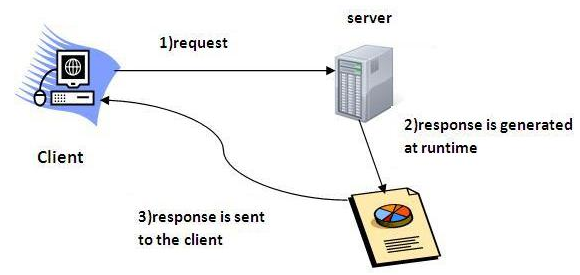
**Servlets**

**Servlet** technology is used to create web application (resides at server side and generates dynamic web page). It is robust and scalable because of java language.

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

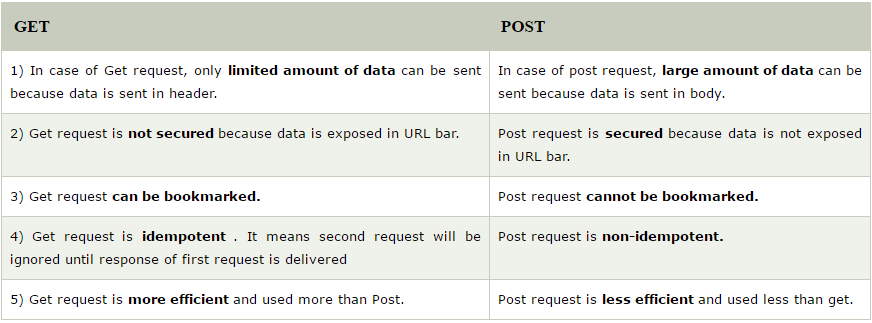
* Servlet is a technology i.e. used to create web application.
* Servlet is an API that provides many interfaces and classes including documentations.
* Servlet is an interface that must be implemented for creating any servlet.
* Servlet is a class that extend the capabilities of the servers and respond 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.



**Web Application:**

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

**GET vs POST:**



**Web Server vs Application Server:**

1. Web Server is designed to serve HTTP Content. Application Server can also serve HTTP Content but is not limited to just HTTP. It can be provided other protocol support such as RMI/RPC.
2. Web Server is mostly designed to serve static content, though most Web Servers have plugins to support scripting languages like Perl, PHP, ASP, JSP etc. through which these servers can generate dynamic HTTP content.
3. Most of the application servers have Web Server as integral part of them, that means App Server can do whatever Web Server is capable of. Additionally Application Server have components and features to support Application level services such as Connection Pooling, Object Pooling, Transaction Support, Messaging services etc.
4. As web servers are well suited for static content and app servers for dynamic content, most of the production environments have web server acting as reverse proxy to app server. That means while service a page request, static contents such as images/Static html is served by web server that interprets the request. Using some kind of filtering technique (mostly extension of requested resource) web server identifies dynamic content request and transparently forwards to app server
5. Example of such configuration is Apache HTTP Server and BEA WebLogic Server. Apache HTTP Server is Web Server and BEA WebLogic is Application Server.

**Servlet API:**

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

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

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

**javax.servlet package:**

|  |  |
| --- | --- |
| **Interfaces** | **Classes** |
| Servlet | GenericServlet |
| ServletRequest | ServletInputStream |
| ServletResponse | ServletOutputStream |
| RequestDispatcher | ServletRequestWrapper |
| ServletConfig | ServletResponseWrapper |
| ServletContext | ServletRequestEvent |
| SingleThreadModel | ServletContextEvent |
| Filter | ServletRequestAttributeEvent |
| FilterConfig | ServletContextAttributeEvent |
| FilterChain | ServletException |
| ServletRequestListener | UnavailableException |
| ServletRequestAttributeListener |  |
| ServletContextListener |  |
| ServletContextAttributeListener |  |

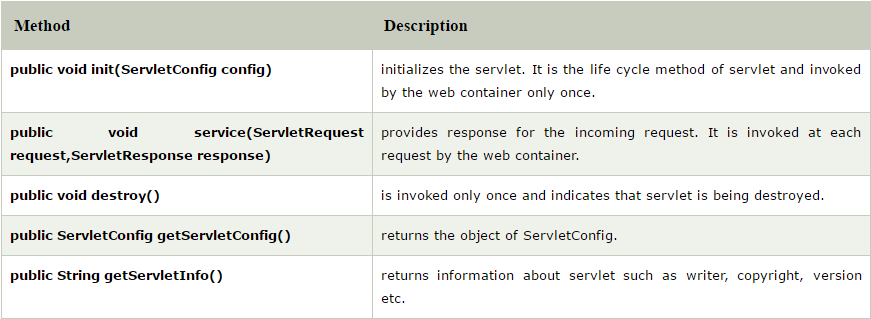
**javax.servlet.http package:**

|  |  |
| --- | --- |
| **Interfaces** | **Classes** |
| HttpServletRequest | HttpServlet |
| HttpServletResponse | Cookie |
| HttpSession | HttpServletRequestWrapper |
| HttpSessionListener | HttpServletResponseWrapper |
| HttpSessionAttributeListener | HttpSessionEvent |
| HttpSessionBindingListener | HttpSessionBindingEvent |
| HttpSessionActivationListener | HttpUtils (deprecated now) |
| HttpSessionContext (deprecated now) |  |

