How I Started Using Java Server Pages

James W. Cooper

I've spent some time recently trying to understand Java Server Pages (JSP) and why and where you use them. They're a pretty clever technology although they do result in some pretty messy pages of code.

You'll remember that we discussed Java servlets in this column in the September, 1999 issue. Servlets are a Java technology to replace cgi-bin client-server programming with a more efficient model. Ordinary cgi-bin programs are launched by the web server when HTML form pages are sent to a particular directory. These cgi-bin programs parse the form information and create a new dynamic HTML response page which they return to the client. Java servlets improve on this by keeping a servlet engine running which catches cgi-bin programs and parses them in Java. The servlet then computes the desired response page and returns it to the client in much the same way.

The problem with both of these approaches is that there is no really good way to construct complex output HTML pages. Usually, programmers rely on some kind of template pages which they copy and fill in with data from a particular cgi-bin query. This is one reason why most response pages are simple looking, and many are downright ugly.

Java Server Pages are one solution to remedy this problem. A JSP contains both the HTML to create the final output page and the Java to fill in the parts that must be computed. It also does the form parsing directly and almost invisibly for you. Java Server Pages are supported by a class of programs called Application Servers. There are quite a list of such packages, some of which add on to existing web servers and others of which perform both the web server and the JSP functions directly. You will find a complete list of current products at java.sun.com/products/jsp. You should be careful in ordering these products as some of them support only back level versions of JSP (0.91 was popular, but quite different) and the syntax of JSPs has changed substantially in the final 1.0 release.

Installing the JSP Development Kit

You will also find the complete JSP development kit there, under the name JSWDK-1.0. If you download this package and install it, you will have a complete working JSP system where you can develop and test JSPs as well as servlets and then deploy them to your actual site. The JSP engine provided in this kit is perfectly adequate for rather small sites, but for larger sites you really need one of the application servers. It is a superset of the servlet engine we discussed earlier, and supports both JSPs and ordinary servlets.

To try out the servlet engine on Windows NT, go to the JSWDK-1.0 directory and run startserver.bat. On Unix machines, run the startserver shell script that is also provided. I personally did all my testing on Windows NT 4.0 with SP5 installed. I wrote my test programs on a laptop that was not connected to any network. I was not able to get the examples to run on the same laptop running Windows-95b (OSR2).

To see the test programs that come with the JSWDK, start your web browser, and enter the URL

http://localhost:8080
This brings up a page describing the available examples and the servlet API. If you click on the JSP Samples link you will get the page shown in Figure 1. You can learn a lot from these examples, by watching them run and by examining the code. However, since the amount of detailed documentation on JSPs remains fairly sparse, we'll start at the beginning with a "Hi" program and work up to some of the concepts they illustrate.

