Crossroads
The ACM Student Magazine

An Introduction to Java Servlet Programming

by **Vandana Pursnani**

Introduction

Early web sites were collections of web pages linked together by Hypertext Markup Language (HTML). Today, websites feature multimedia, e-commerce applications, and other sophisticated web-based applications such as voice over IP and online banking. In addition to advances in content, there have been advances in web server technologies such as the development of application servers, and dynamic content creation technologies like Java Server Pages (JSP) and Active Server Pages (ASP).

This article describes Java Servlet technology. It first provides background information about web servers, web containers and application servers. The article next discusses the implementation of the Servlet API in web applications. Finally, the article discusses the advantages of servlets and a brief comparison with CGI (Common Gateway Interface) scripts.

Interaction on the Internet

When you visit a restaurant, a waiter takes your order and passes it to the kitchen staff. When the order is completed the waiter brings the food to you. The activities of the kitchen are invisible to you. The kitchen, like the Web, operates according to a client/server model. Typing a web site address or a URL (Uniform Resource Locator) into a browser, or client, initiates a request to the server hosting the site. Any actions performed by the server are transparent to site visitors. In a typical 2-tier client/server architecture, the web site content (the presentation layer of the web site) like the HTML pages, servlets, JSP, ASP, mail software, CGI scripts, etc. are housed on the web server. Older 2-tier systems, such as in **Figure 1**, were made of a web server or container and the browser.

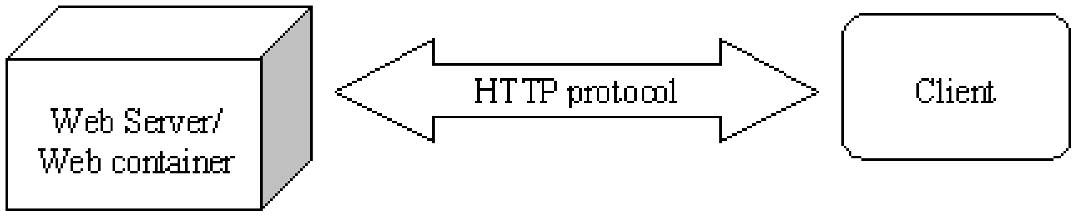

Figure 1: 2-Tier Architecture (Client/Server)

The application server is an extension of the 2-tier client/server model into an N-tier approach [1]. The application server has a built-in web server/web container and an application container. An example of this model is the J2EE application model (Java 2 Enterprise Edition) [12]. This model is implemented in application servers like IBM WebSphere, BEA WebLogic, etc. **Figure 2** demonstrates the N-tier approach.

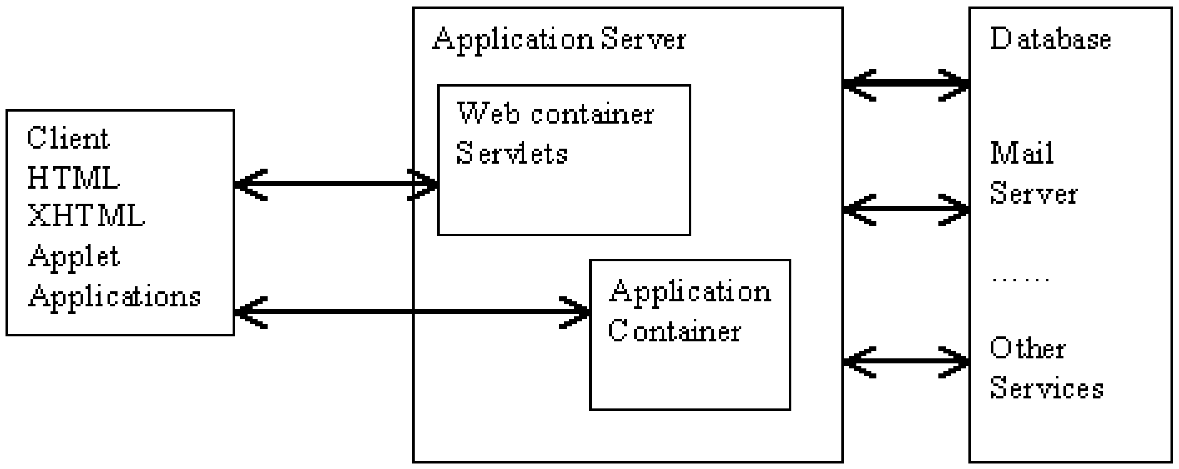

Figure 2: N-tier Architecture

The N-tier approach is ideal for dynamic content generation. Java Servlets, which inherit the features of Java and more, have become a very prominent player in application servers.

What are Java Servlets?

