# Java Servlets

Today, we all are aware of the need to create dynamic web pages i.e. the ones that can change the site contents according to the time or can generate the content according to the request received from the client. If you like coding in Java, then you will be happy to know that using Java there also exists a way to generate dynamic web pages and that way is Java Servlet. But before we move forward with our topic let's first understand the need for server-side extensions.

## What is Java Servlet?

Java Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, and then send a response back to the web server.

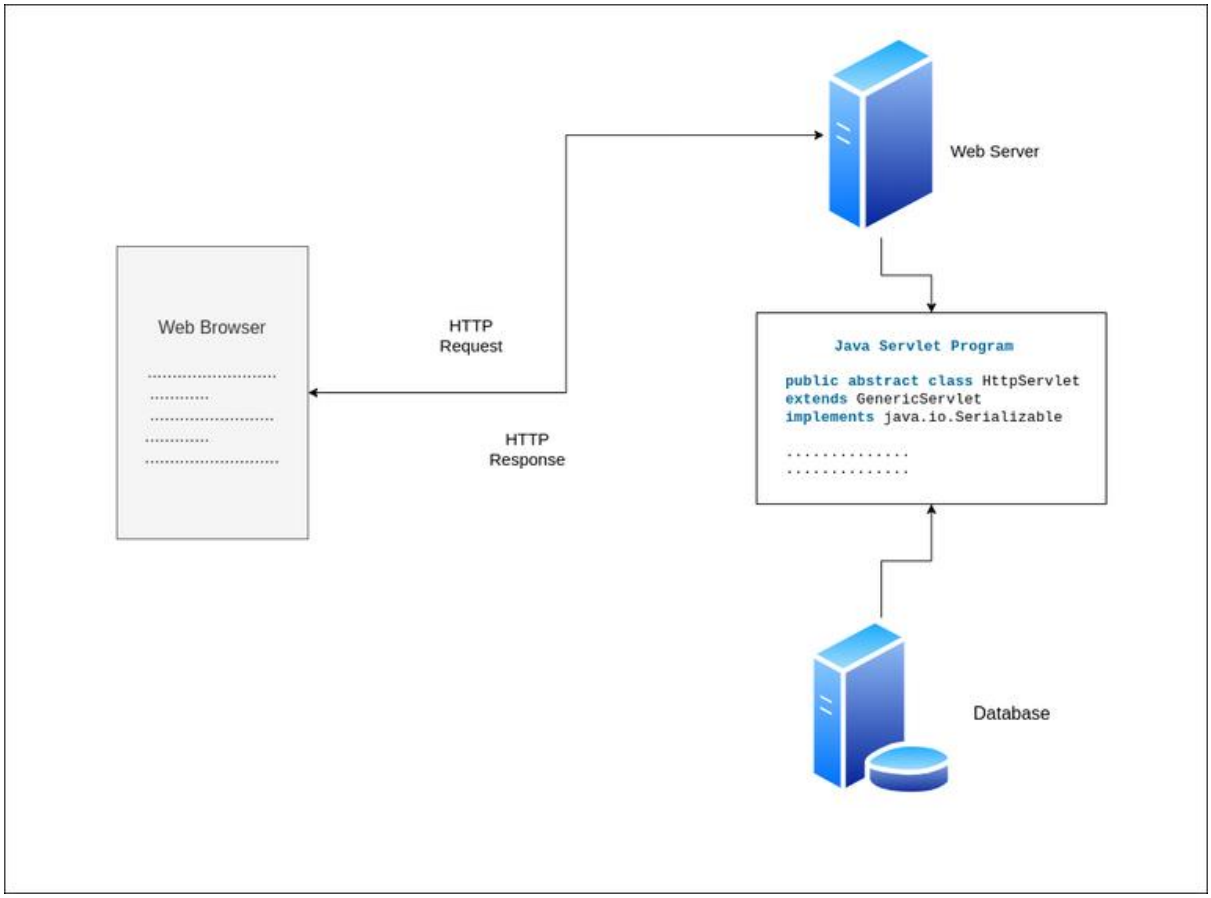
## Properties of Java Servlet

The properties of Servlets are as follows:

- Servlets work on the server side.
- Servlets are capable of handling complex requests obtained from the web server.