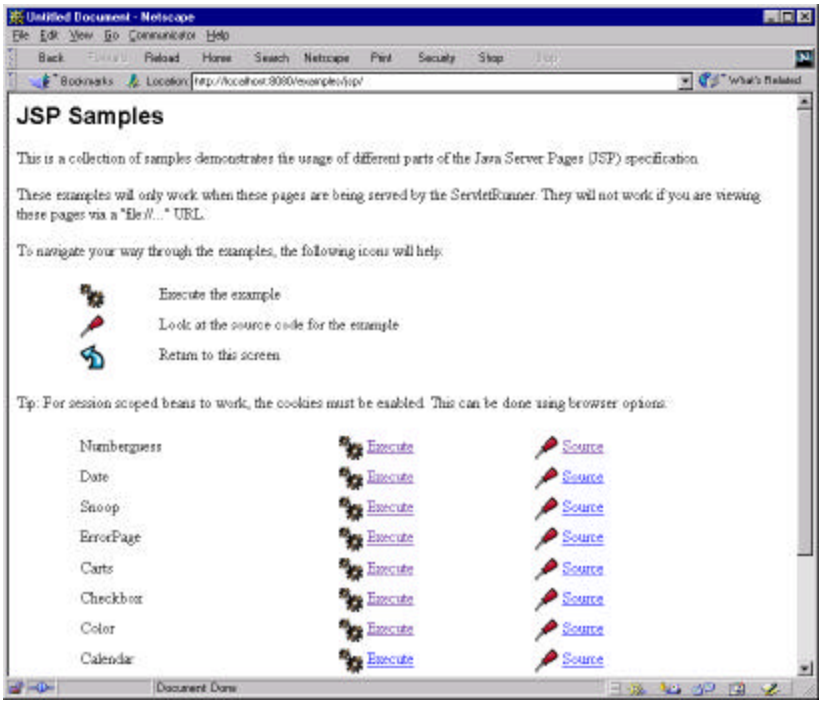

Figure 1 – The JSP samples page.

A Simple Hello World JSP program

A Java Server Page consists of HTML, of JSP directives and of actual Java code. The JSP engine compiles this code and caches it when you first access the page. This makes further accesses to the page much faster. The JSP page is a server-side concept: the actual code that is sent back to the client is pure HTML generated by the code on that page. This makes the client requirements much less demanding than ones that require particular levels of Java support and the like.

Our first simple program, hi.jsp consists of a simple form, some output and communication with a back end Java bean. All JSP pages must start even before the <html> tag with declarations defining the fact that it is a JSP and the Java bean it will communicate with:

```
<%@ page language="java" import="mine.JTest" %>
```
or
```
<jsp:useBean id="mine" scope = "page" class="mine.JTest" />
```

The first line is a jsp directive, describing the language (only Java is currently supported) and the bean the page will communicate with. The second line is the same jsp directive in an alternate format. You only need one or the other, although both do no harm. Note that the longer form follows the XML custom, terminating the tag with a /> to indicate that nothing more follows that relates to that tag.

The Java Bean and the JSP Page

Most JSP pages connect with simple Java beans which contain and return the contents of the fields on the page. The beans that JSPs use are invisible server-side beans which just havd get and set methods. They have no visual properties.

For example, you might have a <form> section on your page and inside it a text entry field.

```
<input type="text" name="entryText" size="25">
```

The associated Java bean then has get and set methods for this field:

```
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;
   }
}
```

Note that the case is odd but required. The name of the field must start in lower case, but the get and set methods convert it to upper case. When you begin using this bean on a JSP, you must initialize all of its properties in order to access them.

```
<jsp:setProperty name="mine" property="*"/>
```

This line initializes all the bean's properties at once. This statement is required, or your bean properties won't work.

Communicating with the Java Bean

The complete form section of our JSP page looks like this;

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

Note that while there is a post method specified, there is no *action* specified which tells the page where to send the form. Instead, submit just sends all of the form data to the Java bean, executing the set method for each form field. In this case, the JSP calls the bean's *setEntryText* method with the string from the entry field as its argument. Thus, following the submit, the Bean contains all the data in the form's entry fields.

You can access that data using the get methods, and can print it out on the web page using JSP statements:

```
<%
out.println(mine.getEntryText());
%>
```

Note that you can actually embed real Java statements on a JSP page as long as then are enclosed in the <% and %>. You can also use a shorter form, where the equals sign replaces the "out.println" method call:

```
<%= mine.getEntryText() %>
```

This latter form is not a complete Java statement and does not need a terminating semicolon. Since it is so compact, you see it frequently.

Our First Complete JSP

Combining the above fragments, we have a complete page as follows:

```
<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>

Hello there: <%= mine.getEntryText() %>

</body>
</html>
```

This program initializes the *mine* Bean and uses the submit button to load it. Then it prints out the result.