**Servlet Interface:**

It provides common behaviour to all the servlets. It 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:**



Example:

**import** java.io.\*;

**import** javax.servlet.\*;

**public** **class** First **implements** Servlet{

ServletConfig config=**null**;

**public** **void** init(ServletConfig config){

**this**.config=config;

System.out.println("servlet is initialized");

}

**public** **void** service(ServletRequest req,ServletResponse res)

**throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello simple servlet</b>");

out.print("</body></html>");

}

**public** **void** destroy(){System.out.println("servlet is destroyed");}

**public** ServletConfig getServletConfig(){**return** config;}

**public** String getServletInfo(){**return** "copyright 2007-1010";}

}

**GenericServlet class:**

It implements Servlet, ServletConfig and Serializable interfaces.

It provides the implementation of all the methods of these interfaces except the service method.

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

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

**Methods of GenericServlet class:**

1. **public void init(ServletConfig config):** 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():** 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.

**HttpServlet class:**

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

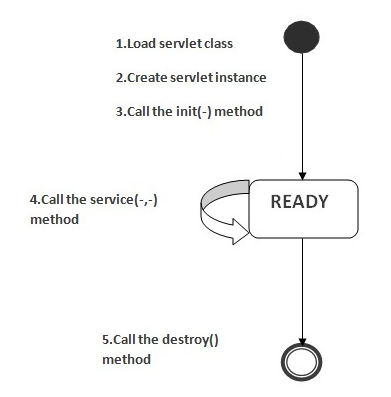
**Methods of Http Servlet class:**

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:**

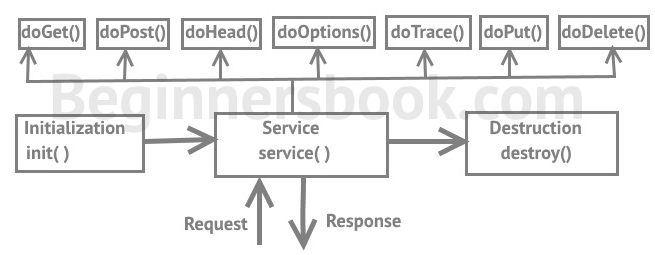
The web container maintains the life cycle of a servlet instance. It happens in 5 below steps:

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.



There are three states of a servlet: **new, ready and end.**

1. The servlet is in new state if servlet instance is created.
2. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks.
3. When the web container invokes the destroy() method, it shifts to the end state.



* **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.

* **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.

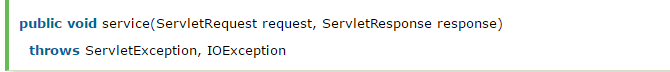
* **init method is invoked:**

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:



* **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:



* **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:



**Steps to create a Servlet with example:**

The servlet example can be created by three ways:

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

Mostly HttpServlet class is used because it provides http request specific method such as doGet(), doPost(), doHead() etc.

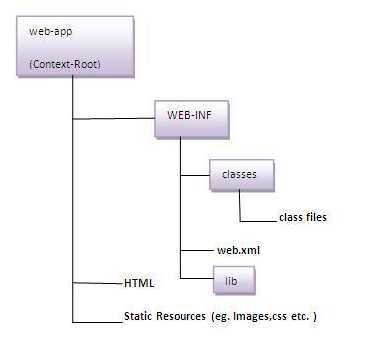
The steps to create Servlet are:

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
7. **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.

The web.xml file must be under the WEB-INF folder.



1. **Create a Servlet:**

There are three ways to create the servlet.

* By implementing the Servlet interface
* By inheriting the GenericServlet class
* By inheriting the HttpServlet class

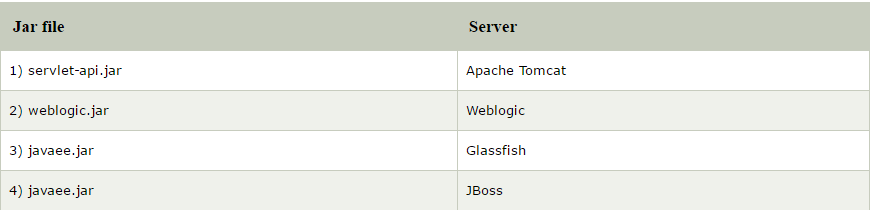
The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In the below 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.



1. **Compile the servlet:**

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



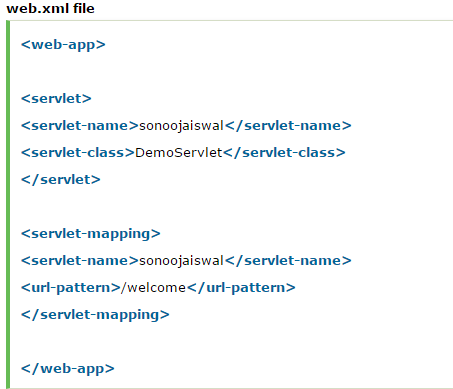
There are two ways to load the jar file:

1. set classpath
2. paste the jar file in JRE/lib/ext folder
3. **Create the deployment descriptor (web.xml file):**

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked.

