# Comparing JSPs, ASPs and Servlets

James W. Cooper

In the last couple of columns I introduced Java Server Pages (JSPs) and showed how you can use them to generate interactive web pages. In both cases, you create a standard HTML form web page and send it to a server that parses the data on the form and generates a result. Recently, I had occasion to delve into Active Server Pages (ASPs) and I think it is really instructive to review the differences between these technologies.

In this article, we'll refer to the original HTML page as the *query page* and the page created by the server programs as the *output page*. Before these Java and ASP technologies, you wrote server-side code in C, C++ or perl to receive data from a query web page and generate a result. This was slow to debug and hard to change, because of the compile-test-debug cycle, and the code in those languages was harder to write.

Microsoft's Active Server Pages are a very nice, seductively simple, technology for generating web pages. They were the first of these technologies in the web page market, and have the greatest number of adoptees. On the other hand, as we'll see, they also suffer from begin first, since the later technologies could learn from the limitations of the first design.

## *Comparing Simple Hello Programs*

Java servlets and JSPs have in common a servlet engine that runs in conjunction with or in addition to the web server. If you are using one of the programs termed "application servers," the servlet engine is built in. If you are using Apache or Microsoft IIS, there are any number of add-in servlet engines available. Just check the Javasoft site java.sun.com/products/jsp for links to the most recent list of products.

To write a simple "Hello there" program as a servlet, we start by writing a web page to contact the servlet engine:

```
<html>
<title>Simple Hello There Servlet</title>
<body>
<h1>Simple servlet test</h1>
<form action="/servlet/HiYou" method="POST">
<input type=text name="name" size=20>
<input type=submit value="Submit">
</form>
</body></html>
```

The form action sends the form data to the servlet engine which looks for a servlet called HiYou.class in a special servlet directory tree. It loads and executes this Java program: The program fetches data from the *name* field of the submitted form and generates a simple output page.

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

import java.io.*;
```

```
import java.util.*;
/**
 * Hello there servlet
 */
public class HiYou extends HttpServlet
implements SingleThreadModel {

//------------------------------
    public void init(ServletConfig svg) throws ServletException{
        super.init(svg);
     }
//------------------------------
    public void doPost (HttpServletRequest req,
        HttpServletResponse response)
        throws ServletException, IOException {

            response.setContentType("text/html");
            String name=req.getParameter("name");
            PrintWriter out = response.getWriter();
            out.println("<html>");
            out.println("<body>");
            out.println("<center><h2>Hi ");
            out.println(name);
            out.println("</h2></center>");
            out.println("</body></html>");
      }
//------------------------------
    public void destroy() {
        super.destroy();
    }
}
```

The query page and output page are shown in Figures 1a and 1b.

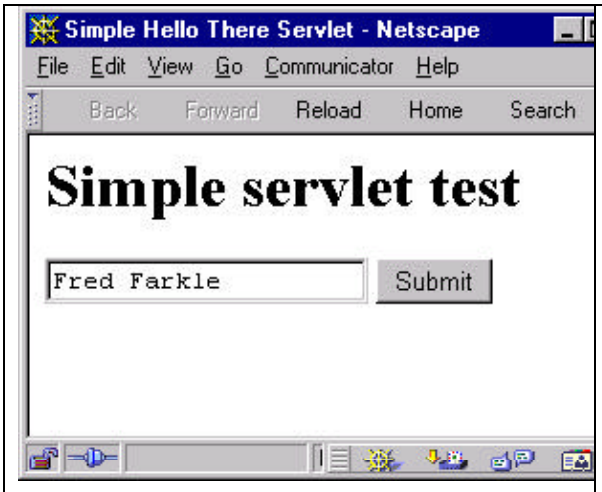


Figure 1a – The servlet query page.

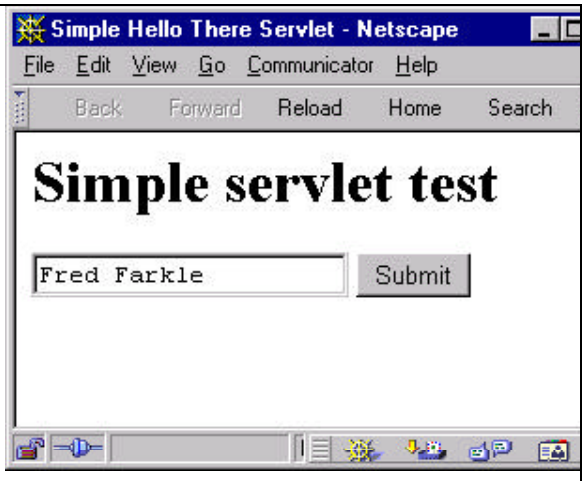


Figure 1b – The servlet output page

So, we see in the servlet case that there is an HTML page on the client which is filled in and send to a Java servlet program on the server.

The JSP approach is only slightly different. The JSP page can be addressed directly from the browser and the output can be created by the same page.

