Servlets

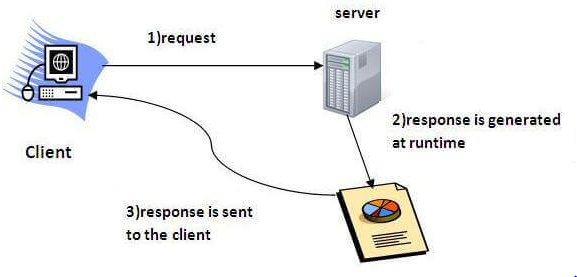
**Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).

**Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.

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

## **What is a Servlet?**

* Servlet is a technology which is used to create a web application.
* Servlet is an API that provides many interfaces and classes including documentation.
* Servlet is an interface that must be implemented for creating any Servlet.
* Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
* Servlet is a web component that is deployed on the server to create a dynamic web page.



### **What is a 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 elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

**Container:** It is used in java for dynamically generating the web pages on the server side.

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.

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)

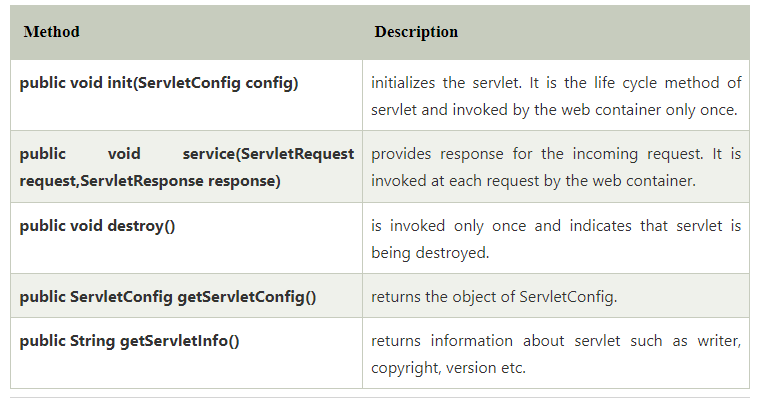
# Servlet Interface

Servlet interface defines methods that all servlets must implement.

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

### **Methods of Servlet interface**

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

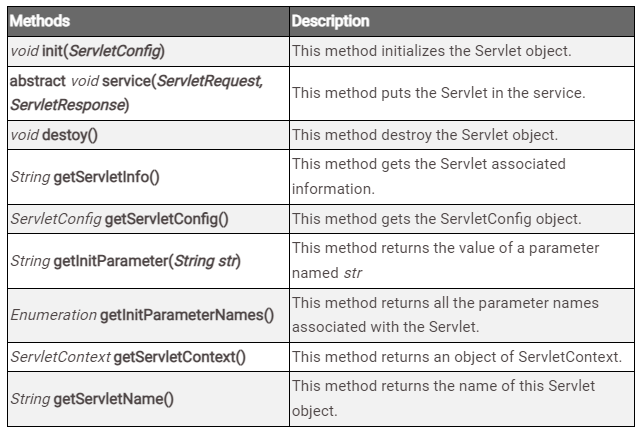


## **GenericServlet class:**

GenericServlet class implements the Servlet and ServletConfig interfaces. GenericServlet is protocol-independent. It not provides the implementation of service method.

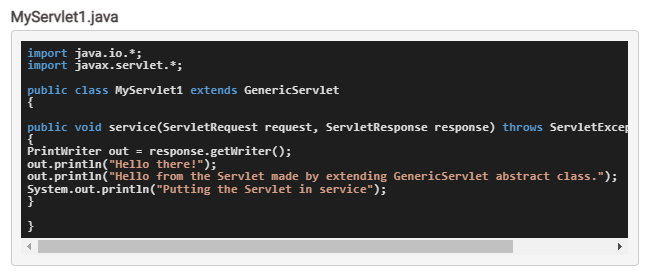
GenericServlet class is in javax.servlet package (javax.servlet.GenericServlet).

Let's take a look at the methods part of **javax.servlet.GenericServlet** class.



## *Creating the Servlet class by extending GenericServlet*

We are creating a Servlet extending the **GenericServlet** abstract class. **GenericServlet** class implements **Servlet** and **ServletConfig** interface. A Servlet class is just a regular Java class which ends with a .java extension, hence we have named this file **MyServlet1.java**.



## *Creating the Deployment Descriptor file*

As per the Java Servlet specifications, every web application based on Servlet must have a Deployment Descriptor file(an **XML** file) named **web.xml**. So, let's create one –



In *deployment descriptor* file, **<servlet>** has two child tags **<servlet-name>** and **<servlet-class>** :

* **<servlet-name>** tag is used to specify a *unique name for our Servlet class*, we have given it a unique name **MyGenericServlet**.