The web container uses the Parser to get the information from the web.xml file. 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-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.   1. **Start the Server and deploy the project:** |

If you are using IDE like eclipse it’s easy to start the server.

1. **How to access the servlet**

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

http://localhost:9999/demo/welcome

**How a Servlet works?**

The server checks if the servlet is requested **for the first time**.

**If yes,** web container does the following tasks:

* loads the servlet class.
* instantiates the servlet class.
* calls the init method passing the ServletConfig object.

Else

* calls the service method passing request and response objects.

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

**How web container handles the servlet request?**

The web container is responsible to handle the request.

* maps the request with the servlet in the web.xml file.
* creates request and response objects for this request
* calls the service method on the thread
* The public service method internally calls the protected service method
* The protected service method calls the doGet method depending on the type of request.
* The doGet method generates the response and it is passed to the client.
* After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

**public service method:**

The public service method converts the ServletRequest object into the HttpServletRequest type and ServletResponse object into the HttpServletResponse type. Then, calls the service method passing these objects.



**welcome-file-list in web.xml:**

The **welcome-file-list** element of **web-app**, is used to define a list of welcome files. Its sub element is **welcome-file** that is used to define the welcome file.

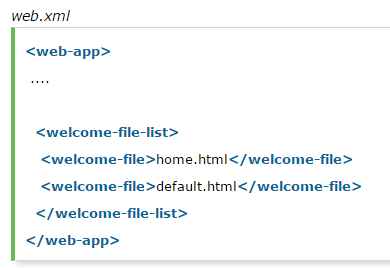
A **welcome file** is the file that is invoked automatically by the server, if you don't specify any file name. By default server looks for the welcome file in following order:

* welcome-file-list in web.xml
* index.html
* index.htm
* index.jsp

If none of these files are found, server renders 404 error.

If you have specified welcome-file in web.xml, and all the files index.html, index.htm and index.jsp exists, priority goes to welcome-file.

If welcome-file-list entry doesn't exist in web.xml file, priority goes to index.html file then index.htm and at last index.jsp file.



If you have the welcome file, you can directly invoke the project as given below:



No need to specify page name.

**load on startup in web.xml:**

The **load-on-startup** element of **web-app** loads the servlet at the time of deployment or server start if value is positive. It is also known as **pre initialization of servlet**.

We can have +ve as well as –ve value.

**Advantage**: Servlet is loaded at first request. That means it consumes more time at first request. If you specify the load-on-startup in web.xml, servlet will be loaded at project deployment time or server start. So, it will take **less time** for responding to first request.

**Passing positive value:**

If we pass positive value, the lower integer value servlet will be loaded before the higher integer value servlet. In other words, container loads the servlets in ascending integer value. The 0 value will be loaded first then 1, 2, 3 and so on.

****

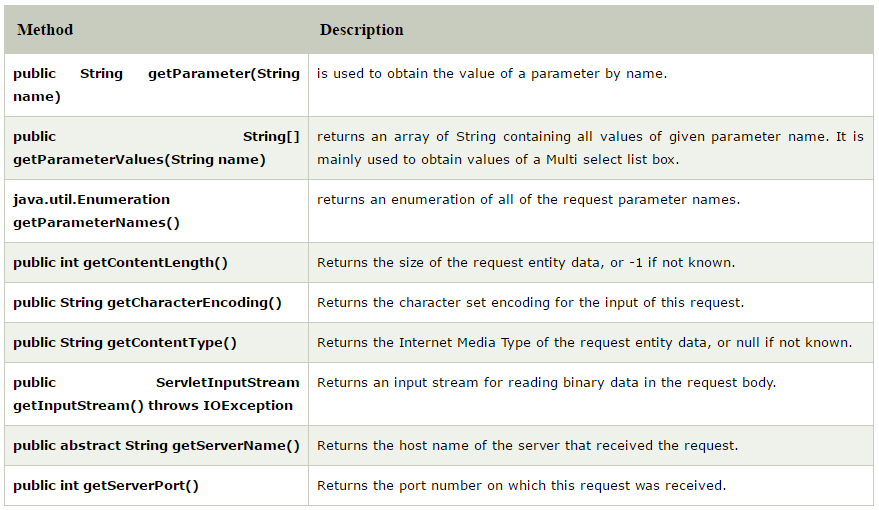
**Passing negative value:**

If you pass the negative value, servlet will be loaded at request time, at first request.

