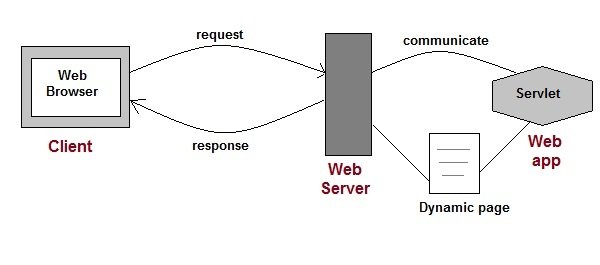
**Introduction to Servlet**

**Servlet** Technology is used to create web applications. **Servlet** technology uses Java language to create web applications.

Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

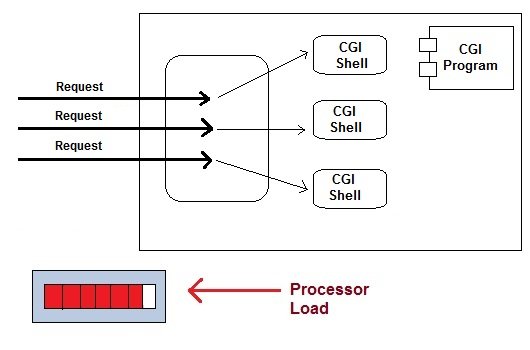


As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.

## **CGI (Common Gateway Interface)**

Before Servlets, CGI (Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works:

* User clicks a link that has URL to a dynamic page instead of a static page.
* The URL decides which CGI program to execute.
* Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
* The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.



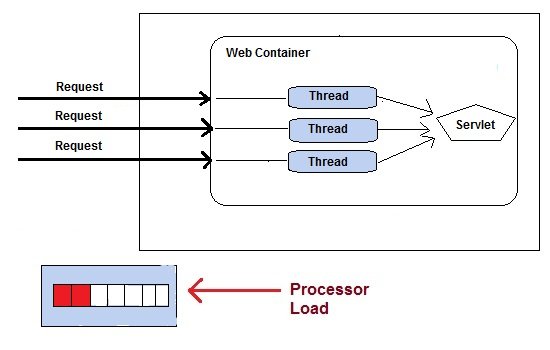
### **Drawbacks of CGI programs**

* High response time because CGI programs execute in their own OS shell.
* CGI is not scalable.
* CGI programs are not always secure or object-oriented.
* It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

## **Advantages of using Servlets**

* Less response time because each request runs in a separate thread.
* Servlets are scalable.
* Servlets are robust and object oriented.
* Servlets are platform independent.



## **HTTP Requests**

The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

The HTTP client sends the request to the server in the form of request message which includes following information:

* The Request-line
* The analysis of source IP address, proxy and port
* The analysis of destination IP address, protocol, port and host
* The Requested URI (Uniform Resource Identifier)
* The Request method and Content
* The User-Agent header
* The Connection control header
* The Cache control header

The HTTP request method indicates the method to be performed on the resource identified by the **Requested URI (Uniform Resource Identifier)**. This method is case-sensitive and should be used in uppercase.

The HTTP request methods are:

|  |  |
| --- | --- |
| **HTTP Request** | **Description** |
| **GET** | Asks to get the resource at the requested URL. |
| **POST** | Asks the server to accept the body info attached. It is like GET request with extra info sent with the request. |
| **HEAD** | Asks for only the header part of whatever a GET would return. Just like GET but with no body. |
| **TRACE** | Asks for the loopback of the request message, for testing or troubleshooting. |
| **PUT** | Says to put the enclosed info (the body) at the requested URL. |
| **DELETE** | Says to delete the resource at the requested URL. |
| **OPTIONS** | Asks for a list of the HTTP methods to which the thing at the request URL can respond |

## **Get vs. Post**

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

|  |  |
| --- | --- |
| **GET** | **POST** |
| 1) In case of Get request, only **limited amount of data**can be sent because data is sent in header. | In case of post request, **large amount of data**can be sent because data is sent in body. |
| 2) Get request is **not secured**because data is exposed in URL bar. | Post request is **secured**because data is not exposed in URL bar. |
| 3) Get request **can be bookmarked.** | Post request **cannot be bookmarked.** |
| 4) Get request is **idempotent**. It means second request will be ignored until response of first request is delivered | Post request is **non-idempotent.** |
| 5) Get request is **more efficient**and used more than Post. | Post request is **less efficient**and used less than get. |

