Java Servlets

Akanksha, Akansha Rakheja and Ajay Singh

ABSTACT: The rise of server-side Java applications is one of the latest and most exciting trends in Java programming. Java servlets are a key component of server-side Java development. A servlet is a small, pluggable extension to a server that enhances the server's functionality. Servlets allow developers to extend and customize any Java-enabled server—a web server, a mail server, an application server, or any custom server—with a hitherto unknown degree of portability, flexibility, and ease. This paper also includes the life cycle of servlet. The applet –servlet interaction is described by two methods, Http Connection and Socket Connection. Lastly it includes various methods for server debugging.

Keywords: HTTPServlets, Life cycle, Applet-Servlet Communication, Socket Communication

I. INTRODUCTION

The servlet is a Java programming language class used to extend the capabilities of a server. Although servlets can respond to any types of requests, they are commonly used to extend the applications hosted by web servers, so they can be thought of as Java Applets that run on servers instead of in web browsers.[1] These kinds of servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET. Technically speaking, a "servlet" is a Java class in Java EE that conforms to the Java Servlet API,[2] a standard for implementing Java classes which respond to requests. Servlets could in principle communicate over any client–server protocol, but they are most often used with the HTTP protocol. Thus "servlet" is often used as shorthand for "HTTP servlet".[3] Thus, a software developer may use a servlet to add dynamic content to a web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets can maintain state in session variables across many server transactions by using HTTP cookies, or URL rewriting. Servlets are most often used to

- Process or store data that was submitted from an HTML form
- Provide dynamic content such as the results of a database query
- Manage state information that does not exist in the stateless HTTP protocol, such as filling the articles into the shopping cart of the appropriate customer

To deploy and run a servlet, a web container must be used. A web container (also known as a servlet container) is essentially the component of a web server that interacts with the servlets. The web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights. The Servlet API, contained in the Java package hierarchy "javax.servlet", defines the expected interactions of the web container and a servlet.[3]

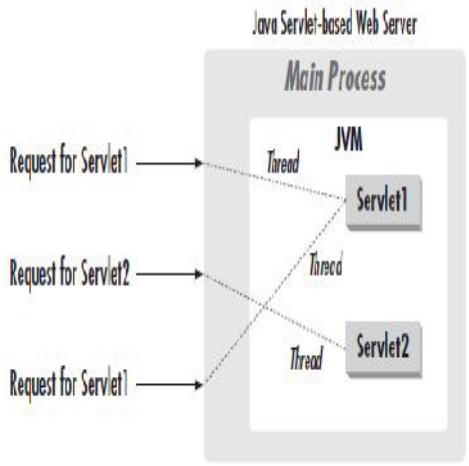
A Servlet is an object that receives a request and generates a response based on that request. The basic Servlet package defines Java objects to represent servlet requests and responses, as well as objects to reflect the servlet's configuration parameters and execution environment. The package "javax.servlet.http" defines HTTP-specific subclasses of the generic servlet elements, including session management objects that track multiple requests and responses between the web server and a client. Servlets may be packaged in a WAR file as a web application.

Servlets can be generated automatically from Java Server Pages (JSP) by the JavaServer Pages compiler. The difference between servlets and JSP is that servlets typically embed HTML inside Java code, while JSPs embed Java code in HTML. While the direct usage of servlets to generate HTML (as shown in the example below) has become rare, the higher level MVC web framework in Java EE (JSF) still explicitly uses the servlet technology for the low level request/response handling via the FacesServlet.

As was said earlier, a servlet is a generic server extension—a Java class that can be loaded dynamically to expand the functionality of a server. Servlets are commonly used with web servers, where they can take the place of CGI scripts. A servlet is similar to a proprietary server extension, except that it runs inside a Java Virtual Machine (JVM) on the server, so it is safe and portable. Servlets operate solely within the domain of the server: unlike applets, they do not require support for Java in the web browser. Unlike CGI and

Akanksha, Akansha Rakheja and Ajay Singh are with *Information Technology (IT), Dronacharya College of Engineering, Gurgaon*

FastCGI, which use multiple processes to handle separate programs and/or separate requests, servlets are all handled by separate threads within the web server process. This means that servlets are also efficient and scalable. Because servlets run within the web server, they can interact very closely with the server to do things that are not possible with CGI scripts. Another advantage of servlets is that they are portable: both across operating systems as we are used to with Java and also across web servers. Although servlets are most commonly used as a replacement for CGI scripts on a web server, they can extend any sort of server.



