Writing Servlets

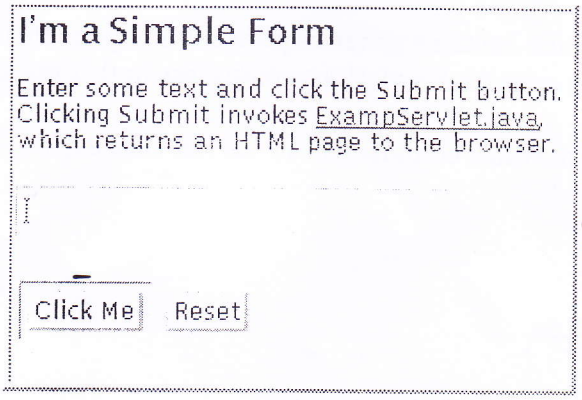
A servlet is an extension to a server that enhances the server's functionality. The most common use for a servlet is to extend a web server by providing dynamic web content. Web servers display documents written in HyperText Markup Language (HTML) and respond to user requests using the HyperText Transfer Protocol (HTTP). HTTP is the protocol for moving hypertext files across the internet. HTML documents contain text that has been marked up for interpretation by an HTML browser such as Netscape.

Servlets are easy to write. All you need is the Java® 2 Platform software, and JavaServer™ Web Development Kit (JWSDK).

About the Example

A browser accepts end user input through an HTML form. The simple form used in this lesson has one text input field for the end user to enter text and a Submit button. When the end user clicks the Submit button, the simple servlet is invoked to process the end user input.

In this example, the simple servlet returns an HTML page that displays the text entered by the end user.

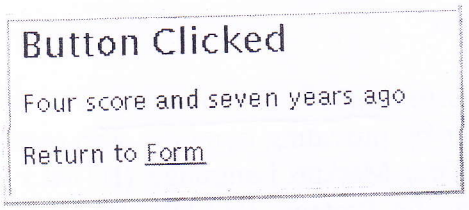


HTML Form

The HTML form is embedded in this HTML file. The diagram shows how the HTML page looks when it is opened in a browser.

The HTML file and form are similar to the simple application and applet examples so you can compare the code and learn how servlets, applets, and applications handle end user inputs. When the user clicks the Click Me button, the servlet gets the entered text, and returns an HTML page with the text.

The HTML page returned to the browser by the ExampServlet.java servlet is shown below. The servlet code to retrieve the user's input and generate the HTML page follows with a discussion.



Note: To run the example, you have to put the servlet and HTML files in the correct directories for the Web server you are using. For example, with Java WebServer 1.1.1, you place the servlet in the ~/JavaWebServer1.1.1/servlets and the HTML file in the ~/JavaWebServer1.1.1/public_html directory.

Servlet Backend

ExampServlet.java builds an HTML page to return to the end user. This means the servlet code does not use any Project Swing or Abstract Window Toolkit (AWT) components or have event handling code. For this simple servlet, you only need to import these packages:

- java.io for system input and output. The HttpServlet class uses the IOException class in this package to signal that an input or output exception of some kind has occurred.
- javax.servlet, which contains generic (protocol-independent) servlet classes. The HttpServlet class uses the ServletException class in this package to indicate a servlet problem.
- javax.servlet.http, which contains HTTP servlet classes. The HttpServlet class is in this package.

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

public class ExampServlet extends HttpServlet {
  public void doPost(HttpServletRequest request,
            HttpServletResponse response)
      throws ServletException, IOException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<title>Example</title>" +
      "<body bgcolor=FFFFFF>");

    out.println("<h2>Button Clicked</h2>");

    String DATA = request.getParameter("DATA");
```

```
  if(DATA != null){
    out.println(DATA);
  } else {
    out.println("No text entered.");
  }

  out.println("<P>Return to
        <A HREF="../simpleHTML.html">Form</A>");
  out.close();
 }
}
```

Class and Method Declarations

All servlet classes extend the HttpServlet abstract class. HttpServlet simplifies writing HTTP servlets by providing a framework for handling the HTTP protocol. Because HttpServlet is abstract, your servlet class must extend it and override at least one of its methods. An abstract class is a class that contains unimplemented methods and cannot be instantiated itself.

```
public class ExampServlet extends HttpServlet {
 public void doPost(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException
 {
```

The ExampServlet class is declared public so the web server that runs the servlet, which is not local to the servlet, can access it.

The ExampServlet class defines a doPost method with the same name, return type, and parameter list as the doPost method in the HttpServlet class. By doing this, the ExampServlet class overrides and implements the doPost method in the HttpServlet class.

The doPost method performs the HTTP POST operation, which is the type of operation specified in the HTML form used for this example. The other possibility is the HTTP GET operation, in which case you would implement the doGet method instead.

In short, POST requests are for sending any amount of data directly over the connection without changing the URL, and GET requests are for getting limited amounts of information appended to the URL. POST requests cannot be bookmarked or emailed and do not change the Uniform Resource Locators (URL) of the response. GET requests can be bookmarked and emailed and add information to the URL of the response.

The parameter list for the doPost method takes a request and a response object. The browser sends a request to the servlet and the servlet sends a response back to the browser.

The doPost method implementation accesses information in the request object to find out who made the request, what form the request data is in, and which HTTP headers were sent, and uses the response object to create an HTML page in response to the browser's request. The doPost method throws an IOException if there is an input or output problem when it handles the request, and a ServletException if the request could not be handled. These exceptions are handled in the HttpServlet class.

Method Implementation

The first part of the doPost method uses the response object to create an HTML page. It first sets the response content type to be text/html, then gets a PrintWriter object for formatted text output.

