Introduction to Servlets, JSP, and Servlet Engines

by Jeff Volkheimer

What are Servlets?

Servlets are the Java Technologies' answer to CGI programming. They are programs which run on the server side and generate dynamic content. Why would one prefer to use Servlets over traditional CGI programming?

- **Efficiency** - Using CGI programming, each time an HTTP request is received a new process is started, which can result in poor performance and scalability issues. Using Servlets, the Java VM is always running, therefore, starting a Servlet creates a Java thread as opposed to a system process.
- **Power** - Servlets allow you power unknown by traditional CGI. They allow you to do things that would either be very difficult or otherwise impossible because they have access to the entire family of Java APIs. Servlets easily share data and maintain information, making session tracking and other chores a breeze.
- **Security** - Servlets can be run by the Servlet engine in a restrictive sandbox, similar to web browser's sandbox for applets. The helps protect against malicious Servlets.
- **Cost** - There are plenty of "free" or low-cost web servers available for either personal use or low volume traffic. If you have a web server, chances are you can add Servlet technology quickly, easily, and best of all, cheaply.
- **Portability** - The Servlet API takes advantage of the Java platform. It is a fairly simple API which is supported by nearly all web servers so that Servlets may be moved from platform to platform, usually without any modification whatsoever.

A Servlet is a Java class and thus needs to be executed by the Java VM, called a Servlet engine. Servlets are loaded by the engine when they are called and remain running until the servlet is explicitly unloaded or the engine is shut down.

The Java Servlet Development Kit (JSDK) is available for download from the Sun Site. It includes the Servlet API and a simple Servlet engine.

What is JSP?

JSP is a extension of Servlets technology. Anything that is done in JSP can be done with Servlets, however, JSP allows you to easily mix static HTML with your code. Typically, it is also easier to read the code and visualize the page that will ultimately be generated. For instance:

JSP

```
<HTML>
   <HEAD>
      <TITLE>Hello World!!</TITLE>
   </HEAD>
   <BODY>
      Hello World! Your name is:<% out.println(response.getParameter("name"));%>
   </BODY>
</HTML>
```

Servlet

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

public class HelloWorld extends HttpServlet
{

      public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException,
```

```
        ServletException
        {

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                out.println("<HTML>");
                out.println("<HEAD>");
                out.println("<TITLE>Hello World!!</TITLE>");
                out.println("</HEAD>");
                out.println("<BODY>");
                out.println("Hello World! Your name is: "
                + response.getParameter("name"));
                out.println("</BODY>");
                out.println("</HTML>");


        }


}
```

In a nutshell, the JSP page is being converted to a normal servlet, with static data being written to an output stream. There are ways to reduce the actual amount of code written in the Servlet, but as you can see, even though they both generate the same output, JSP is easier to read and easier to write. Clearly, JSP and Servlets have their own distinct roles and uses which allow the developer freedom and ease of use.

So, how do I set things up?

There are a number of different Servlet engines available to supplement whatever server you are currently using. A list of commonly used engines and other addons is available here.

Having tried several different engines, I suggest that you try Allaire's JRun. JRun works by intercepting all requests for servlets, JSP, etc., executes them, and returns the response through the Web server to the client. JRun is much more sophisticated than Sun's Servlet engine and supports more features.

JRun supports:

- Enterprise Java Beans 1.1
- Java Transaction API 1.0
- Java Messaging Service 1.0
- Java Server Pages 1.1
- Java Servlets 2.2

JRun is easy to set up and easy to configure. Another nice feature is the ability to set JDBC data sources so that you no longer have to hard code database information into your Servlets or other applications. **Installation**

Here we will briefly outline the installation of JRun. Many of the following steps are common but may vary from engine to engine.

First off, make sure that you have JDK 1.2.2 or later installed. Next, disable your web server and close all other programs. Run the executable.

After beginning the install and progressing through several self-explanatory prompts, you will be asked for a port number to use to connect to the server. It is suggested that you use 8000, however, you may choose any port outside the range 8100 to 8199 (JRun uses a port in this range for its JWS service).

You need to configure your web server to work with JRun. If you are using IIS 4.0 or 5.0, open the Microsoft Management Console. Select WWW service and then click edit. Select the local pathfield and set permissions to Execute (include script). You may set this globally as well by selecting the server, however, this may have security consequences. If you are prompted to change Inheritance Overrides, click OK.

You may now open the JMC (JRun Management Console). To do this, in your browser type //localhost:8000/ . After logging in as the administrator, you can configure each server (admin and default). It is suggested that you configure default first, since this is where applications will be deployed. It is not really ever necessary to configure the admin server to work with IIS. Select the default server and select the appropriate third party web server information. Depending upon your web server, you may have different information to enter.

Test all the demos and see if they work!

If you would like to set up some data sources within JRun, chances are good that you will need a 3rd party driver. Here is a fairly complete list.

Installing your Servlets & JSP pages

Compiling your Servlets, requires you to properly set the CLASSPATH. For example, if you are using JRun in a Windows environment with JDK 1.3, you would open a prompt and do the following:

```
C:\set CLASSPATH=C:\<JRun directory>\lib\ext\servlet.jar; %CLASSPATH%
```

To install your Servlet copy the .java file to C:\<JRun directory>\servers\default\default-app\WEB-INF\classes
Compile your Servlet:

```
C:\...\classes\> javac HelloWorld.java
```

To test your Servlet, type this URL in your browser: //localhost:8100/default-app/servlet/HelloWorld

In order to install .jsp pages, simply copy them to your default-app directory.

For other servers and engines, your necessary steps may vary, however, you will almost always need to set the classpath to servlet.jar and you always need to compile Servlets.

A Simple Example

Let's see the most simple Servlet example to get you started:

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

public class HelloWeb extends HttpServlet
{

	public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
	IOException
	{

		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
		out.println(<HTML><HEAD><TITLE>Hello Web</TITLE> " + "</HEAD><BODY>Hello Web!</BODY></HTML>");

		out.close();


	}

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


		doGet(request, response);


	}


}
```

Let's break down the code and analyze its parts.

First, we import java.io.*, javax.servlet.*, and javax.servlet.http.* because they contain many classes which are commonly used by Servlets. You will almost always need to import these packages. Next, we declare our class, which extends the standard base class for all Servlets. We

also need to override the doGet method of HttpServlet. In the main block of code, we must set the content type. It is required that all header information be set before any PrintWriter or ServletOutputStream is requested to write data to the document. We request a PrintWriter to begin writing data to the response object. Using the PrintWriter object, we write our text. We then close the object.

We have also overridden the doPost method. In this context, it doesn't make too much sense, however, in most circumstances this is something you will want to do. For instance, if you are receiving data using POST data as oppose GET data, this will allow your Servlet to handle both circumstances easily.

Conclusion

Like all technologies, Servlets and JSP have their place. They excel at some things and are deficient in others. It is always important to understand the underlying technology so that one may make an educated choice about which technology to choose. By reading this document you should understand the basics of what is good about JSP and Servlets, how to install and configure a Servlet Engine (JRun), how to compile and install your Servlets and JSP pages, and finally, how to write a basic Servlet.

Developed Under:
Allaire's JRun

References

Sun's Servlet Homepage:
http://java.sun.com/products/servlet/index.html

Sun's JSP Homepage:
http://java.sun.com/products/jsp/index.html

Allaire's JRun Homepage:
http://www.allaire.com/products/jrun/index.cfm

Servlet enabled Web servers and add-on servlet engines:
http://java.sun.com/products/servlet/runners.html

Sun's Java Tutorial site:
http://java.sun.com/docs/books/tutorial/

Not everybody loves JSP and Servlets! Here is an interesting article from another perspective:
http://idm.internet.com/articles/200005/wd_05_05_00a.html

Nice Overview of JSP:
http://developer.netscape.com/viewsource/kuslich_jsp/kuslich_jsp.html

Many code examples:
http://www.javashareware.com/