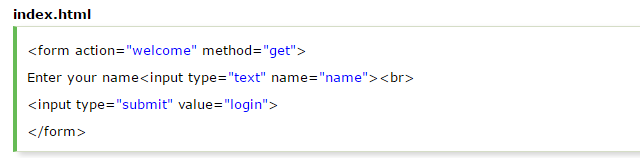
**ServletRequest Interface:**

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

**Methods of ServletRequest interface:**



Exapmle:





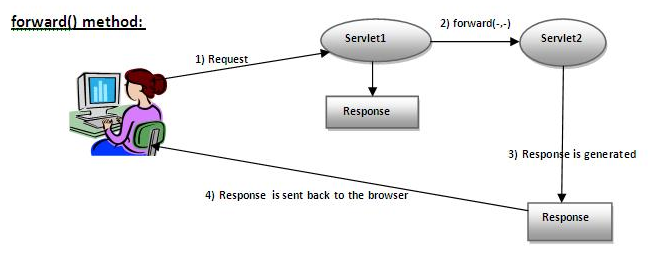
Here we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

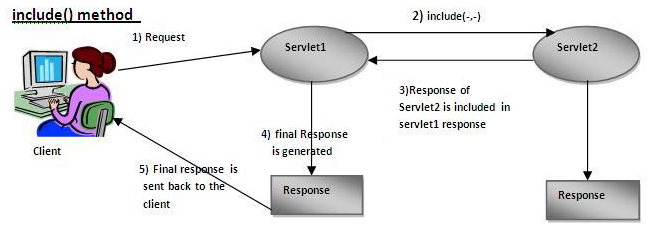
**RequestDispatcher in Servlet:**

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.

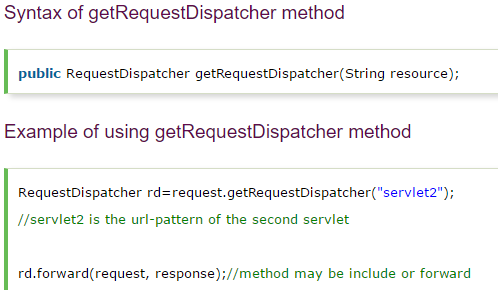
**Methods:**

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

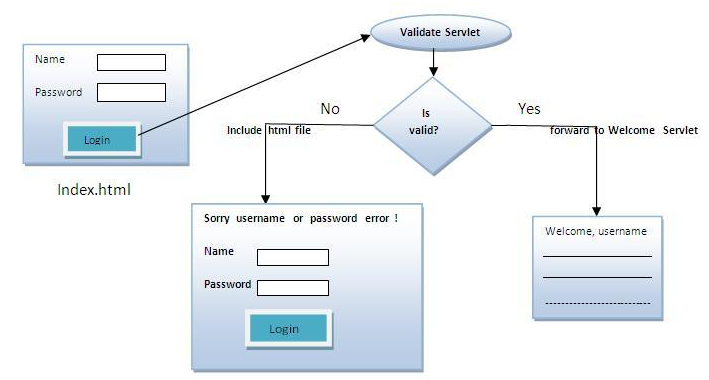


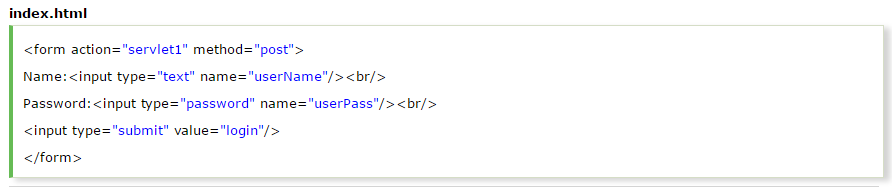


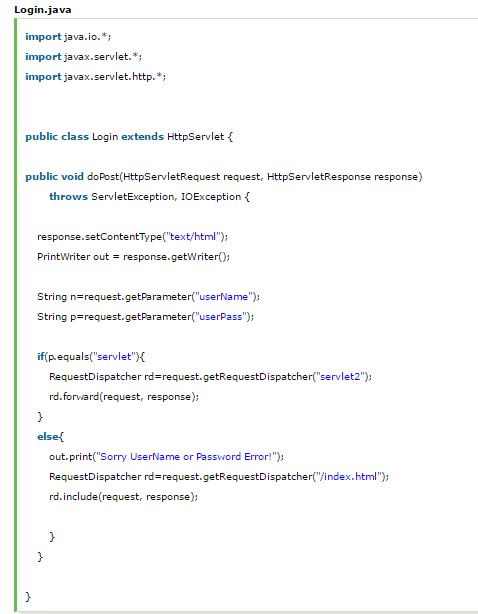
We can get the object of **RequestDispatcher** by using the **getRequestDispatcher()** method of **ServletRequest** interface.



**Simple Example:**







**SendRedirect in Servlet:**

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

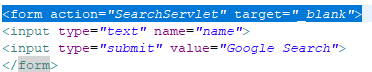
It accepts relative as well as absolute URL.

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 SendRedirect() and forward():**



**NOTE:**  Using sendRedirect if we want a new tab to be open then use **target = “\_blank”** in jsp with **form action.**



**ServletConfig Interface:**

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

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

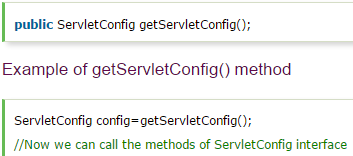
The main advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

**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.

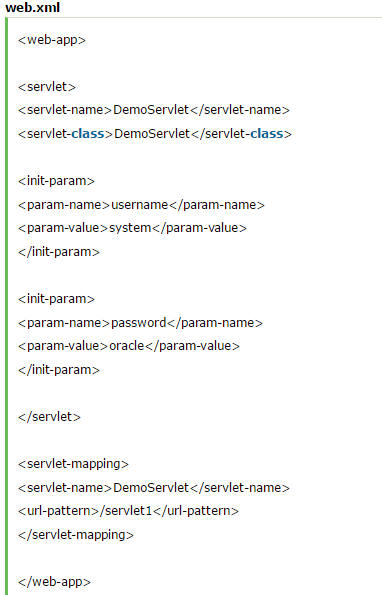
**Note: getServletConfig()** method is used to get the object of ServletConfig.