Servlets offer a number of advantages over other approaches:

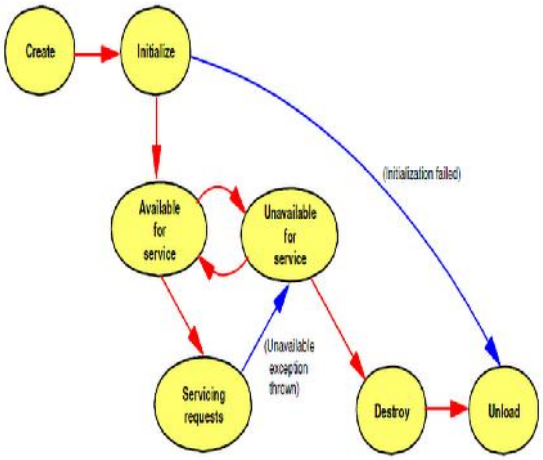
- Portability: Because servlets are written in Java and conform to a well-defined and widely accepted API, they are highly portable across operating systems and across server implementations. You can develop a servlet on a Windows NT machine running the Java Web Server and later deploy it effortlessly on a high-end UNIX server running Apache. With servlets, you can truly "write once, serve everywhere."
- Power: Servlets can harness the full power of the core Java APIs: networking and URL access, multithreading, image manipulation, data compression, database connectivity, internationalization, remote method invocation (RMI), CORBA connectivity, and object serialization, among others. If you want to write a web application that allows employees to query a corporate legacy database, you can take advantage of all of the Java Enterprise APIs in doing so.
- Efficiency and Endurance: Servlet invocation is highly efficient. Once a servlet is loaded, it generally remains in the server's memory as a single object instance. Thereafter, the server invokes the servlet to handle a request using a simple, lightweight method invocation. Servlets, in general, are naturally enduring objects. Because a servlet stays in the server's memory as a single object instance, it automatically maintains its state and can hold on to external resources.
- Safety: Servlets support safe programming practices on a number of levels. Because they are written in Java, servlets inherit the strong type safety of the Java language. In addition, the Servlet API is implemented to be type-safe. Servlets can handle errors safely, due to Java's exception-handling mechanism.
- Integration: Servlets are tightly integrated with the server. This integration allows a servlet to cooperate with the server in ways that a CGI program cannot.
- Extensibilty and Flexibilty: The Servlet API is designed to be easily extensible. As it stands today, the API includes classes that are optimized for HTTP servlets. Servlets are also quite flexible.

2. LIFE CYCLE OF SERVLETS

The servlet life cycle is one of the most exciting features of Servlets. The servlet life cycle allows servlet engines to address both the performance and resource problems of CGI and the security concerns of low-level server API programming. A servlet engine may execute all its servlets in a single Java virtual machine (JVM). Because they are in the same JVM, servlets can efficiently share data with each other, yet they are prevented by the Java language from accessing one another's private data. Servlets may also be allowed to persist between requests as object instances, taking up far less memory than full-fledged processes. Servlet life cycle is highly flexible. Servers have significant leeway in how they choose to support servlets. The only hard and fast rule is that a servlet engine must conform to the following life cycle contract:

1. Create and initialize the servlet.
2. Handle zero or more service calls from clients.
3. Destroy the servlet and then garbage collect it.