* **<servlet-class>** tag is used to specify the *full qualified name of the Servlet class* and in our example our Servlet class is named **MyServlet1**.

# HttpServlet Class In Java

HttpServelt is an abstract class, it comes under package ‘**javax.servlet.http.HttpServlet**‘ . To create a servlet the class must extend the HttpServlet class and override at least one of its methods (doGet, doPost, doDelete, doPut). The HttpServlet class extends the GenericServlet class and implements a Serializable interface.

### Methods of HttpServlet Class

### 1. doGet() Method

* This method is used to handle the GET request on the server-side.
* This method also automatically supports HTTP HEAD (HEAD request is a GET request which returns nobody in response ) request.
* The GET type request is usually used to *preprocess*a request.

### 2. doPost() Method

* This method is used to handle the POST request on the server-side.
* This method allows the client to send data of unlimited length to the webserver at a time.
* The POST type request is usually used to post-process a request.

### 3. doHead() Method

* This method is overridden to handle the HEAD request.
* In this method, the response contains the only header but does not contain the message body.
* This method is used to improve performance (avoid computing response body).

### 4. doPut() Method

* This method is overridden to handle the PUT request.
* This method allows the client to store the information on the server(to save the image file on the server).
* This method is called by the server (via the service method) to handle a PUT request.

### 5. doDelete() Method

* This method is overridden to handle the DELETE request.
* This method allows a client to remove a document or Web page from the server.
* While using this method, it may be useful to save a copy of the affected URL in temporary storage to avoid data loss.

### 6. doOptions() Method

* This method is overridden to handle the OPTIONS request.
* This method is used to determine which HTTP methods the server supports and returns an appropriate header.

### 7. doTrace() Method

* This method is overridden to handle the TRACE request.
* This method returns the headers sent with the TRACE request to the client so that they can be used in debugging.

### 8. getLastModified() Method

* This method returns the time the HttpServletRequest object was last modified.
* If time is unknown the method will return a negative number.
* This method makes browser and proxy caches work more effectively.
* also reducing the load on server and network resources.

### 9. service() Method

This method receives standard HTTP requests from the public *service*method and dispatches them to the *doXXX*methods defined in this class.

### 10. service() Method

This method is used to dispatch client requests to the protected service method.

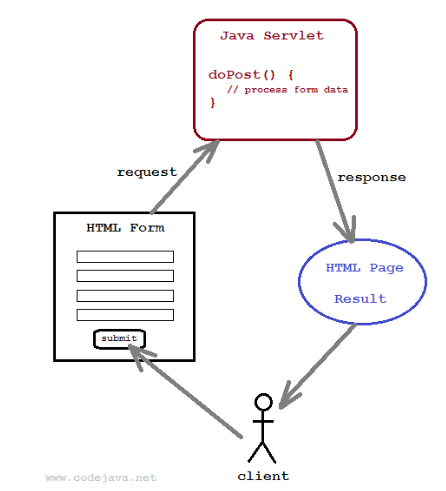
Linking Servlet with HTML

<https://www.codejava.net/java-ee/servlet/handling-html-form-data-with-java-servlet>

# How to handle HTML form data with Java Servlet

In this Java servlet tutorial, I will guide you how to read values of common input fields from HTML form on the server side with [Java Servlet](https://www.codejava.net/java-ee/servlet/quick-start-guide-for-java-servlet-annotations).

You know, handling form data represented in HTML page is a very common task in web development. A typical scenario is the user fills in fields of a form and submits it. The server will process the request based on the submitted data, and send response back to the client. The following picture depicts that workflow with Java servlet on the server side:



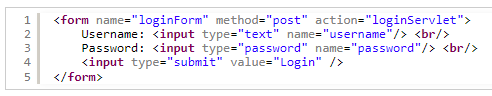
To create a form in HTML we need to use the following tags:

* + <form>: to create a form to add fields in its body.
  + <input>, <select>, <textarea>…: to create form fields like text boxes, dropdown list, text area, check boxes, radio buttons,… and submit button.

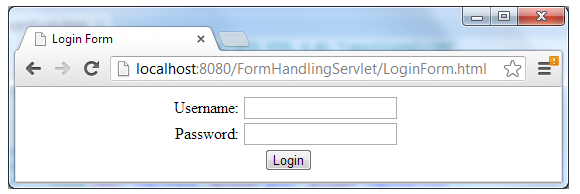
To make the form works with Java servlet, we need to specify the following attributes for the <form> tag:

* + **method**=”post”: to send the form data as an HTTP POST request to the server. Generally, form submission should be done in HTTP POST method.
  + **action**=”*URL of the servlet*”: specifies relative URL of the servlet which is responsible for handling data posted from this form.