**Syntax:**



**Example:**





**ServletContext Interface:**

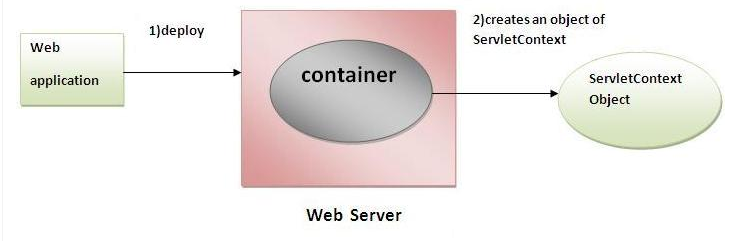
An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

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

**Advantage:** 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:**

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



**Methods:**

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.

**Methods 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.

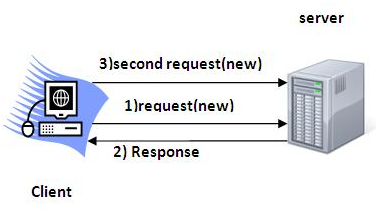
**NOTE:**

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

**Session Tracking in Servlets:**

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

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



It is used to recognize a particular user. It is done in 4 ways:

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

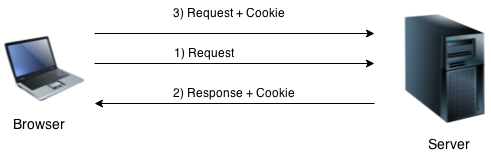
* **Cookies:**

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 it Works:**

By default, each request is considered as a new request. Every time 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.



**Type of Cookies:**

1. **Non-persistent cookie:**

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

1. **Persistent cookie:**

It is **valid for multiple sessions**. It is not removed each time when user closes the browser. It is removed only if user logout or sign-out.

**Advantage:**

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

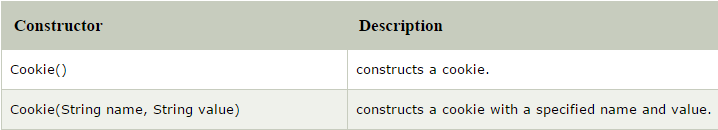
**Disadvantage:**

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

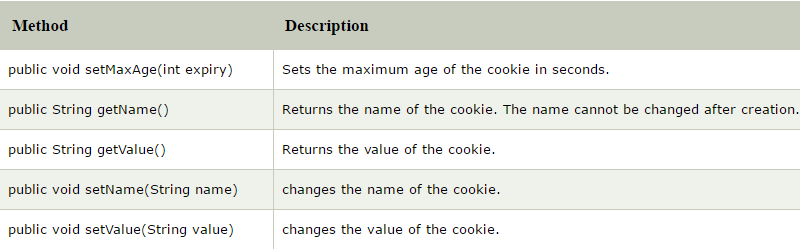
**Cookie Class:**

**javax.servlet.http.Cookie** class provides the functionality of using cookies.

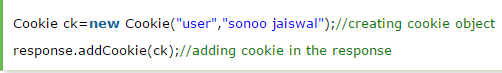
**Constructor:**



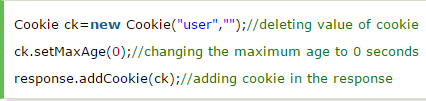
**Methods:**



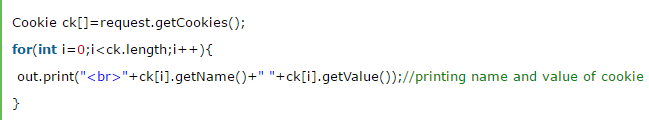
**Creation of Cookie:**



**Deletion of Cookie:**

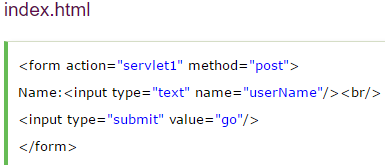


**How to get Cookie:**



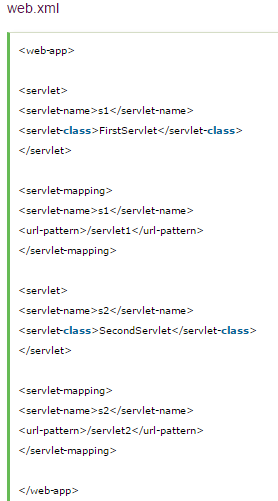
**Example of a Servlet Cookie:**

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.