## Java Servlets Architecture

Servlet Architecture can be depicted from the image itself as provided below as follows:




**Execution of Java Servlets**

Execution of Servlets basically involves Six basic steps:

- The Clients send the request to the Web Server.
- The Web Server receives the request.
- The Web Server passes the request to the corresponding servlet.
- The Servlet processes the request and generates the response in the form of output.
- The Servlet sends the response back to the webserver.
- The Web Server sends the response back to the client and the client browser displays it on the screen.

Now let us do discuss eccentric point that why do we need For Server-Side extensions?

**Need of Server-Side Extensions**

The Server-Side Extensions are nothing but the technologies that are used to create dynamic Web pages. Actually, to provide the facility of dynamic Web pages, Web pages need a container or Web server. To meet this requirement, independent Web server providers offer some proprietary solutions in the form of APIs (Application Programming Interface).

These APIs allow us to build programs that can run with a Web server. In this case, Java Servlet is also one of the component APIs of Java Platform Enterprise Edition (nowdays known as – 'Jakarta EE') which sets standards for creating dynamic Web applications in Java.

Before learning about something, it's important to know the need for that something, it's not like that this is the only technology available for creating Dynamic Web Pages. The Servlet technology is similar to other Web server extensions such as Common Gateway Interface (CGI) scripts and Hypertext Preprocessor (PHP). However, Java Servlets are more acceptable since they solve the limitations of CGI such as low performance and low degree scalability.

## Servlets APIs

Servlets are built from two packages:

- javax.servlet(Basic)
- javax.servlet.http(Advance)

Various classes and interfaces present in these packages are:

| Component | Type | Package |
|---|---|---|
| Servlet | Interface | javax.servlet.* |
| ServletRequest | Interface | javax.servlet.* |
| ServletResponse | Interface | javax.servlet.* |
| GenericServlet | Class | javax.servlet.* |
| HttpServlet | Class | javax.servlet.http.* |
| HttpServletRequest | Interface | javax.servlet.http.* |
| HttpServletResponse | Interface | javax.servlet.http.* |
| Filter | Interface | javax.servlet.* |
| ServletConfig | Interface | javax.servlet.* |

**Advantages of a Java Servlet**

- Servlet is faster than CGI as it doesn't involve the creation of a new process for every new request received.
- Servlets, as written in Java, are platform independent.
- Removes the overhead of creating a new process for each request as Servlet doesn't run in a separate process. There is only a single instance that handles all requests concurrently. This also saves the memory and allows a Servlet to easily manage the client state.
- It is a server-side component, so Servlet inherits the security provided by the Web server.
- The API designed for Java Servlet automatically acquires the advantages of the Java platforms such as platform-independent and portability. In addition, it obviously can use the wide range of APIs created on Java platforms such as JDBC to access the database.
- Many Web servers that are suitable for personal use or low-traffic websites are offered for free or at extremely cheap costs eg. Java servlet. However, the majority of commercial-grade Web servers are rather expensive, with the notable exception of Apache, which is free.

**The Servlet Container**

Servlet container, also known as Servlet engine, is an integrated set of objects that provide a run time environment for Java Servlet components. In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

Services provided by the Servlet container:

- *Network Services:* Loads a Servlet class. The loading may be from a local file system, a remote file system or other network services. The Servlet container provides the network services over which the request and response are sent.
- *Decode and Encode MIME-based messages:* Provides the service of decoding and encoding MIME-based messages.
- *Manage Servlet container:* Manages the lifecycle of a Servlet.
- *Resource management* Manages the static and dynamic resources, such as HTML files, Servlets, and JSP pages.
- *Security Service:* Handles authorization and authentication of resource access.
- *Session Management:* Maintains a session by appending a session ID to the URL path.

## Life Cycle of a Servlet

The entire life cycle of a Servlet is managed by the Servlet container which uses the javax.servlet.Servlet interface to understand the Servlet object and manage it. So, before creating a Servlet object, let's first understand the life cycle of the Servlet object which is actually understanding how the Servlet container manages the Servlet object.