Servlets are server side components.These components can be run on any platform or any server due to the core java technology which is used to implement them. Servlets augment the functionality of a web application.They are dynamically loaded by the server's Java runtime environment when needed. On receiving an incoming request from the client, the web server/container initiates the required servlet. The servlet processes the client request and sends the response back to the server/container, which is routed to the client [1].

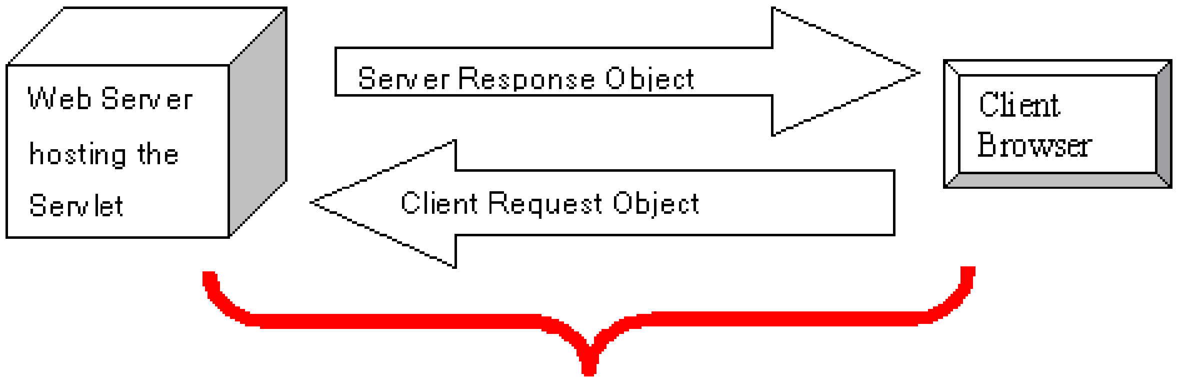

Figure 3: HTTP request response model.

Web based Client/server interaction uses the HTTP (hypertext transfer protocol). HTTP is a stateless protocol based on a request and response model [9] with a small, finite number of request methods like GET, POST, HEAD, OPTIONS, PUT, TRACE, DELETE, CONNECT, etc. The response contains the status of the response and meta information describing the response. Most of the servlet-based web applications are built around the framework of the HTTP request/response model (**Figure 3**).

Requirements to run Java Servlets

1. Java Servlets require some prior working knoweldge of Java and HTML.
2.
3. Web applications and Web pages requiring initiation of servlets must run on web servers with built-in web containers, such as the iPlanet web server [4] or on a standalone servlet container like Tomcat [5]. The servlets may also run on application servers with built-in web containers like BEA WebLogic Server [2].
4.
5. The Servlet API (which we will shortly try to understand) is mostly built in the web container/server. The container/server achieves this by implementing the Java Servlet 2.1 or 2.2 specification [10]. For example, Tomcat 3.3 is compliant to Servlet API 2.2 and JSP API 1.1 specification [6].

Once the basic container/server is downloaded and configured, the next step is to understand servlet programming. The Servlet API has two packages: javax.servlet [7], which has classes and packages independent of protocols, and javax.servlet.http [7], which is more specific to HTTP protocol.

Servlet Structure and Lifecycle

Before going into the servlet lifecycle we need to understand the basic classes and interfaces used for implementing the servlet.

Public Interface Servlet

A servlet must directly or indirectly implement this interface. Like any other java interface this is also a collection of empty method signatures. The following methods are declared in the Servlet interface. [8]

1. `public abstract void init (ServletConfig config) throws ServletException.`

   The init method is used for initializing the servlet parameters provided by the ServletConfig object. It is called only once unless the servlet is reinitialized by destroying and subsequently reloading. The init method is the place to initialize the setup parameters like database connection, file initialization and other environmetal settings. None of the servlets methods can be called unless the servlet is initialized using the init() method.
2. `public abstract ServletConfig getServletConfig ()`

   This method provides the ServletConfig object for initializing the servlet's parameters. Additional parameters can be created by specifying in the servlet.properties file. Once they have been specified in this file they can be accessed using the ServletConfig object.
3. `public abstract void service (ServletRequest req, ServletResponse res) throws ServletException, IOException.`

   The service method is the actual crux of the HTTP request response model. It recieves a client request in the form of ServletRequest object. The client parameters are also passed with the request object (though there are other ways also to send client-side parameters to the servlet, e.g. by using cookies or URL rewriting). The resulting response is sent to the client using the ServletResponse object.
4. `public abstract String getServletInfo()`

   This method is used for extracting the servlet meta data like author, servlet version, and other copyright information. The method will have to be overridden inside the servlet for it to return this information
5. `public abstract void destroy ()`

The destroy method is called in order to clear all retained resources like database and other server resources. It also takes care of synchronization of any pending threads. This method is called only once automatically, like the init method.

The GenericServlet class provides the basic implementation of the Servlet interface. To write a servlet specifically for the HTTP protocol, an HTTPServlet class that extends the GenericServlet class is used (**Figure 4**).