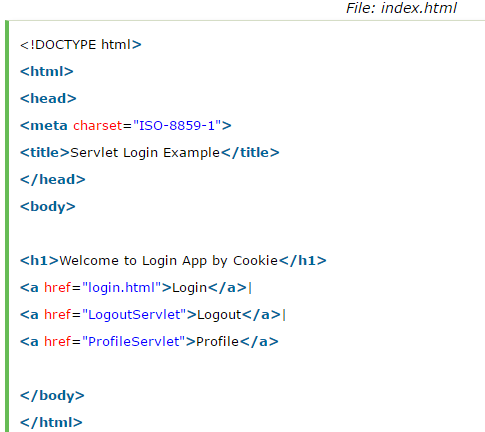


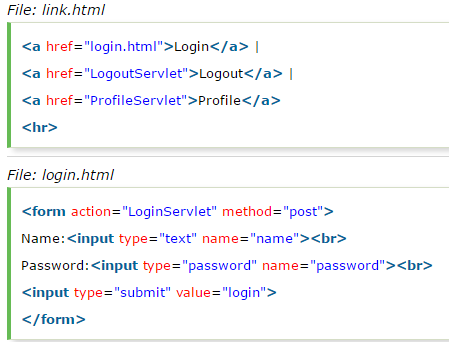


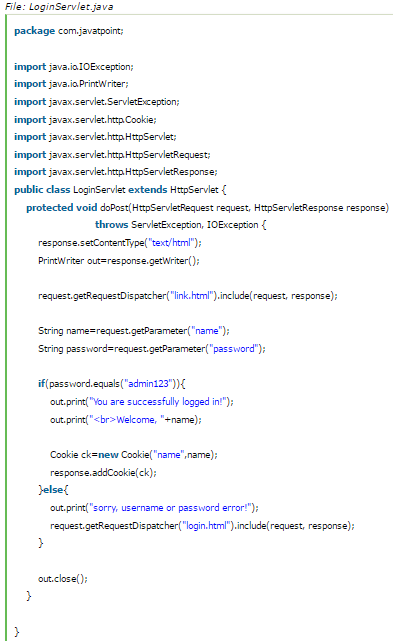




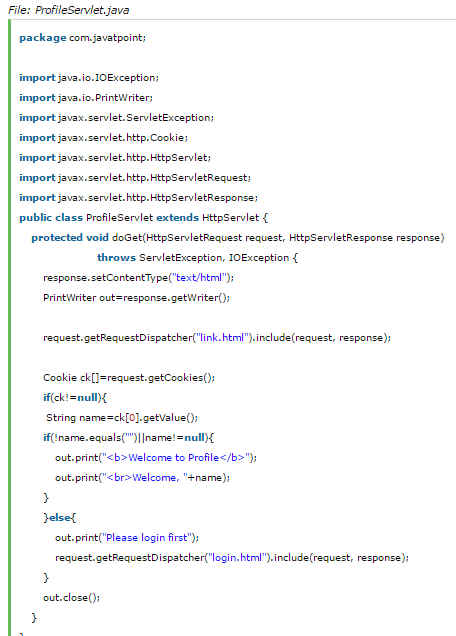
**Login and Logout example using Cookies:**













* **Hidden Form Field:**

A hidden (invisible) textfield is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.



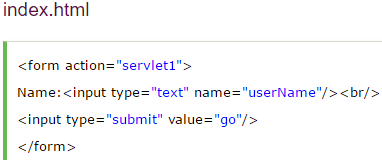
**Advantage:**

It will always work whether cookie is disabled or not.

**Disadvantage:**

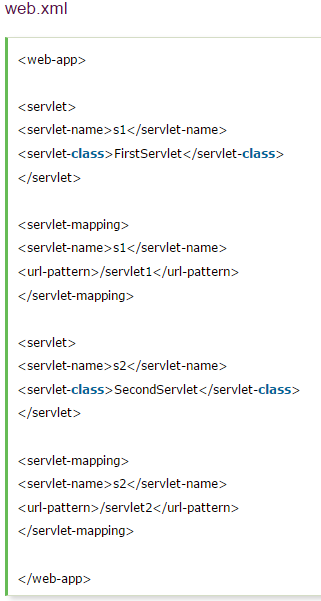
* It is maintained at server side.
* Extra form submission is required on each page.
* Only textual information can be used.

**Example:**







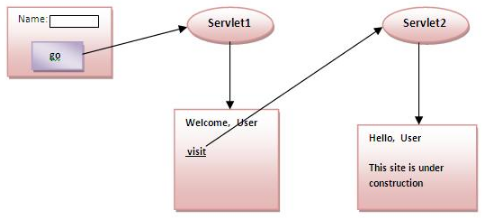


* **URL Rewriting:**

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

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

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



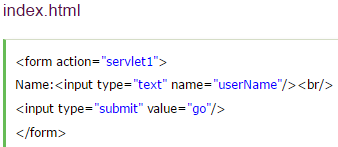
**Advantage:**

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

**Disadvantage:**

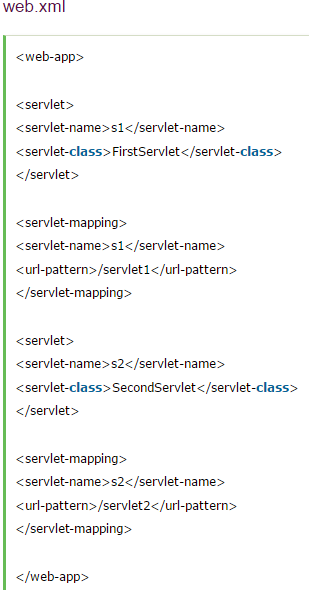
* It will work only with links.
* It can send only textual information.

