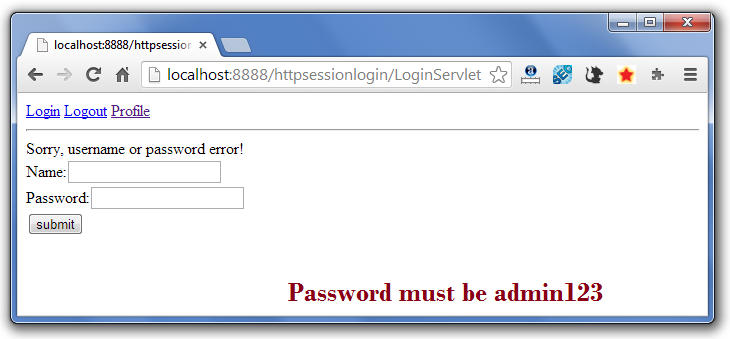
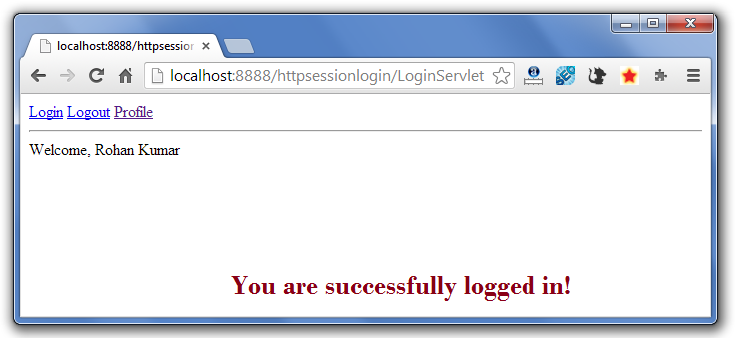
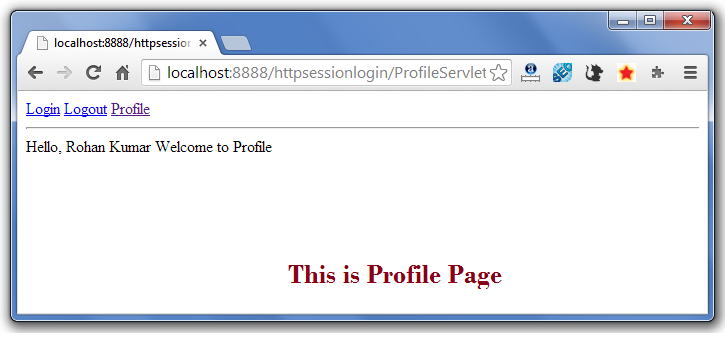
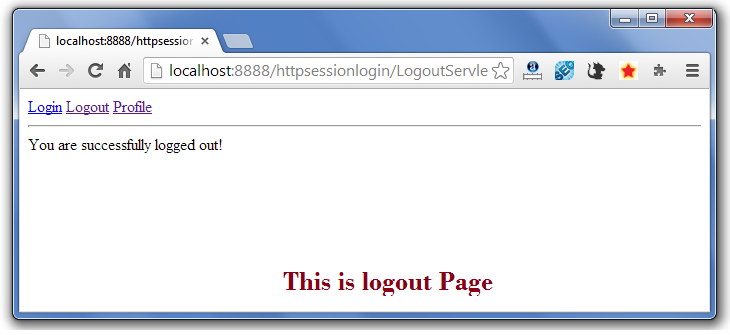
1. **import** java.io.IOException;
2. **import** java.io.PrintWriter;
3. **import** javax.servlet.ServletException;
4. **import** javax.servlet.http.HttpServlet;
5. **import** javax.servlet.http.HttpServletRequest;
6. **import** javax.servlet.http.HttpServletResponse;
7. **import** javax.servlet.http.HttpSession;
8. **public** **class** ProfileServlet **extends** HttpServlet {
9. **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)
10. **throws** ServletException, IOException {
11. response.setContentType("text/html");
12. PrintWriter out=response.getWriter();
13. request.getRequestDispatcher("link.html").include(request, response);
15. HttpSession session=request.getSession(**false**);
16. **if**(session!=**null**){
17. String name=(String)session.getAttribute("name");
19. out.print("Hello, "+name+" Welcome to Profile");
20. }
21. **else**{
22. out.print("Please login first");
23. request.getRequestDispatcher("login.html").include(request, response);
24. }
25. out.close();
26. }
27. }

*File: web.xml*

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
4. http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5"**>**
6. **<servlet>**
7. **<description></description>**
8. **<display-name>**LoginServlet**</display-name>**
9. **<servlet-name>**LoginServlet**</servlet-name>**
10. **<servlet-class>**LoginServlet**</servlet-class>**
11. **</servlet>**
12. **<servlet-mapping>**
13. **<servlet-name>**LoginServlet**</servlet-name>**
14. **<url-pattern>**/LoginServlet**</url-pattern>**
15. **</servlet-mapping>**
16. **<servlet>**
17. **<description></description>**
18. **<display-name>**ProfileServlet**</display-name>**
19. **<servlet-name>**ProfileServlet**</servlet-name>**
20. **<servlet-class>**ProfileServlet**</servlet-class>**
21. **</servlet>**
22. **<servlet-mapping>**
23. **<servlet-name>**ProfileServlet**</servlet-name>**
24. **<url-pattern>**/ProfileServlet**</url-pattern>**
25. **</servlet-mapping>**
26. **<servlet>**
27. **<description></description>**
28. **<display-name>**LogoutServlet**</display-name>**
29. **<servlet-name>**LogoutServlet**</servlet-name>**
30. **<servlet-class>**LogoutServlet**</servlet-class>**
31. **</servlet>**
32. **<servlet-mapping>**
33. **<servlet-name>**LogoutServlet**</servlet-name>**
34. **<url-pattern>**/LogoutServlet**</url-pattern>**
35. **</servlet-mapping>**
36. **</web-app>**

[download this example (developed using Eclipse IDE)](http://www.javatpoint.com/src/servlet/eclipse/httpsessionlogin.zip)

#### Output

If again you click on the profile link, you need to login first.