```
response.setContentType("text/html");
  PrintWriter out = response.getWriter ();

  out.println("<title>Example</title>" +
    "<body bgcolor=#FFFFFF>");

  out.println("<h2>Button Clicked</h2>");
```

The next line uses the request object to get the data from the text field on the form and store it in the DATA variable. The getparameter method gets the named parameter, returns null if the parameter was not set, and an empty string if the parameter was sent without a value.

```
  String DATA = request.getParameter("DATA");
```

The next part of the doPost method gets the data out of the DATA parameter and passes it to the response object to add to the HTML response page.

```
  if(DATA != null){
    out.println(DATA);
  } else {
    out.println("No text entered.");
  }
```

The last part of the doPost method creates a link to take the end user from the HTML response page back to the original form, and closes the response.

```
  out.println("<P>Return to
     <A HREF="../simpleHTML.html">Form</A>");
  out.close();
 }
```

Note: To learn how to use the other methods available in the HttpServlet, HttpServletRequest, and HttpServletResponse classes.

Package : javax.servlet

Interfaces

Interface javax.servlet.Servlet

public interface **Servlet**
This interface is for developing servlets. A servlet is a body of Java code that is loaded into and runs inside a servlet engine, such as a web server. It receives and responds to requests from clients. For example, a client may need information from a database; a servlet can be written that receives the request, gets and processes the data as needed by the client, and then returns it to the client.

All servlets implement this interface. Servlet writers typically do this by subclassing either GenericServlet, which implements the Servlet interface, or by subclassing GenericServlet's descendent, HttpServlet. Developers need to directly implement this interface only if their servlets cannot (or choose not to) inherit from GenericServlet or HttpServlet. For example, RMI or CORBA objects that act as servlets will directly implement this interface.

The Servlet interface defines methods to initialize a servlet, to receive and respond to client requests, and to destroy a servlet and its resources. These are known as life-cycle methods, and are called by the network service in the following manner:

1. Servlet is created then **init**ialized.
2. Zero or more **service** calls from clients are handled
3. Servlet is **destroy**ed then garbage collected and finalized

Initializing a servlet involves doing any expensive one-time setup, such as loading configuration data from files or starting helper threads. Service calls from clients are handled using a request and response paradigm. They rely on the underlying network transport to provide quality of service guarantees, such as reordering, duplication, message integrity, privacy, etc. Destroying a servlet involves undoing any initialization work and synchronizing persistent state with the current in-memory state.

In addition to the life-cycle methods, the Servlet interface provides for a method for the servlet to use to get any startup information, and a method that allows the servlet to return basic information about itself, such as its author, version and copyright.

Methods

1. init

public abstract void init(ServletConfig config) throws ServletException

Initializes the servlet. The method is called once, automatically, by the servlet engine when it loads the servlet. It is guaranteed to finish before any service requests are accepted.

The init method should save the ServletConfig object so that it can be returned by the getServletConfig method. If a fatal initialization error occurs, the init method should throw an appropriate "UnavailableException" exception.

Parameters:

config - object containing the servlet's startup- configuration and initialization parameters
Throws: ServletException
if a servlet exception has occurred

2. getServletConfig

public abstract ServletConfig getServletConfig()

Returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet. This is the ServletConfig object passed to the init method; the init method should have stored this object so that this method could return it.

3. service

public abstract void service(ServletRequest req,
ServletResponse res) throws ServletException, IOException

Carries out a single request from the client. The method implements a request and response paradigm. The request object contains information about the service request, including parameters provided by the client. The response object is used to return information to the client. The request and response objects rely on the underlying network transport for quality of service guarantees, such as reordering, duplication, privacy, and authentication.

Service requests are not handled until servlet initialization has completed. Any requests for service that are received during initialization block until it is complete. Note that servlets typically run inside multi-threaded servers; servers can handle multiple service requests simultaneously. It is the servlet writer's responsibility to synchronize access to any shared resources, such as network connections or the servlet's class and instance variables.

Parameters:
req - the client's request of the servlet
res - the servlet's response to the client
Throws: ServletException
if a servlet exception has occurred
Throws: IOException
if an I/O exception has occurred

4. getServletInfo

public abstract String getServletInfo()

Returns a string containing information about the servlet, such as its author, version, and copyright. As this method may be called to display such information in an administrative tool that is servlet engine specfic, the string that this method returns should be plain text and not contain markup.

Returns:
String containing servlet information

5. destroy

public abstract void destroy()

Cleans up whatever resources are being held (e.g., memory, file handles, threads) and makes sure that any persistent state is synchronized with the servlet's current

in-memory state. The method is called once, automatically, by the network service when it unloads the servlet. After destroy is run, it cannot be called again until the network service reloads the servlet.