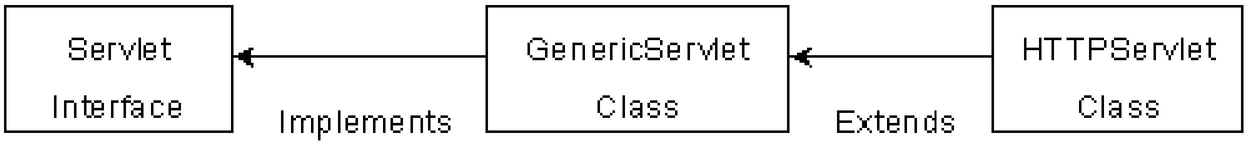

Figure 4: Relevant Servlets Classes and Interfaces

As discussed earlier the lifecycle events for a servlet (**Figure 5**) are specified in the javax. servlet.Servlet interface. All servlets follow the lifecycle model. The web container has the responsibility of creating an instance of the servlet and calling the init method (1). If a client has made a request to the web container then that request is passed to the service method (2) of the servlet and a response (3) is sent back to the web container. Finally when the servlet has finished its purpose then the web container calls the destroy (4) method.

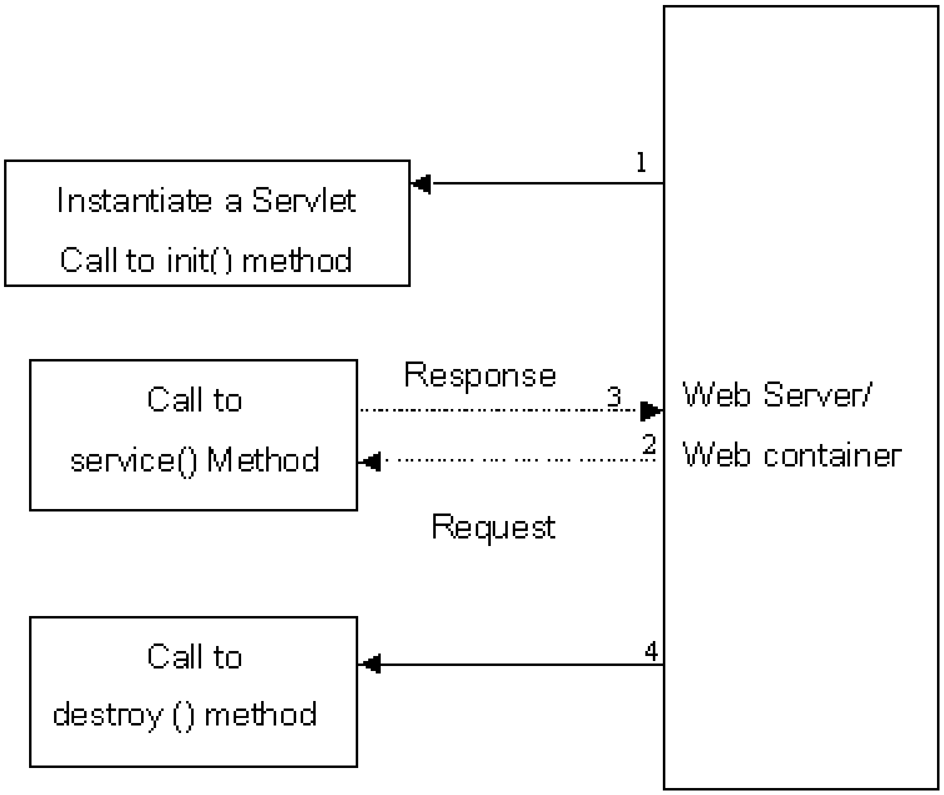

Figure 5: Servlet Lifecycle

Apart from the methods that inherit from the GenericServlet class, the HTTPServlet class has additional methods for each of the HTTP request methods [9] discussed earlier.

1. `doDelete (HttpServletRequest, HttpServletResponse)`
2. `doGet (HttpServletRequest, HttpServletResponse)`
3. `doOptions (HttpServletRequest, HttpServletResponse)`
4. `doPost (HttpServletRequest, HttpServletResponse)`
5. `doPut (HttpServletRequest, HttpServletResponse`
6. `doTrace (HttpServletRequest, HttpServletResponse)`

For example the Servlet class has to override `doGet` or `doPost` (or both methods, depending on whether the data is being sent by GET or by POST HTTP request methods). These methods take two arguments: an `HttpServletRequest` and an `HttpServletResponse` object. These two objects give full access to all information about the client's request and help to control the output sent to the client as the response to the request. The `doGet` and `doPost` are called by the service method. The service method can directly be used by overriding it.