**Example:**

****

****

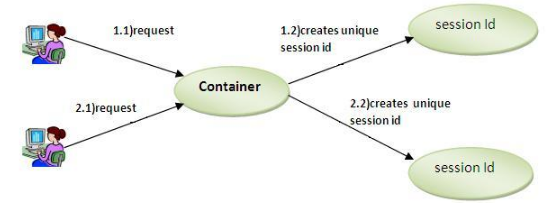
****

****

* **HttpSession interface:**

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

* Bind objects
* View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



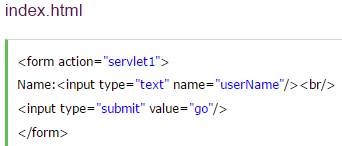
**Getting HttpSession object:**

* **public HttpSession getSession():** Returns the current session associated with this request, or if the request does not have a session, creates one.
* **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.

**Common Methods:**

* **public String getId():** Returns a string containing the unique identifier value.
* **public long getCreationTime():** Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
* **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.
* **public void invalidate():** Invalidates this session then unbinds any objects bound to it.

**Example:**







**Event and Listener in Servlet:**

Events are basically occurrence of something. Changing the state of an object is known as an event.

We can perform some important tasks at the occurrence of these exceptions, such as counting total and current logged-in users, creating tables of the database at time of deploying the project, creating database connection object etc.

**Event Classes:**

* ServletRequestEvent
* ServletContextEvent
* ServletRequestAttributeEvent
* ServletContextAttributeEvent
* HttpSessionEvent
* HttpSessionBindingEvent

**Event Interfaces:**

* ServletRequestListener
* ServletRequestAttributeListener
* ServletContextListener
* ServletContextAttributeListener
* HttpSessionListener
* HttpSessionAttributeListener
* HttpSessionBindingListener
* HttpSessionActivationListener
* **HttpSessionEvent and HttpSessionListner:**

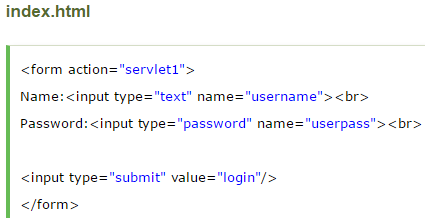
The HttpSessionEvent is notified when session object is changed. The corresponding Listener interface for this event is HttpSessionListener.

We can perform some operations at this event such as counting total and current logged-in users; maintain a log of user details such as login time, logout time etc.

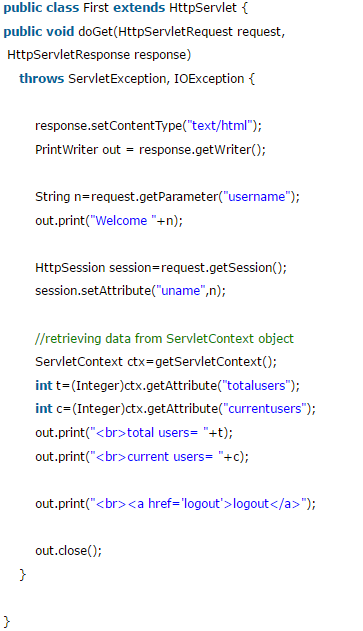
**Methods of HttpSessionListner:**

* **public void sessionCreated(HttpSessionEvent e)** : invoked when session object is created.
* **public void sessionDestroyed(ServletContextEvent e)** : invoked when session is invalidated.

**Example to count total current user and logged in user**:



****

****

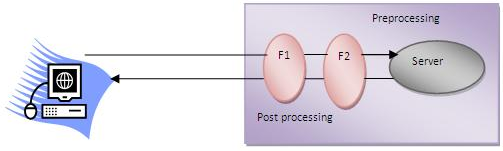
****

**Servlet Filter:**

A **filter** is an object that is invoked at the pre-processing and post-processing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

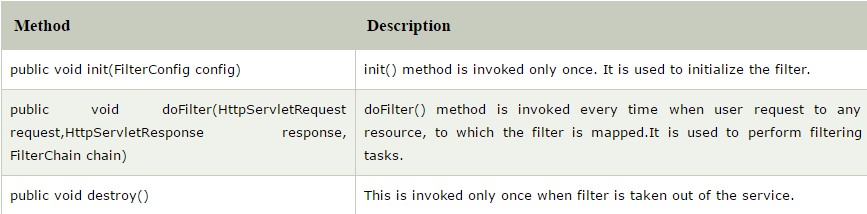
The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.



**Advantage:**

* Filter is pluggable.
* One filter doesn’t have dependency onto another resource.
* Less Maintenance

**Filter Interface:**



**FilterChain Interface:**

The object of FilterChain is responsible to invoke the next filter or resource in the chain. This object is passed in the doFilter method of Filter interface. The FilterChain interface contains only one method:

**public void doFilter(HttpServletRequest request, HttpServletResponse response):** it passes the control to the next filter or resource.

