Introduction to Servlets

Servlets provide a component-based, platform-independent method for building Web-based applications. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

What are Servlets?

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

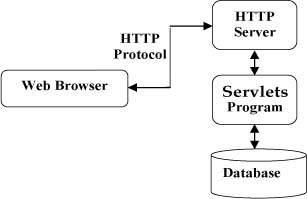
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

* Performance is significantly better.
* Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
* Servlets are platform-independent because they are written in Java.
* Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
* The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture:

Following diagram shows the position of Servelts in a Web Application.



Servlets Tasks:

Servlets perform the following major tasks:

* Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
* Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
* Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
* Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
* Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

## Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http**packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications.

# Servlets - Environment Setup

A development environment is where you would develop your Servlet, test them and finally run them.

Like any other Java program, you need to compile a servlet by using the Java compiler **javac** and after compilation the servlet application, it would be deployed in a configured environment to test and run.

This development environment setup involves following steps:

Setting up Java Development Kit

This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up PATH environment variable appropriately.

Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set PATH and JAVA\_HOME environment variables to refer to the directory that contains java and javac, typically java\_install\_dir/bin and java\_install\_dir respectively.

If you are running Windows and installed the SDK in C:\jdk1.5.0\_20, you would put the following line in your C:\autoexec.bat file.

set PATH=C:\jdk1.5.0\_20\bin;%PATH%

set JAVA\_HOME=C:\jdk1.5.0\_20

Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.

## Setting up Web Server: Tomcat

A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them.

Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server. Here are the steps to setup Tomcat on your machine:

* Download latest version of Tomcat from <http://tomcat.apache.org/>.
* Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-5.5.29 on windows, or /usr/local/apache-tomcat-5.5.29 on Linux/Unix and create CATALINA\_HOME environment variable pointing to these locations.

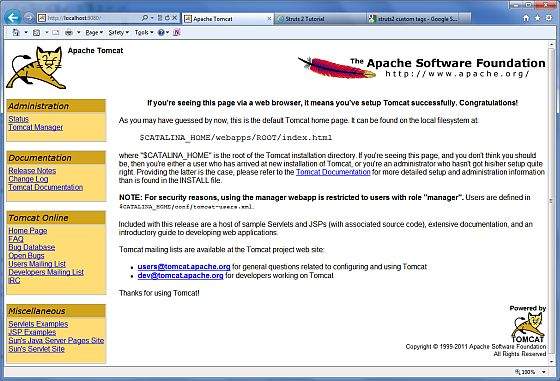
Tomcat can be started by executing the following commands on windows machine:

%CATALINA\_HOME%\bin\startup.bat

or

C:\apache-tomcat-5.5.29\bin\startup.bat

After startup, the default web applications included with Tomcat will be available by visiting **http://localhost:8080/**. If everything is fine then it should display following result:



Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: http://tomcat.apache.org

Tomcat can be stopped by executing the following commands on windows machine:

C:\apache-tomcat-5.5.29\bin\shutdown

## Setting up CLASSPATH

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.

If you are running Windows, you need to put the following lines in your C:\autoexec.bat file.

set CATALINA=C:\apache-tomcat-5.5.29

set CLASSPATH=%CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%

Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the CLASSPATH value and press the OK button.

# Servlets - Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

* The servlet is initialized by calling the **init ()** method.
* The servlet calls **service()** method to process a client's request.
* The servlet is terminated by calling the **destroy()** method.
* Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in details.

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

public void init() throws ServletException {

// Initialization code...

}

The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

public void service(ServletRequest request,

ServletResponse response)

throws ServletException, IOException{

}

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Servlet code

}

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

public void destroy() {

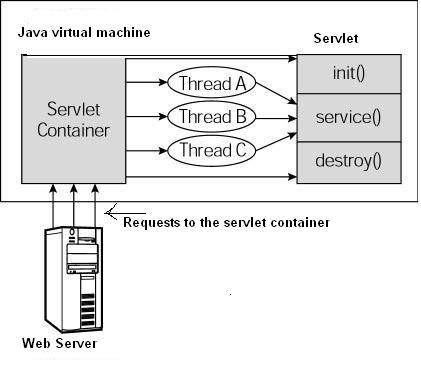
// Finalization code...

}

Architecture Digram:

The following figure depicts a typical servlet life-cycle scenario.

* First the HTTP requests coming to the server are delegated to the servlet container.
* The servlet container loads the servlet before invoking the service() method.
* Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



# Examples :

Servlets are Java classes which service HTTP requests and implement the**javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet.http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code for Hello World:

Following is the sample source code structure of a servlet example to write Hello World:

// Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloWorld extends HttpServlet {

private String message;

public void init() throws ServletException

{

// Do required initialization

message = "Hello World";

}

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

{

// Set response content type

response.setContentType("text/html");

// Actual logic goes here.

PrintWriter out = response.getWriter();

out.println("<h1>" + message + "</h1>");

}

public void destroy()

{

// do nothing.

}

}

Compiling a Servlet:

Let us put above code if HelloWorld.java file and put this file in C:\ServletDevel (Windows) or /usr/ServletDevel (Unix) then you would need to add these directories as well in CLASSPATH.

Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows:

$ javac HelloWorld.java

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable.

If everything goes fine, above compilation would produce **HelloWorld.class** file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment:

By default, a servlet application is located at the path <Tomcat-installation-directory>/webapps/ROOT and the class file would reside in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in**web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

<servlet>

<servlet-name>HelloWorld</servlet-name>

<servlet-class>HelloWorld</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloWorld</servlet-name>

<url-pattern>/HelloWorld</url-pattern>

</servlet-mapping>

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installation-directory>\bin\startup.bat (on windows) or <Tomcat-installation-directory>/bin/startup.sh (on Linux/Solaris etc.) and finally type**http://localhost:8080/HelloWorld** in browser's address box. If everything goes fine, you would get following result:



# Servlets - Form Data

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

## GET method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows:

http://www.test.com/hello?key1=value1&key2=value2

The GET method is the defualt method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limtation: only 1024 characters can be in a request string.

This information is passed using QUERY\_STRING header and will be accessible through QUERY\_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

## POST method:

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

## Reading Form Data using Servlet:

Servlets handles form data parsing automatically using the following methods depending on the situation:

* **getParameter():** You call request.getParameter() method to get the value of a form parameter.
* **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
* **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.