## **GET and POST**

Two common methods for the request-response between a server and client are:

* **GET**- It requests the data from a specified resource
* **POST**- It submits the processed data to a specified resource

## **Anatomy of Get Request**

The query string (name/value pairs) is sent inside the URL of a GET request:

1. GET/RegisterDao.jsp?name1=value1&name2=value2

Some other features of GET requests are:

* It remains in the browser history
* It can be bookmarked
* It can be cached
* It have length restrictions
* It should never be used when dealing with sensitive data
* It should only be used for retrieving the data

## **Anatomy of Post Request**

The query string (name/value pairs) is sent in HTTP message body for a POST request:

1. POST/RegisterDao.jsp HTTP/1.1
2. Host: www. javatpoint.com
3. name1=value1&name2=value2

Some other features of POST requests are:

* This requests cannot be bookmarked
* This requests have no restrictions on length of data
* This requests are never cached
* This requests do not retain in the browser history

# Difference between Web Server and Application Server

A Server is a central place where information and programs are stored and accessed by applications over the network. Web Server is a server which accepts a request for data and sends the relevant document in return whereas Application Server contains a ejb container component as well to run the enterprise applications.

Following are the important differences between Web Server and Application Server.

| **Sr. No.** | **Key** | **Web Server** | **Application Server** |
| --- | --- | --- | --- |
| 1 | Purpose | Web Server contains Web container only. | Application Server contains Web Container plus EJB Container. |
| 2 | Useful | A web server is good in case of static contents like static html pages. | Applcation server is relevant in case of dynamic contents like bank websites. |
| 3 | Resource Consumption | Web server consumes less resources like CPU, Memory etc. as compared to application server. | Application server utilizes more resources. |
| 4 | Target Environment | Web Server provides the runtime environment for web applications. | Application server provides the runtime environment for enterprise applications. |
| 5 | Multithreading support | Multithreading is not supported. | Multithreading is supported. |
| 6 | Protocol(s) supported | Web Server supports HTTP Protocol. | Application Server suppots HTTP as well as RPC/RMI protocols. |
| 7 | Example | Apache Web Server. | Weblogic, JBoss. |

# Life Cycle of a Servlet (Servlet Life Cycle)

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

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



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

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
10. //writing html in the stream
11. pw.println("<html><body>");
12. pw.println("Welcome to servlet");
13. pw.println("</body></html>");
14. pw.close();//closing the stream
15. }}

### **3)Compile the servlet**

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

|  |  |
| --- | --- |
| **Jar file** | **Server** |
| 1) servlet-api.jar | Apache Tomcat |
| 2) weblogic.jar | Weblogic |
| 3) javaee.jar | Glassfish |
| 4) javaee.jar | JBoss |

### **Two ways to load the jar file**

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

### **4)Create the deployment descriptor (web.xml file)**

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

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

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

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

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

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

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

# War File

The WAR file (Web Application Resource or Web Application ARchive) is a container for JAR files, JavaServer Pages, Java Servlets, Java classes, XML files, tag libraries, static sites (HTML and associated files), and other resources that make up an online application. A file entitled web.xml is located in the /WEB-INF directory of the WAR file, and it describes the structure of the online application. The web.xml file isn’t technically essential if the online application is only providing JSP files. If the online apps utilize servlets, the servlet container looks at web.xml to figure out which servlet a URL request should go to.

### **Advantages**

1. Web applications may be easily tested and deployed.
2. The version of the deployed application may be easily identified.
3. WAR files are supported by all Java EE containers.
4. WAR files are supported by the MVC framework.

### **How to create a WAR File?**

On the command prompt, type the following command to generate a war file:

**jar -cvf project\_name.war\***

Here,

* -c: It is used to create a file
* -v: It is used to generate the verbose output
* -f: It is used to specify the archive filename
* \*: It signifies all the files of this directory

**Example – D:\apps\gfgapp>jar -cvf testapp1.war\***

### **How to deploy the war files?**

There are two ways to deploy the WAR File:

* Deploy WAR File to Tomcat – Console
* Deploy WAR File to Tomcat – Developer Portal

**Steps to be followed:**

* Copy the WAR file that has been updated.
* Paste the files into the designated folder.