A Java servlet has a lifecycle that defines how the servlet is loaded and initialized, how it receives and responds to requests, and how it is taken out of service. In code, the servlet lifecycle is defined by the javax.servlet.Servlet interface. All Java servlets must, either directly or indirectly, implement the
javax.servlet.Servlet interface so that they can run in a servlet engine. The servlet engine is a customized extension to a Web server for processing servlets, built in conformance with the Java Servlet API by the Web server vendor. The servlet engine provides network services, understands MIME requests, and runs servlet
containers. The javax.servlet.Servlet interface defines methods that are called at specific
times and in a specific order during the servlet lifecycle.



2.1 Understanding the Life Cycle

This section describes in detail some of the important servlet life-cycle methods of the Java Servlet API.

2.1.1 Servlet Initialization: init method

Servlets can be dynamically loaded and instantiated when their services are first requested, or the Web server can be configured so that specific servlets are loaded and instantiated when the Web server initializes. In either case, the *init* method of the servlet performs any necessary servlet initialization, and is guaranteed to be called once for each servlet instance, before any requests to the servlet are handled. An example of a task which may be performed in the *init* method is the loading of default data parameters or database connections. The most common form of the *init* method of the servlet accepts a *ServletConfig* object parameter. This interface object allows the servlet to access name/value pairs of initialization parameters that are specific to that servlet. The *ServletConfig* object also gives us access to the *SevletContext* object that describes information about our servlet environment. Each of these objects will be discussed in more detail in the servlet examples sections.

2.1.2 Servlet Request Handling

Once the servlet has been properly initialized, it may handle requests (although it is possible that a loaded servlet may get no requests). Each request is represented by a *ServletRequest* object, and the corresponding
response by a *ServletResponse* object in the Java Servlet API. Since we will be dealing with *HttpServlets*, we will deal exclusively with the more specialized *HttpServletRequest* and *HttpServletResponse* objects.
The *HttpServletRequest* object encapsulates information about the client request, including information about the client's environment and any data that may have been sent from the client to the servlet.
The *HttpServletRequest* class contains methods for extracting this information from the request object.
The *HttpServletResponse* is often the dynamically generated response, for instance, an HTML page which is sent back to the client. It is often built with data from the *HttpServletRequest* object. In addition to an HTML page, a response object may also be an HTTP error response, or a redirection to another URL, servlet, or JavaServer Page. The redirection techniques will be discussed in more detail in the servlet interaction section of this chapter.

Each time a client request is made, a new servlet thread is spawned which services the request. In this way, the server can handle multiple concurrent requests to the same servlet. For each request, usually the *service, doGet,* or *doPost* methods will be called. These methods are passed the *HttpServletRequest* and *HttpServletResponse* parameter objects.

- *doPost*: Invoked whenever an HTTP POST request is issued through an HTML form. The parameters associated with the POST request are communicated from the browser to the server as a separate HTTP request. The *doPost* method should be used whenever modifications on the server will take place.
- *doGet*: Invoked whenever an HTTP GET method from a URL request is issued, or an HTML form. An HTTP GET method is the default when a URL is specified in a Web browser. In contrast to the *doPost* method, *doGet* should be used when no modifications will be made on the server, or when the parameters are not sensitive data.

2.1.3 Destroy Method

The destroy method is called when the Web server unloads the servlet. A subclass of *HttpServlet* only needs to implement this method if it needs to perform cleanup operations, such as releasing database connections or closing files. The server calls a servlet's destroy() method when the servlet is about to be unloaded. In the destroy() method, a servlet should free any resources it has acquired that will not be garbage collected. The destroy() method also gives a servlet a chance to write out its unsaved cached information or any persistent
information that should be read during the next call to init().
Other methods include:

- *getServletConfig:* The getServletConfig method returns a *ServletConfig* instance that can be used to return the initialization parameters and the *ServletContext* object.
- *getServletInfo:* The getServletInfo method is a method that can provide information about the servlet, such as its author, version, and copyright. This method is generally overwritten to have it return a meaningful value for your application. By default, it returns an empty string.