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

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

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

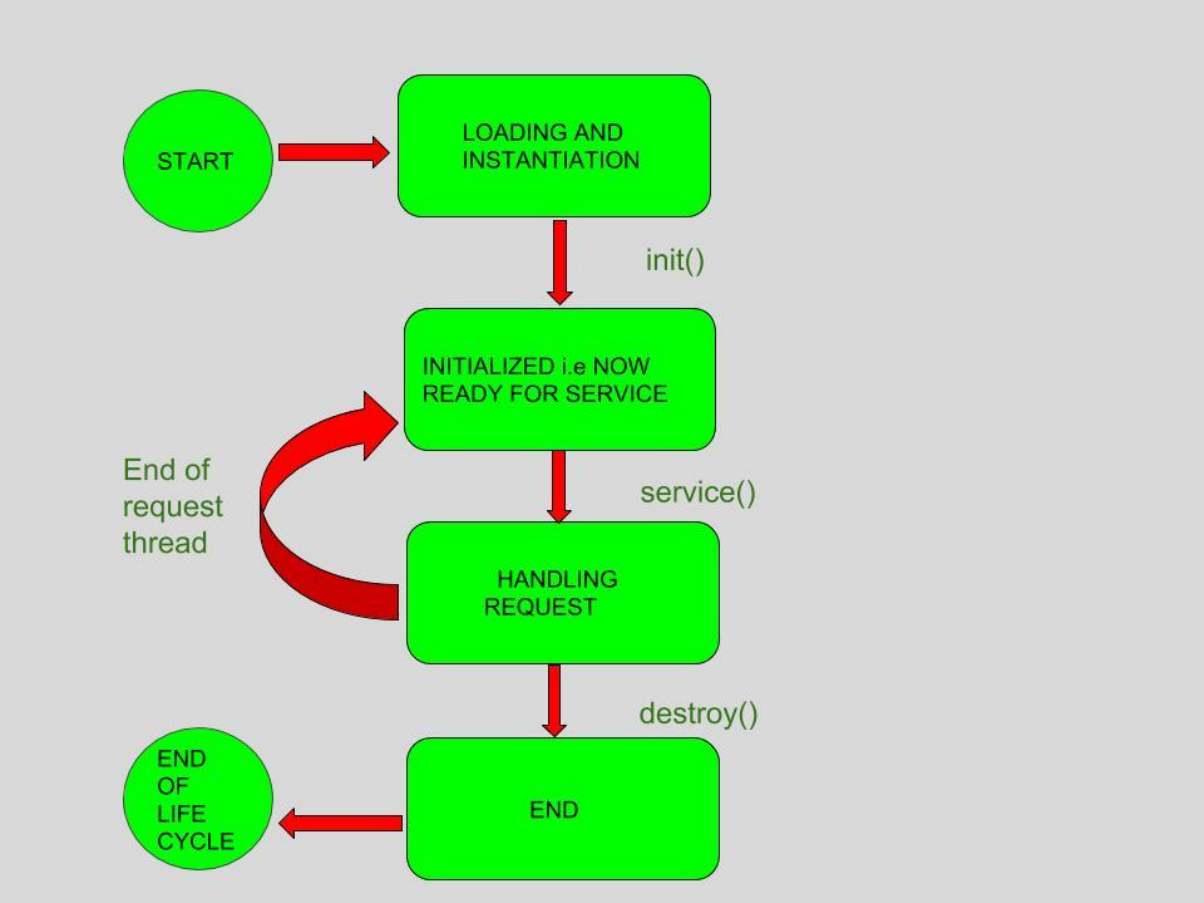
1. **Loading a Servlet:** The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container. The Web container or Servlet Container can load the Servlet at either of the following two stages :

- Initializing the context, on configuring the Servlet with a zero or positive integer value.
- If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

**Loading:** Loads the Servlet class.

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




2. **Initializing a Servlet:** After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object. The container initializes the Servlet object by invoking the Servlet.init(ServletConfig) method which accepts ServletConfig object reference as parameter.

The Servlet container invokes the Servlet.init(ServletConfig) method only once, immediately after the Servlet.init(ServletConfig) object is instantiated successfully. This method is used to initialize the resources, such as JDBC datasource.

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

3. **Handling request:** After initialization, the Servlet instance is ready to serve the client requests. The Servlet container performs the following operations when the Servlet instance is located to service a request:

- It creates the ServletRequest and ServletResponse objects. In this case, if this is a HTTP request, then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively.
- After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The service() method while processing the request may throw the ServletException or UnavailableException or IOException.

4. **Destroying a Servlet:** When a Servlet container decides to destroy the Servlet, it performs the following operations,

- It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
- After currently running threads have completed their jobs, the Servlet container calls the destroy() method on the Servlet instance.

After the destroy() method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

## HTTP Methods

HTTP (Hypertext Transfer Protocol) specifies a collection of request methods to specify what action is to be performed on a particular resource. The most commonly used HTTP request methods are GET, POST, PUT, PATCH, and DELETE. These are equivalent to the CRUD operations (create, read, update, and delete).