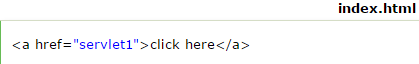
**FilterConfig:**

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

**Methods:**

* **public void init(FilterConfig config):** init() method is invoked only once it is used to initialize the filter.
* **public String getInitParameter(String parameterName):** Returns the parameter value for the specified parameter name.
* **public java.util.Enumeration getInitParameterNames():** Returns an enumeration containing all the parameter names.
* **public ServletContext getServletContext():** Returns the ServletContext object.

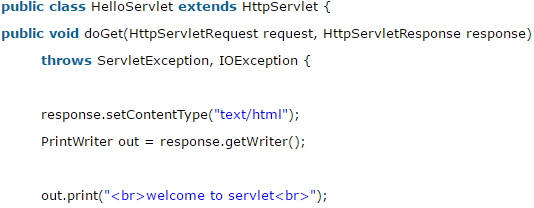
**Example:**



**MyFilter.java**



**HelloServlet.java**



**ServletInputStream class:**

**ServletInputStream** class provides stream to read binary data such as image etc. from the request object. It is an abstract class.

The **getInputStream()** method of **ServletRequest** interface returns the instance of ServletInputStream class.



**Method:**

**int readLine(byte[] b, int off, int len)**it reads the input stream.

**ServletOuputStream class:**

**ServletOutputStream** class provides a stream to write binary data into the response. It is an abstract class.

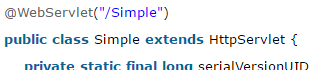
The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class.



**Servlet Annotations:**

Annotation represents the metadata. If you use annotation, deployment descriptor (web.xml file) is not required. But you should have tomcat7 as it will not run in the previous versions of tomcat. @WebServlet annotation is used to map the servlet with the specified name.

**Simple.java**



|  |  |
| --- | --- |
| **Sr.No.** | **Annotation & Description** |
| 1 | **@WebServlet:** To declare a servlet. |
| 2 | **@WebInitParam:** To specify an initialization parameter. |
| 3 | **@WebFilter:** To declare a servlet filter. |
| 4 | **@WebListener:** To declare a WebListener |
| 5 | **@HandlesTypes:** To declare the class types that a ServletContainerInitializer can handle. |
| 6 | **@HttpConstraint:** This annotation is used within the ServletSecurity annotation to represent the security constraints to be applied to all HTTP protocol methods for which a corresponding HttpMethodConstraint element does NOT occur within the ServletSecurity annotation. |
| 7 | **@HttpMethodConstraint:** This annotation is used within the ServletSecurity annotation to represent security constraints on specific HTTP protocol messages. |
| 8 | **@MultipartConfig:** Annotation that may be specified on a Servlet class, indicating that instances of the Servlet expect requests that conform to the multipart/form-data MIME type. |
| 9 | **@ServletSecurity:** This annotation is used on a Servlet implementation class to specify security constraints to be enforced by a Servlet container on HTTP protocol messages. |

**@WebServlet:**

The following table contains the list of attributes used for WebServlet annotation:

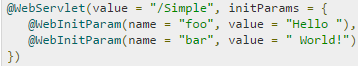
|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **String name:** Name of the Servlet |
| 2 | **String[] value:** Array of URL patterns |
| 3 | **String[] urlPatterns:** Array of URL patterns to which this Filter applies |
| 4 | **Int loadOnStartup:** The integer value gives you the startup ordering hint |
| 5 | **WebInitParam[] initParams:** Array of initialization parameters for this Servlet |
| 6 | **Boolean asyncSupported:** Asynchronous operation supported by this Servlet |
| 7 | **String smallIcon:** Small icon for this Servlet, if present |
| 8 | **String largeIcon:** Large icon for this Servlet, if present |
| 9 | **String description:** Description of this Servlet, if present |
| 10 | **String displayName:** Display name of this Servlet, if present |

**@WebInitParam:**

The @WebInitParam annotation is used for specifying an initialization parameter for a Servlet or a Filter. It is used within a WebFilter or WebServlet annotations. The following table contains the list of attributes used for WebInitParam annotation:

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **String name:** Name of the initialization parameter |
| 2 | **String value:** Value of the initialization parameter |
| 3 | **String description:** Description of the initialization parameter |

**Example:** usage of @WebInitParam:



**@WebFilter:**

|  |  |
| --- | --- |
| **Sr.No.** | **Attribute & Description** |
| 1 | **String filterName:** Name of the filter |
| 2 | **String[] urlPatterns:** Provides array of values or urlPatterns to which the filter applies |
| 3 | **DispatcherType[] dispatcherTypes:** Specifies the types of dispatcher (Request/Response) to which the filter applies |
| 4 | **String[] servletNames:** Provides an array of servlet names |
| 5 | **String displayName:** Name of the filter |
| 6 | **String description:** Description of the filter |
| 7 | **WebInitParam[] initParams:** Array of initialization parameters for this filter |
| 8 | **Boolean asyncSupported:** Asynchronous operation supported by this filter |
| 9 | **String smallIcon:** Small icon for this filter, if present |
| 10 | **String largeIcon:** Large icon for this filter, if present |