For example, following is HTML code of a login form:



This form would look like this in browser:



On the server side, we need to create a Java servlet which is mapped to the URL: *loginServlet*, as specified in the form’s action attribute. Following is code of the servlet:



Notice that the servlet’s URL is specified by the [@WebServlet](https://www.codejava.net/java-ee/servlet/webservlet-annotation-examples) annotation before the servlet class. When the user submits the login form above, the servlet’s doPost() method will be invoked by the servlet container. Typically we will do the following tasks inside doPost() method:

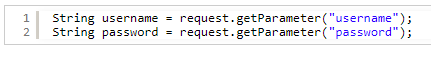
* + Read values of the fields posted from the form via the request object (implementation of javax.servlet.http.HttpServletRequest interface).
  + Do some processing, e.g. connecting to database to validate the username and password.
  + Return response back to the user via the respone object (implementation of javax.servlet.http.HttpServletResponse interface).

To read values of form’s fields, the HttpServletRequest interface provides the following methods:

* + **String getParameter(String name)**: gets value of a field which is specified by the given name, as a String. The method returns null if there is no form field exists with the given name.
  + **String[] getParameterValues(String name)**: gets values of a group of fields which have same name, in an array of String objects. The method returns null if there is no field exists with the given name.

Note that the above methods can also deal with parameters in URL’s query string, hence the name getParameter.

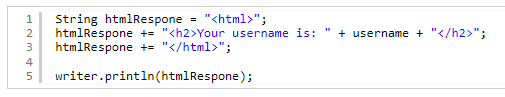
For example, we can write the following code in the doPost() method to read values of form’s fields:



To send response back to the client, we need to obtain a writer from the response object by calling the method getWriter() of the HttpServletResponse interface:



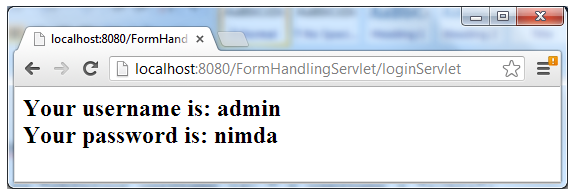
Then use the print() or println() method to deliver the response (in form of HTML code). For example:



Here’s complete code of the servlet class to process the login form:

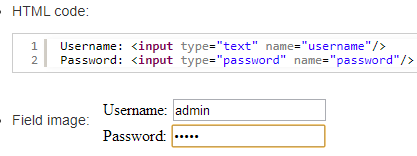


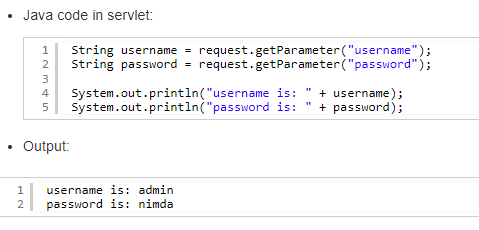
Here’s an example output when submitting the above login form in browser:



So far you have got the ins and outs when handling HTML form data with Java servlet. For your reference, we provide a list of examples for handling common HTML form fields as below. Note that we use the System.out.println() statement in servlet to demo the output.

## Read values of text field and password field

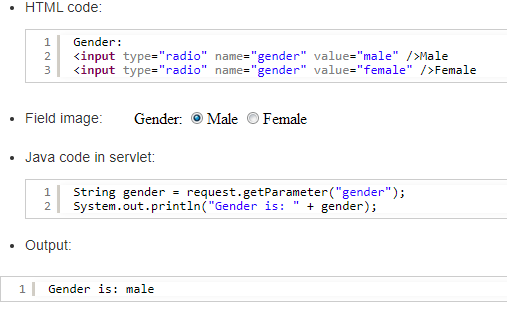




## 2. Read value of checkbox field



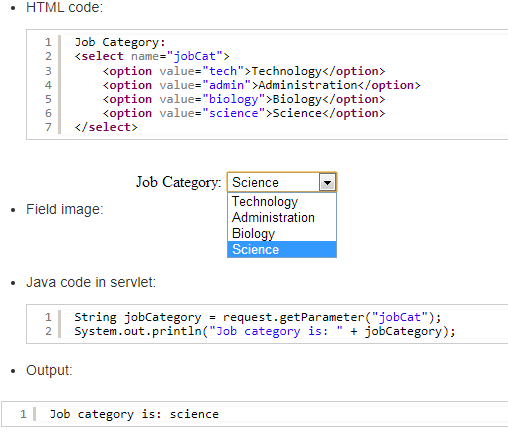
## 3. Read value of radio button field



## 4. Read value of text area field

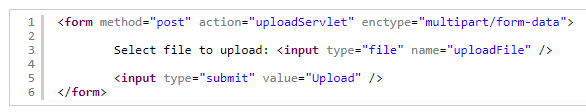


## 5. Read value of dropdown list (combobox) field



## 6. Read data of file upload field

To create a form to upload file, we need to specify the enctypeattribute for the <form> tag as follow:



For handling file upload on the server side with Java servlet, we recommend these tutorials:

* + [File upload servlet with Apache Common File Upload](https://www.codejava.net/java-ee/servlet/eclipse-file-upload-servlet-with-apache-common-file-upload).
  + [How to write upload file servlet with Servlet 3.0 API](https://www.codejava.net/java-ee/servlet/java-file-upload-example-with-servlet-30-api).

**Tomcat Server**

# What is Tomcat?

# It is an open-source Java servlet container that implements many Java Enterprise Specs such as the Websites API, Java-Server Pages and last but not least, the Java Servlet.

# The complete name of Tomcat is "Apache Tomcat" it was developed in an open, participatory environment and released in 1998

# it is a kind of software that enables the webserver to handle the dynamic(java-based) content using the Http protocols.

# if he wants to make website dynamic, he has to use servlet. We use the HTTP server if we want to send simple data. If we want to send dynamic data or to make our website dynamic, we need to use the servlet. Hence, we need an HTTP server and what else we need is a container where we will run or servlet, so when we combine the [HTTP](https://www.javatpoint.com/computer-network-http) server and the servlet (or we can say servlet container), they both combine to become a single server know as tomcat server.

### **What kind of server is Tomcat?**

The Java ecosystem supports a wide variety of application servers, so let's have a little discussion on each of them and see where Tomcat fits in:

# Used for the purpose of hosting Java servlets.

# We can say that, at the center, the Tomcat is [JSP (Java Server Pages)](https://www.javatpoint.com/jsp-tutorial) and Servlet. The JSP is one of the server-side programming technologies that enables the developers to create platform-independent dynamic content and also known as the server-side view rendering technology. A servlet is a java-based software component that helps in extending the capabilities of a server. However, it can also respond to several kinds of requests and generally implemented web server containers to host the web-applications on the webservers. As the developer's point of view, we just have to write the java server pages (or JSP) or the servlet and not required to worry about routing; the Tomcat will handle the routing.

**Installation steps**

### **Advantages of Tomcat:**

**It is open-source**

**Incredibly Lightweight**

**Highly flexible**

**Stability**

**It provides us an extra level of security**

**It is well documented**

**It is one of the most widely used application servers**

**JSP**

**Introduction**

* It stands for **Java Server Pages**.
* It is a server side technology.
* It is used for creating web application.
* It is used to create dynamic web content.
* In this JSP tags are used to insert JAVA code into HTML pages.
* It is an advanced version of Servlet Technology.
* It is a Web based technology helps us to create dynamic and platform independent web pages.
* In this, Java code can be inserted in HTML/ XML pages or both.
* JSP is first converted into servlet by JSP container before processing the client’s request.

**JSP pages are more advantageous than Servlet:**

* They are easy to maintain.
* No recompilation or redeployment is required.
* JSP has access to entire API of JAVA .
* JSP are extended version of Servlet.

**Features of JSP**

* **Coding in JSP is easy** :- As it is just adding JAVA code to HTML/XML.
* **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
* **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
* **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.
* **Portable, Powerful, flexible and easy to maintain** :- as these are browser and server independent.
* **No Redeployment and No Re-Compilation** :- It is dynamic, secure and platform independent so no need to re-compilation.
* **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags

**JSP syntax**

Syntax available in JSP are following

1. **Declaration Tag** :-It is used to declare variables.

**Syntax:-**

<%! Dec var %>

**Example:-**

<%! int var=10; %>

1. **Java Scriplets** :- It allows us to add any number of JAVA code, variables and expressions.

**Syntax:-**

<% java code %>

1. **JSP Expression** :- It evaluates and convert the expression to a string.

**Syntax:-**

<%= expression %>

**Example:-**

<% num1 = num1+num2 %>

1. **JAVA Comments** :- It contains the text that is added for information which has to be ignored.

**Syntax:-**

<% -- JSP Comments %>

**Process of Execution**

Steps for Execution of JSP are following:-

* + Create html page from where request will be sent to server eg try.html.
  + To handle to request of user next is to create .jsp file Eg. new.jsp
  + Create project folder structure.
  + Create XML file eg my.xml.
  + Create WAR file.
  + Start Tomcat
  + Run Application