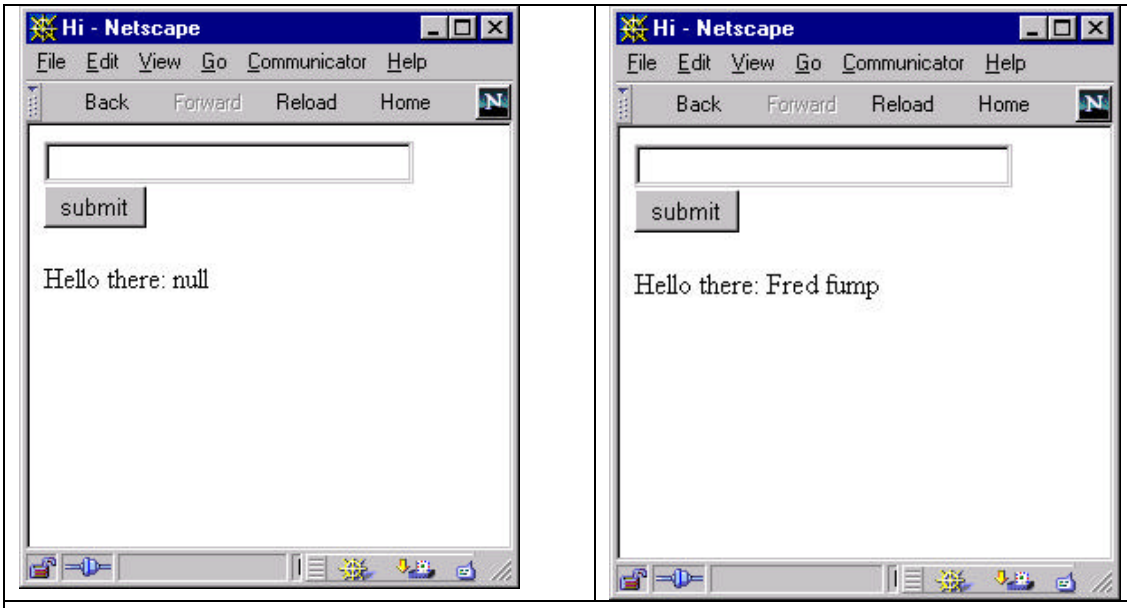

Figure 2 – The simple hi1.jsp program before and after entering a name and pressing submit.

The displays in Figure 2 are not very satisfactory, because we display "Hello there: null" before any name has been submitted. It would be better to display this only when there is real data. The second JSP example accomplishes this by testing for whether the name is non-null before displaying it:

```
<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>
```

In this example we do not see any "Hello there" text or null name displayed. Note how we can interweave Java and HTML in the same page to make decisions as to what to display. This is both the greatest strength and weakness of JSPs. You can write Java code to display data, but interleaving it makes the page look unstructured and confusing.

Now the purpose of JSPs is to keep the HTML and the server side logic together but logically separated to diminish clutter. The final example hi3.jsp just moves the test for printing Hello there and a name to another file which is included.

```
<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>

<%@ include file="myResponse.jsp" %>

</body>
</html>
```

This gives the web designer control over the base jsp file and the server programmer control over the output in the myResponse.jsp, which is much the same code as in hi2.jsp:

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

Where to put all these classes?

If you install the JSP JDK as we described earlier, you will have a tree which contains

`Jswdk-1.0\examples\jsp`

Along with a whole lot of little directories underneath for each of the provided examples. Unzip the jsp code into a new “hi” folder at this level.

The Java bean goes into the hierarchy

`Jswdk-1.0\examples\Web-inf\jsp\beans`

Create a folder called “mine” and unzip the bean into it.

Automated Tools

You can see the power of JSPs compared to other approaches. However, you can also see the clutter and confusion they can generate. Newest versions of several of the Java IDEs contain methods for building JSPs a little more automatically. I’ve seen it in the forthcoming version of Borland’s JBuilder, and I believe IBM’s Visual Age for Java supports it as well.

We’ve seen a lot of the beginnings of the JSP infrastructure in the introductory article, but we’ll have a future column where we go into more details.