When the network service removes a servlet, it calls destroy after all service calls have been completed, or a service-specific number of seconds have passed, whichever comes first. In the case of long-running operations, there could be other threads running service requests when destroy is called. The servlet writer is responsible for making sure that any threads still in the service method complete.

Interface javax.servlet.ServletConfig

public interface **ServletConfig**

This interface is implemented by services in order to pass configuration information to a servlet when it is first loaded. A service writer implementing this interface must write methods for the servlet to use to get its initialization parameters and the context in which it is running.

Methods

1. **getServletContext**
public abstract ServletContext getServletContext()
Returns the context for the servlet.
2. **getInitParameter**
public abstract String getInitParameter(String name)
Returns a string containing the value of the named initialization parameter of the servlet, or null if the parameter does not exist. Init parameters have a single string value; it is the responsibility of the servlet writer to interpret the string.
Parameters:
name - the name of the parameter whose value is requested
3. **getInitParameterNames**
public abstract Enumeration getInitParameterNames()
Returns the names of the servlet's initialization parameters as an enumeration of strings, or an empty enumeration if there are no initialization parameters.

Interface javax.servlet.ServletContext

public interface **ServletContext**

The ServletContext interface gives servlets access to information about their environment, and allows them to log significant events. Servlet writers decide what data to log. The interface is implemented by services, and used by servlets.

In a server that supports the concept of multiple hosts (and even virtual hosts), the context must be at least as unique as the host. Servlet engines may also provide context objects that are unique to a group of servlets and which is tied to a specific portion of the

URL path namespace of the host. This grouping may be administratively assigned or defined by deployment information.

Servlets get the ServletContext object via the getServletContext method of ServletConfig. The ServletConfig object is provided to the servlet at initialization, and is accessible via the servlet's getServletConfig method.

Interface javax.servlet.ServletRequest

public interface **ServletRequest**

This interface is for getting data from the client to the servlet for a service request. Network service developers implement the ServletRequest interface. The methods are then used by servlets when the service method is executed; the ServletRequest object is passed as an argument to the service method.

Some of the data provided by the ServletRequest object includes parameter names and values, attributes, and an input stream. Subclasses of ServletRequest can provide additional protocol-specific data. For example, HTTP data is provided by the interface HttpServletRequest, which extends ServletRequest. This framework provides the servlet's only access to this data.

Interface javax.servlet.ServletResponse

public interface **ServletResponse**

Interface for sending MIME data from the servlet's service method to the client. Network service developers implement this interface; its methods are then used by servlets when the service method is run, to return data to clients. The ServletResponse object is passed as an argument to the service method.

To write MIME bodies which consist of binary data, use the output stream returned by *getOutputStream*. To write MIME bodies consisting of text data, use the writer returned by *getWriter*. If you need to mix binary and text data, for example because you're creating a multipart response, use the output stream to write the multipart headers, and use that to build your own text bodies.

Interface javax.servlet.SingleThreadModel

public interface **SingleThreadModel**

Defines a "single" thread model for servlet execution. This empty interface allows servlet implementers to specify how the system should handle concurrent calls to the same servlet.

If the target servlet is flagged with this interface, the servlet programmer is **guaranteed** that no two threads will execute concurrently the service method of that servlet. This guarantee is ensured by maintaining a pool of servlet instances for each such servlet, and dispatching each service call to a *free* servlet.

Classes

Class javax.servlet.GenericServlet

public abstract class **GenericServlet** implements Servlet, ServletConfig

The GenericServlet class implements the Servlet interface and, for convenience, the ServletConfig interface. Servlet developers typically subclass GenericServlet, or its descendent HttpServlet, unless the servlet needs another class as a parent. (If a servlet does need to subclass another class, the servlet must implement the Servlet interface directly. This would be necessary when, for example, RMI or CORBA objects act as servlets.)

The GenericServlet class was created to make writing servlets easier. It provides simple versions of the life-cycle methods init and destroy, and of the methods in the ServletConfig interface. It also provides a log method, from the ServletContext interface. The servlet writer must override only the service method, which is abstract. Though not required, the servlet implementer should also override the getServletInfo method, and will want to specialize the init and destroy methods if expensive servlet-wide resources are to be managed.

package javax.servlet.http

Interface javax.servlet.http.HttpServletRequest

public interface **HttpServletRequest**

An HTTP servlet request. This interface gets data from the client to the servlet for use in the HttpServlet.service method. It allows the HTTP-protocol specified header information to be accessed from the service method. This interface is implemented by network-service developers for use within servlets.

Interface javax.servlet.http.HttpServletResponse

public interface **HttpServletResponse**

An HTTP servlet response. This interface allows a servlet's service method to manipulate HTTP-protocol specified header information and return data to its client. It is implemented by network service developers for use within servlets.