**GET:** GET request is used to read/retrieve data from a web server. GET returns an HTTP status code of 200 (OK) if the data is successfully retrieved from the server.

**POST:** POST request is used to send data (file, form data, etc.) to the server. On successful creation, it returns an HTTP status code of 201.

**PUT:** A PUT request is used to modify the data on the server. It replaces the entire content at a particular location with data that is passed in the body payload. If there are no resources that match the request, it will generate one.

**PATCH:** PATCH is similar to PUT request, but the only difference is, it modifies a part of the data. It will only replace the content that you want to update.

**DELETE:** A DELETE request is used to delete the data on the server at a specified location.

## Attributes in Servlets

An attribute in servlet is an object that can be set, get or removed by the following aspects

1. Request Scope
2. Application Scope
3. Session Scope

To pass the value from servlet to html/jsp files, setAttribute() method is called by the request object. setAttribute() method takes an input as an object which sends the data from servlet to the requesting website

```
public void setAttribute(String name, Object obj)
```

Sets the specified object in the application scope.

At the user-end, html uses a syntax by which attributes can be fetched

```
${ var-name }
```

in which var-name is same as name in setAttribute() method

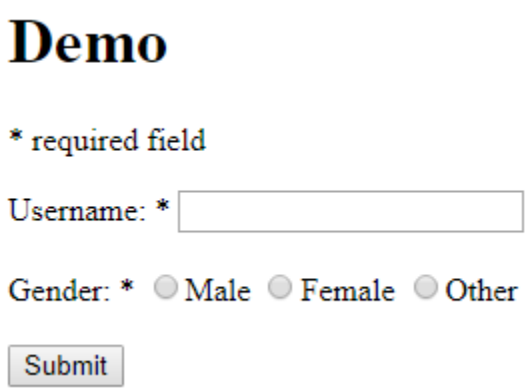
Let's look at an example of website which validates the form in server side

**HTML File 1 (Requesting Website)**

The code will send the input data to the Servlet to process the validation, which in return get the error text if any validation occurs.

```
<body>
<h1>Demo</h1>
<p style="color: black;">* required field</p>
<form method="post" action="./CommitServlet">
    <label>Username: *</label>

    <!-- Received response bundle data from the servlet as ${ var-name } -->

    <input type="text" value="${after.inputName}" name="inputName"/>
    <span name="errorName">${errors.Name}</span>
    <br/><br>

    <label>Gender: *</label>
    <input type="radio" name="gender" value="male" />Male
    <input type="radio" name="gender" value="female" />Female
    <input type="radio" name="gender" value="other" />Other
    <span name="errorGender">${errors.Gender}</span>
    <br><br>
    <input type="submit"/>
</form>
</body>
```

**Output:**




This Program process the requesting data and checks its validation, if any error encounter it will add the error text in the Bundle known as MAP class. This bundle is again sent to the requesting site for an error correction.

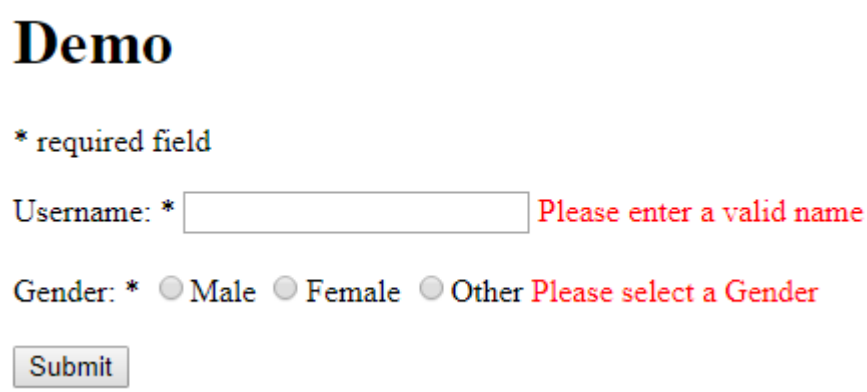
```
// Servlet code
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/CommitServlet")
public class CommitServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

```
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse
response)
        throws ServletException, IOException
    {

        // Create a Bundle of errors in the form of
Map
        Map<String, String> errors = new
HashMap<String, String>();
        Map<String, String> after = new
HashMap<String, String>();

        // Get the input values from the website
        String inputName =
request.getParameter("inputName");
        String inputGender =
request.getParameter("gender");