2.1.4 Service Method

This method is declared Abstract in the basic *GenericServlet* class, and so subclasses, such as *HttpServlet*, must override it. In our subclass of *HttpServlet*, when using this method, we must implement this method according to the signature defined in *HttpServlet*, namely, that it accepts *HttpServletRequest* and *HttpServletResponse* arguments. We do some handling of the response object, which is

responsible for sending our response back to the client. Our response here is a formatted HTML. page, so we first set the response content type to text/html by coding **res.setContentType("text/html")**. Next, we request a *PrintWriter* object to write text to the response by coding **PrintWriter out = res.getWriter()**. We could also have used a *ServletOutputStream* object to write out our response, but *getWriter* gives us more flexibility with Internationalization. In either case, the content type of the response must be set before references to these objects can be made.

2.1.5 How the Servlet gets invoked

We could invoke this servlet with either a GET or POST form action method; the *service* method will execute for either. If we knew something about how this servlet was ultimately to be called, for instance, what the HTML form method was going to be, we could have implemented the above functionality through specific *doGet* or *doPost* methods. The result would be the same. The simplest way to invoke the servlet would be by specifying a URL in the Web browser. A URL forces the Web browser to send the request using GET, similar to the way a standard HTML page is requested. The above servlet could be invoked from the Web browser with the URL:
http://host/servlet/itso.servjsp.servletapi.SimpleHttpServlet
http://host/itsoservjsp/servlet/itso.servjsp.servletapi.SimpleHttpServlet

3. APPLET-SERVLET INTERACTION

An applet is a component that exists and operates in the client tier; the servlet acts as a bridge to the Web application or a standalone component offering services. Applets allow another form of component encapsulation and there are many applets available for business. Whether you use applets, JavaBeans, or Servlets depend on their function and what tier you want these components to operate through. For example, there are applets that set up menus and toolbars, offer a wide range of financial calculators and show market prices. let's think about applets that need to communicate with the server. There are a number of good examples. Take a look at the administration applet that manages the Java Web Server. Think about how it works--it executes on the client, but it configures the server. To do this, the applet and the server need to be in near constant communication. As another example, take a look at one of the popular chat applets. One client says something, and all the rest see it. How does that work? They certainly don't communicate applet to applet. Instead, each applet posts its messages to a central server, and the server takes care of updating the other clients. Finally, imagine an applet that tracks the price of a set of stocks and offers continuous updates. How does the applet know the current stock prices, and, more importantly, how does it know when they change?

3.1 Communication Choices

There are many means of communication, and the two most common are Internet and intranet. An internal network has specific communication channels that it must use and these channels are often proprietary to the network operating system. The Internet offers a wide range of communication protocols. the most common being FTP and HTTP. The method of communication for applets and Servlets are either HTTP or socket.

3.1.1 HTTP Communication

Having an applet make an HTTP connection to a CGI program works well for these reasons:

- It's easy to write.
- It works even for applets running behind a firewall. Most firewalls allow HTTP connections but disallow raw socket connections.
- It allows a Java applet to communicate with a program written in any language. The CGI program doesn't have to be written in Java. It can be in Perl, C, C++, or any other language.
- It works with applets written using JDK 1.0, so it works with all Java-enabled browsers.
- It allows secure communication. An applet can communicate with a secure server using the encrypted HTTPS (HTTP + SSL) protocol.
- The CGI program can be used by browsers as well as applets. In the case of our stock tracker example, the CGI program can do double duty, also acting as the back-end for an HTML form-based stock quote service. This makes it especially convenient for an applet to leverage existing CGI programs.

But the HTTP connection to a CGI program also has some problems:

- It's slow. Because of the HTTP request/response paradigm, the applet and the CGI program cannot communicate interactively. They have to reestablish a new communication channel for each request and response. Plus, there is the standard delay while the CGI program launches and initializes itself to handle a request.
- It usually requires requests to be formed as an awkward array of name/value pairs.
- It forces all responses to be formatted using some arbitrary, previously agreed-upon standard.
- Only the applet can initiate communication. The CGI program has to wait passively for the applet to request something before it can respond. If a stock price changes, the applet can find out only when it asks the right question.