Set up your Tomcat server to run as a service (optional), If you wish to manually deploy the war file in Apache Tomcat, navigate to the web app’s directory and paste the war file there.

# Servlet – Request Interface

When a Servlet accepts a call from a client, then it receives two objects, one is a ServletRequest and the other is the ServletResponse. ServletRequest encapsulates the Communications from the client to the server, while ServletResponse encapsulates the Communication from the Servlet back to the client. The Object of the ServletRequest is used to provide the client request information data to the Servlet as a content type, a content length, parameter names, and the values, header information and the attributes, etc.

**ServletRequest allows the Servlet to access information such as:**

* Names of the parameters are passed by the client
* The protocol [scheme] such as the HTTP POST and PUT methods being used by the client
* The names of the remote host that are made the request
* The server that received it
* An input stream is for reading the binary data from the request body

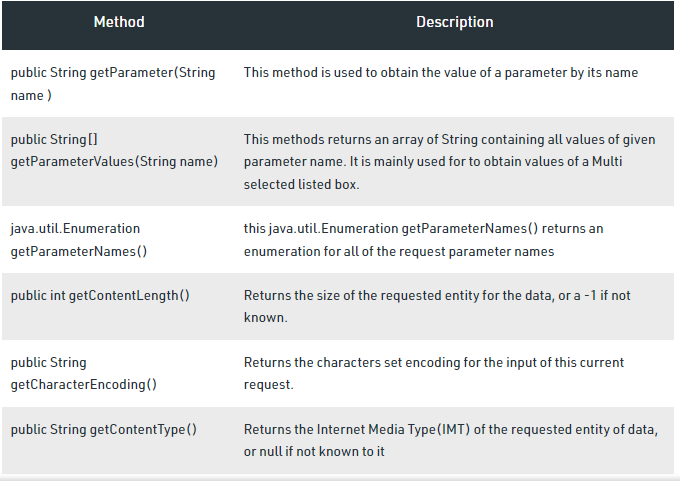
The Subclasses of the ServletRequest allows the Servlet to retrieve more protocol-based-specific data. For example, HttpServletRequest contains the methods for accessing HTTP-specific based header Information.

### **ServletResponse allows Servlet**

* To settle up the content length and mime type of the reply
* Provides an output stream and a Writer

Through ServletResponse, the Servlet can send the reply data. The Subclasses of ServletResponse provide the Servlet with more protocol-based-specific capabilities. For example, up ServletResponse can contain methods that allow the Servlet to Control the HTTP-specific header information.

### **Methods of ServletRequest Interface**



# ServletConfig Interface

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

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

### **Advantage of ServletConfig**

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

### **Methods of ServletConfig interface**

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

### **How to get the object of ServletConfig**

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

#### **Syntax of getServletConfig() method**

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

### **Example of getServletConfig() method**

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

# ServletContext Interface

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

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

### **Advantage of ServletContext**

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

### **Usage of ServletContext Interface**

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

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



### **Commonly used methods of ServletContext interface**

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

### **ServletConfig vs ServletContext Comparison Chart**

| **ServletConfig** | **ServletContext** |
| --- | --- |
| ServletConfig object is one per servlet class. | ServletContext object is global to the entire web application. |
| Object of ServletConfig will be created during the initialization process of the servlet. | Object of ServletContext will be created at the time of web application deployment |
| We have to give the request explicitly in order to create the ServletConfig object for the first time | ServletContext object can be available even before giving the first request |
| Scope: As long as a servlet is executing, the ServletConfig object will be available, it will be destroyed once the servlet execution is completed | Scope: As long as a web application is executing, the ServletContext object will be available, and it will be destroyed once the application is removed from the server |
| ServletConfig object is used while only one servlet requires information shared by it. | ServletContext object is used while application requires information shared by it |
| getServletConfig() method is used to obtain Servletconfig object | getServletContext() method is used to obtain ServletContext object |
| In web.xml — <init-param> tag will be appear under <servlet-class> tag. | In web.xml — <context-param> tag will be appear under <web-app> tag. |

**Note: ServletConfig is used by only single servlet to get configuration information from web.xml whereas ServletContext is used by multiple objects to get configuration information from xml files.**

# GenericServlet class

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

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

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

### **Methods of GenericServlet class**

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.\*;
3. **public** **class** First **extends** GenericServlet{
4. **public** **void** service(ServletRequest req,ServletResponse res)
5. **throws** IOException,ServletException{
6. res.setContentType("text/html");
7. PrintWriter out=res.getWriter();
8. out.print("<html><body>");
9. out.print("<b>hello generic servlet</b>");
10. out.print("</body></html>");
11. }
12. }

# HttpServlet class

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

### **Methods of HttpServlet class**

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

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** 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.