```
<jsp:useBean id="mine" scope = "page" class="mine.JTest" />
<jsp:setProperty name="mine" property="*"/>

<html>
<head><title>Hi</title></head>
<body>

<form method="post">
<input type="text" name="entryText" size="25">
<br>
<input type="submit" value="submit">
</form>

<% if (mine.getEntryText() != null) { %>
        Hello there:
        <%= mine.getEntryText() %>
<% } %>

</body>
</html>
```

You will notice that before the HTML even starts we have a couple of jsp statements that refer to a bean called mine.Jtest, and declare that all properties of the bean can be set by this JSP page. Then, the HTML form code is interspersed with Java fragments enclosed in <% and %> symbols. Note also the convenient shorthand <%= which means to print the value of the expression that follows.

You can even go in and out of Java and HTML as we show here, with an if statement in Java which stops, is followed by some HTML and then closes with an ending bracket inside the language delimiters:

```
<% } %>
```

JSPs always exist in conjunction with server side Java Beans which hold and supply the data between forms. The mine.JTest bean is extremely simple:

```
package mine;
//A simple Java bean for the Hi demonstration programs
public class JTest {
   private String text; //text stored here

   public JTest() {
      text = null;
   }
   //set text here
   public void setEntryText(String t) {
      text = t;
   }
   //get it back here
   public String getEntryText() {
       return text;
   }
}
```

When you launch a JSP, the servlet engine compiles and loads the Java as a servlet and uses it to generate the page you get on the client browser. When you send data back to the

server, it loads thje JavaBean if it is not already loaded and passes the data from the query page to it. Most important, the submitted form automatically calls the set XXX methods of the underlying Bean to store its values there. They are thus persistent across pages and available to any other JSP that references that bean. The JSP then can generate the output page from the data in that bean. You can see the input query and output pages in Figure 2.

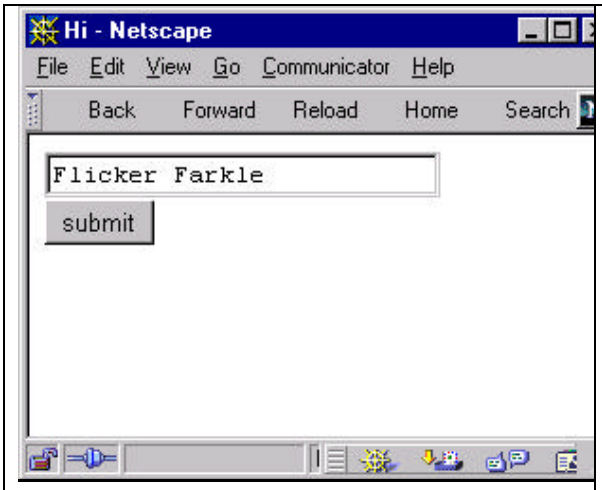


Figure 2a - The JSP Query page


Figure 2b – The JSP output page.

### *Active Server Pages*

ASPs are somewhat different from their Java counterparts, because they do not require a connection to some back end server program or JavaBean. All the code for an ASP is right there on the page. An analogous "Hello there" query page would be the plain HTML hi1.asp form shown below, with the form action pointing to a second ASP, hiback1.asp.

```
<% @LANGUAGE = VBScript %>

<html><head>
<title>Simple ASP Program</title>
</head>
<h1 align="center">Simple ASP Program</h1>

<form action=hiback1.asp method=POST>
Enter your name:
  <input type="text" name="name" width=20><br>
<p><input type="submit" value="Submit" ></p>
</form>
</body></html>
```

Every query page must point to a response ASP page which parses the query and generates the results. Here the response page is:

```
<% @LANGUAGE = VBScript %>
<%
Option Explicit
```

```
Dim strName
%>

<html>
<head>
<title>Simple ASP Program</title>
</head>

<h1 align="center">Simple ASP Program</h1>
<%
strName = Request.Form("Name")
Response.write "Hi there " & strName
%>
</body></html>
```

Note that like the JSP, the embedded code is enclosed in the same <% and %> delimiters and that it can be interspersed with HTML. The main difference is that there is no underlying bean to store the results of various fields, and that code like

```
strName = Request.Form("Name")
```

fetches the contents of the fields on the query form.

There are some other major differences in ASPs. For example, you can use the VB declaration

```
Option Explicit
```

to require that all ASP variables be declared in a Dim statement before they are used, butr all variables are of a Variant type, and no type checking is performed while ASPs are compiled. This has the advantage that ASP output formatting is pretty much automatica and handled by the ASP engine, but the disadvantage that you can write some pretty wrong code and not find out without some debugging.

### *Using Databases with Query Pages*

You can connect to any JDBC compatible database, including Microsoft Access from Java, using either a servlet or a JavaBean. In either case you write pretty much the same code, sending an SQL statement to a database and generating formatted output results. For example, a method from the grocPrices Bean we developed last month generates and issues a query to find the prices of a particular food at a series of stores:

```
public void setFood(String  myfood) {
  food = myfood;
  String queryText =
  "SELECT DISTINCTROW FoodName, StoreName, Price " +
 "FROM (Food INNER JOIN FoodPrice ON " +
 "Food.FoodKey=FoodPrice.FoodKey) "+
 "INNER JOIN Stores ON FoodPrice.StoreKey = Stores.StoreKey " +
 "WHERE (((Food.FoodName)=\'" + food + "\')) ORDER BY " +
 "FoodPrice.Price;";
  rs = db.Execute(queryText);
  queryDone = true;
}
```

Then a series of calls to the Bean methods can move through the ResultSet and generate a table on the JSP output page:

```
<% if (grocBean.hasData()) { %>
  <h2> Prices of <%= grocBean.getFood() %> </h2>
  <table>
<% } %>

<% while(grocBean.hasMoreElements()) { %>
    <tr>
    <td> <%= grocBean.getStore() %></td>
    <td> <%= grocBean.getPrice() %></td>
    </tr>
<%}%>
</table>
```

By contrast, in an ASP, the SQL and all the database handling has to be right there in the page, rather than being controlled by an associated Bean”

```
<% @LANGUAGE = VBScript %>
<%
Option Explicit
Dim objConn, objRS, strQuery
Dim StrConnection, strFood
Set objConn = Server.CreateObject("ADODB.Connection")
strConnection="DSN=Groceryprices;Database=groceries;UID=sa;PWD=;"
objConn.Open strConnection

strFood = Request.Form("Food")
strQuery="SELECT DISTINCTROW FoodName, StoreName, Price " & _
    "FROM (Food INNER JOIN FoodPrice ON Food.FoodKey =" & _
    "FoodPrice.FoodKey) "  & _
     "INNER JOIN Stores ON FoodPrice.StoreKey = Stores.StoreKey " & _
     "WHERE (((Food.FoodName)='" & strFood & "')) ORDER BY
FoodPrice.Price;"
set objRS = objConn.Execute(strQuery)
%>

<html>
<head><title>Grocery Prices</title></head>
<body bgcolor="c0ffff">

<h1 align="center"><font color="#0000FF">Grocery Prices</font></h1>
<h2>Prices for
<% Response.write strFood %>
</h2>

<table>
<% while not objRS.EOF
Response.write "<tr>"
Response.write "<td>" & objRS("StoreName")& "</td>"
Response.write "<td>" & objRS("Price") & " </td></tr>"
objRS.MoveNext
wend

objRS.close
```

```
objConn.close
Set objRS= Nothing
Set objConn = Nothing
%>

</table>
</body></html>
```

In the ASP above, the query is part of the page's "prelude," and the actual page generation follows the HTML tags in the middle of the page's code.

There are two major differences in these two approaches. The first is that the page generation code and the database access code are all written together in ASPs, but in JSPs, the Bean can be used to hide some of this complexity and make the page generation code somewhat easier to write.

The other significant difference is that the Bean approach allows you to encapsulate and subclass the classes that do the actual database manipulation, while the JSP approach does not allow you to write even any convenient subroutines to simplify the database access process. You can write subroutines in ASPs, but they must be on the same page of code. Thus their applicability is relegated to simplifying repeating segments of code on a single page.

### *How You Use ASPs and JSPs*

Active Server Pages are VB-like and easy to code. However, they basically run only on Windows NT servers. Microsoft has made the ASP specifications available, but the most flexible implementations are probably always going to be tied to Windows NT, whether using IIS or a competing server.

If you want to try writing some ASPs to see how they differ from JSPs, you will need NT Server NT Workstation 4.0 or later. If you are using NT version 4, you will need to install all the available service packs, and then download and install the Windows NT Option Pack CD image as well.

By contrast, you can use JSPs on almost any platform and web server, and can test them with the JSP Development Kit which is free from Sun. A lot of high end application server packages support JSPs as well.

The one place where ASPs have the upper hand is in web site host services. A large number of web hosting companies support ASPs while I could only find one or two that support JSPs.

### *Summary and Conclusions*

ASPs have a head start over JSPs and are heavily used throughout the Internet. However, ASPs have no data persistence across pages, while JSPs can all access persistent data using the same JavaBean. ASPs can't call any methods or subroutine outside the web page itself, while JSPs can call methods in any number of classes.

Both JSPs and ASPs can be cluttered and somewhat hard to read, and nobody could call them object oriented. On the other hand the JavaBean that supports a JSP is completely

object oriented and can support subclassing, factories and design patterns. ASPs can do none of these things.

Finally, you can do all the same things with servlets as you can with JSPs, with the disadvantage that you must maintain templates of the final HTML code you want to generate, while the JSPs themselves contain these templates. Servlets are thus the most flexible solution, but the hardest to use for web page generation.