3.1.2 Socket Communication

An applet and server can also communicate by having the applet establish a socket connection to a non-HTTP server process. This provides the following advantages over the HTTP-based approach:

- It allows bidirectional, sustained communication. The applet and servlet can use the same socket (or even several sockets) to communicate interactively, sending messages back and forth. For security reasons, the applet must always initiate the connection by connecting to a server socket on the server machine, but after a socket connection has been established, either party can write to the socket at any time. This allows our stock tracker to receive stock price updates as soon as they are available.
- It allows a more efficient program to run on the server side. The non-HTTP server can be written to handle a request immediately without launching an external CGI program to do the work.

But a socket connection also has disadvantages versus the HTTP-based approach:

- It fails for applets running behind firewalls. Most firewalls don't allow raw socket connections, and thus they disallow this sort of applet-server communication. Therefore, this mechanism should be used only when an applet is guaranteed to never run on the far side of a firewall, such as for an intranet application.
- It can be fairly complicated to write the code that runs on the server. There must always be some process (such as a stock quote server) listening on a well-known port on the server machine. Developing such an application in Java is easier than in C++, but it is still nontrivial.
- It may require the development of a custom protocol. The applet and server need to define the protocol they use for the communication. While this protocol may be simpler and more efficient than HTTP, it often has to be specially developed.
- The non-HTTP server cannot be conveniently connected to by a web browser. Browsers speak HTTP; they cannot communicate with a non-HTTP server.

4. SERVLET DEBUGGING

Debugging servlets can be tricky because you don't execute them directly. Instead, you trigger their execution by means of an HTTP request, and they are executed by the Web server. This remote execution makes it difficult to insert break points or to read debugging messages and stack traces. So, approaches to servlet
debugging differ somewhat from those used in general development. Here are some general strategies.

- Use print statements: Insert a couple of print statements to find out which line of code generated the error and which object on that line was null.
- Use an integrated debugger in your IDE: Many integrated development environments (IDEs) have sophisticated debugging tools that can be integrated with your servlet and JSP container.
- Use the log file: The HttpServlet class has a method called log that lets you write information into a logging file on the server. Reading debugging messages from the log file is a bit less convenient than watching them directly from a window as with the two previous approaches, but using the log file is an option even when running on a remote server; in such a situation, print statements are rarely useful and only the advanced IDEs support remote debugging.
- Write separate classes: One of the basic principles of good software design is to put commonly used code into a separate function or class so you don't need to keep rewriting it. That principle is even more important when you are writing servlets, since these separate classes can often be tested independently of the server.
- Plan ahead for missing or malformed data: Every time you process data that comes directly or indirectly from a client, be sure to consider the possibility that it was entered incorrectly or omitted altogether.
- Look at the HTML source: If the result you see in the browser looks odd, choose View Source from the browser's menu.
- Look at the request data separately: Servlets read data from the HTTP request, construct a response, and send it back to the client. If something in the process goes wrong, you want to discover if the cause is that the client is sending the wrong data or that the servlet is processing it incorrectly.
- Stop and restart the server: Servers are supposed to keep servlets in memory between requests, not reload them each time they are executed. However, most servers support a development mode in which servlets are supposed to be automatically reloaded whenever their associated class file changes. At times, however, some servers can get confused, especially when your only change is to a lower-level class, not to the top-level servlet class. So, if it appears that changes you make to your servlets are not reflected in the servlet's behavior, try restarting the server.

5. CONCLUSION

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests 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. Servlet's 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.

REFERENCES

[1] http://en.wikipedia.org/wiki/Java_Servlet

[2] *Java Servlet Programming*, Jason Hunter with William Crawford, published by O'Reilly, ISBN 1-56592-391-X.

[3] http://pdf.moreservlets.com/

[4] http://courses.coreservlets.com/

[5] http://www.redbooks.ibm.com/

[6] http://java.sun.com/products/servlet

[7] Dimitrova, M. (2003). Cognitive modelling and Web search: Some heuristics and insights, Journal "*Cognition, Brain, Behaviour*" Vol. VII, No 3, Pp. 251-258.