### **GenericServlet and HttpServlet Comparison Chart**

| **GenericServlet** | **HttpServlet** |
| --- | --- |
| It is defined by javax.servlet package. | It is defined by javax.servlethttp package. |
| It describes protocol-independent servlet | It describes protocol-dependent servlet. |
| GenericServlet is not dependent on any particular protocol. It can be used with any protocol such as HTTP, SMTP, FTP, and so on. | HttpServlet is a dependent protocol and is only used with HTTP protocol. |
| All methods are concrete except the service() method. service() method is an abstract method. | All methods are concrete (non-abstract). service() is non-abstract method. service() can be replaced by doGet() or doPost() methods. |
| The service method is abstract. | The service method is non-abstract |
| It forwards and includes a request and is also possible to redirect a request. | It forwards and includes a request but it is not possible to redirect the request. |
| GenericServlet doesn’t allow session management with cookies and HTTP sessions. | HTTPServlet allows session management with cookies and HTTP sessions. |
| It is an immediate child class of Servlet interface. | It is an immediate child class of GenericServlet class. |
| GenericServlet is a superclass of HttpServlet class. | HttpServlet is a subclass of GenericServlet class. |

# 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.

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.

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

**WelcomeServlet.java**

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

**web.xml**

1. <web-app>
2. <servlet>
3. <servlet-name>Login</servlet-name>
4. <servlet-**class**>Login</servlet-**class**>
5. </servlet>
6. <servlet>
7. <servlet-name>WelcomeServlet</servlet-name>
8. <servlet-**class**>WelcomeServlet</servlet-**class**>
9. </servlet>
10. <servlet-mapping>
11. <servlet-name>Login</servlet-name>
12. <url-pattern>/servlet1</url-pattern>
13. </servlet-mapping>
14. <servlet-mapping>
15. <servlet-name>WelcomeServlet</servlet-name>
16. <url-pattern>/servlet2</url-pattern>
17. </servlet-mapping>
18. <welcome-file-list>
19. <welcome-file>index.html</welcome-file>
20. </welcome-file-list>
21. </web-app>

# 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 forward() and sendRedirect() method**

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

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

### **Syntax of sendRedirect() method**

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

### **Example of sendRedirect() method**

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

### **Full example of sendRedirect method in servlet**

|  |
| --- |
| In this example, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, 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.\*;
4. **public** **class** DemoServlet **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();
10. response.sendRedirect("http://www.google.com");
11. pw.close();
12. }}

### **Creating custom google search using sendRedirect**

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

*index.html*

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

*MySearcher.java*

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

### **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. |

### **How to create Cookie?**

Let's see the simple code to create cookie.

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

### **How to delete Cookie?**

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

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

### **How to get Cookies?**

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

1. Cookie ck[]=request.getCookies();
2. **for**(**int** i=0;i<ck.length;i++){
3. out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
4. }

### **Simple example of Servlet Cookies**

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



### **index.html**

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

### **FirstServlet.java**

1. **import** java.io.\*;
2. **import** javax.servlet.\*;
3. **import** javax.servlet.http.\*;
4. **public** **class** FirstServlet **extends** HttpServlet {
5. **public** **void** doPost(HttpServletRequest request, HttpServletResponse response){
6. **try**{
7. response.setContentType("text/html");
8. PrintWriter out = response.getWriter();
9. String n=request.getParameter("userName");
10. out.print("Welcome "+n);
11. Cookie ck=**new** Cookie("uname",n);//creating cookie object
12. response.addCookie(ck);//adding cookie in the response
13. //creating submit button
14. out.print("<form action='servlet2'>");
15. out.print("<input type='submit' value='go'>");
16. out.print("</form>");
17. out.close();
18. }**catch**(Exception e){System.out.println(e);}
19. }
20. }

### **SecondServlet.java**

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

### **web.xml**

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