        // If error occur, previous entered data will
be reflected
        after.put("inputName", inputName);

        // Check for Validation of Name and Gender
        if (!validateName(inputName))

            // If error occur, create a entry for
            // the bundle and write a alert message
```

```
            errors.put("Name", "Please enter a valid
name");

        if (inputGender == null)

            // If Gender is not select, encounter an
error
            errors.put("Gender", "Please select a
Gender");

        if (errors.isEmpty())

            // If no error occur, redirect to the
response website
            response.sendRedirect("success.html");

        else {

            // Set this bundle into the request
attribute
            // and pass the data to the requested
website
            request.setAttribute("after", after);
            request.setAttribute("errors", errors);

    request.getRequestDispatcher("comment.jsp").forwa
rd(request, response);
        }
    }
```

```
    // Method to validate Proper Name, entered by the
user
    public static boolean validateName(String txt)
    {
        String regex = "^[a-zA-Z ]+$";
        Pattern pattern = Pattern.compile(regex,

    Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(txt);
        return matcher.find();
    }
}
```

**Output :**



# Servlet – RequestDispatcher

The RequestDispatcher is an Interface that comes under package javax.servlet. Using this interface we get an object in servlet after receiving the request. Using the RequestDispatcher object we send a request to other resources which include (servlet, HTML file, or JSP file). A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

How to Create an Object of RequestDispatcher?

There are three ways to get an object:

**1. RequestDispatcher**
requestDispatcher=ServletContext.getRequestDispatcher(String path);

*Description:*

- public interface ServletContext. Defines a set of methods that a servlet uses to communicate with its servlet container.
- path is a string specifying the pathname to the resource(servlet, HTML file, or JSP file).

**2. RequestDispatcher**
requestDispatcher=ServletContext.getNamedDispatcher(String name);

*Description:*

- public interface ServletContext. Defines a set of methods that a servlet uses to communicate with its servlet container.
- name is a string specifying the name of a servlet to wrap.

**3. RequestDispatcher**
**requestDispatcher=request.getRequestDispatcher(“String path”);**

*Description:*

- request is the HttpServletRequest type object.
- path is a string specifying the pathname to the resource. If it is relative, it must be relative to the current servlet.

**Method and Description**
The class contains two methods:

**1. forward**

*Syntax:*

void forward(ServletRequest request,ServletResponse response) throws ServletException,IOException

*Description:*

- Modifier and Type:- void
- This method is used to forward a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
- The method get called before the response has been sent to the client. If the response is already sent then the method will throws an IllegalStateException.
- The parameter request(HttpServletRequest type) and response(HttpServletResponse type) are the same objects as were passed to the calling servlet’s service method.
- This method sets the dispatcher type of the given request to DispatcherType.FORWARD.

**Example:**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GFG extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
    {
        // Perform all the work as per your
        // application's architecture
        try {
            RequestDispatcher requestDispatcher;

            // path is a string specifying the
pathname to
            // the resource. If it is relative, it
must be
            // relative against the current servlet

    requestDispatcher=request.getRequestDispatcher("p
ath");
            requestDispatcher.forward(request,
response);
        }
        catch (ServletException servletException) {
        }
        catch (IOException ioException) {
        }
```

```
        catch (IllegalStateException
illegalStateException) {
        }
    }
}
```

*Note:* The above code will not run in online IDE this is server-side code.

**2. include**

*Syntax:*

void include(ServletRequest request,ServletResponse response) throws ServletException,IOException

*Description:*

- Modifier and Type:- void
- This method is used to include the response of resource(for which the request passed servlet, JSP page, HTML file) in the current servlet response.
- The parameter request(HttpServletRequest type) and response(HttpServletResponse type) are the same objects as were passed to the calling servlet's service method.
- This method sets the dispatcher type of the given request to DispatcherType.INCLUDE.

**Example:**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GFG extends HttpServlet {
```

```
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
    {
        // Perform all the work as
        // per your application's architecture
        try {
            RequestDispatcher requestDispatcher;

            // path is a string specifying the
pathname to
            // the resource. If it is relative, it
must be
            // relative against the current servlet

    requestDispatcher=request.getRequestDispatcher("p
ath");
            requestDispatcher.include(request,
response);
        }
        catch (ServletException servletException) {
        }
        catch (IOException ioException) {
        }
    }
}
```

*Note:* The above code will not run in online IDE this is server-side code.