A Simple Servlet

To demonstrate the lifecycle of the servlet let us take a look at a simple servlet program. This program imports the basic servlet package for creating a servlet namely, javax.servlet. This program will implement the servlet interface. It defines the init method, which takes the parameter as a ServletConfig object for setting the properties of the servlet. It has a main service method which takes the request and response objects as parameters. The entire processing of the servlet revolves around these objects. The request from the client is enclosed in the request object and the response from the servlet is provided in the response object. This servlet just provides a simple response of printing a sentence as an html output.

```
import java.io.*;
import javax.servlet.*;
public SimpleServlet implements Servlet
{
private ServletConfig config;
public void init (ServletConfig config) throws ServletException
{
          this.config = config;
          // config object provides initialization parameters for the servlet.
}
public void destroy() { // can do some ending tasks here }
public void service (ServletRequest req, ServletResponse res) throws
ServletException, IOException
{
          res.setContentType( "text/html" ); // setting MIME type

          PrintWriter out = res.getWriter();

          // output stream for writing data on the client side
          out.println (<html>" );
          out.println( "<head>" );  //Generating Client Side Code
          out.println( "<title>FIRST SERVLET</title>" );
          out.println( "</head>" );
          out.println( "<body>" );
          out.println( "<h1>This is my First Servlet</h1>" );
          out.println( "</body>" );
          out.println( "</html>" );
          out.close();
}

} // end of class
```

Once the servlet is compiled, it needs to be installed in the web container in its proper location or a default servlet directory. This varies from server to server; for iPlanet web servers [_

state information of the process is to be saved for coming back to this process later on. A process is called a heavyweight thread since it initiates a complete process with a huge amount of overhead in terms of time, memory, etc. As opposed to this for each request to a servlet, a lightweight thread is spawned to handle that request. A lightweight thread is a child process, which is controlled by the parent process (in this case the server). In such a scenario the context switch becomes very easy and threads can swap easily from active to inactive or waiting. This dramatically increases the performance of servlets over CGI scripts.

For performance tuning it is possible with servlets to cache common data (such as database queries) in memory to avoid unnecessary and costly database lookups. There is no easy way to implement this in CGI. Servlets also have very powerful exception handling.

However, for certain scenarios creating servlets would not necessarily be required. It might be a good investment only if the web site receives a lot of hits; for less visited web sites the same task can be performed using CGI scripts.

Conclusion and Future Work

This article attempted to explore the Java Servlet technology. The advances in web server and application server technology have simplified server side programming. Components like Java Servlets, JSP, CORBA, and XML, are making the J2EE architecture a beckoning field of study.

References

1
Allamaraju, S. et al. *Professional Java Server Programming J2EE Edition.* Wrox Press, Inc., 2000.

2
BEA Systems. *BEA WebLogic Server.* **http://www.bea.com/products/weblogic/server/index.shtml**.

3
Brown, R.G. and Hahn, W. Choices in server-side programming: a comparative programming exercise. In *Proceedings of the conference on On track to the 21st century.* International Conference on APL, August 10 - 14, 1999, Scranton, Penn. **http://www.acm.org/pubs/articles/proceedings/apl/312627/p35-brown/p35-brown.pdf**.

4
iPlanet E-Commerce Solutions. *Product Map.* **http://www.iplanet.com/products/product_map/product_name_2_0a.html**.

5
The Jakarta Project. *Jakarta Tomcat.* **http://jakarta.apache.org/tomcat**

6
The Jakarta Project. *Tomcat User's Guide.* **http://jakarta.apache.org/tomcat/tomcat-3.3-doc/tomcat-ug.html**.

7
Sun Microsystems. *Hierarchy For Package javax.servlet.* **http://java.sun.com/j2ee/tutorial/api/javax/servlet/package-tree.html**.

8
Sun Microsystems. *Interface javax.servlet.Servlet.* **http://java.sun.com/products/servlet/2.1/api/javax.servlet.Servlet.html**.

9
Sun Microsystems. *The J2EE™ Tutorial.* **http://java.sun.com/j2ee/tutorial/doc/HTTP.html**.

10
Sun Microsystems. *Java™ Servlet Technology: Implementations and Specifications.* **http://java.sun.com/products/servlet/download.html**.

11
Sun Microsystems. *JavaServer Technologies, Part I.* **http://developer.java.sun.com/developer/technicalArticles/Servlets/JavaServerTech1/index.html**.

12
Sun Microsystems. *A New Model for Enterprise Applications.* **http://java.sun.com/j2ee/overview2.html**.

Biography

Vandana Pursnani (**pursnani@acm.org**) holds an M.S. in computer management. She is currently a senior software engineer for SiteGain, Inc., and completing an M.S. degree in computer science from the New Jersey Institute of Technology. Her technical areas of interest include the visualization of network and web site data, web site usage patterns, the effect of usage data on e-commerce applications, and J2EE technology.