Stupid Browser Tricks with JavaServer Pages

James W. Cooper

By now you've seen enough articles about JavaServer Pages (JSPs) to have a pretty good feeling for what you might use them for. They are more powerful than CGI and more portable than ASP, and a lot easier to use than either of these other client server methods. Remember that JavaServer pages usually consist of some page script that runs on the server and performs some computations or makes some decisions and generates pure HTML output. The script usually references some Java classes in a bean on the server and uses them to make the computation.

Usually, you think about using JSPs for looking up data based on some input parameters and displaying that data in the generated browser page. For example, you might display the results of a search engine's search, or a list of products and prices, or the names and addresses of people in a certain category you specify.

But you can use JSPs to control more than the data you display. You can also control how the browser displays them. To see how this can be done, let's review some of the characteristics of JavaScript. Yes, I know that JavaScript isn't Java, but we will end up using Java, too, so trust me on this.

A Bit About JavaScript

As you probably know, JavaScript is an interpreted language which is executed as part of a web page when it is loaded and when you perform certain actions on the form's controls. The most common use of JavaScript is in interacting with HTML form fields. When your page's user fills in some fields and clicks on Submit, you can check the page for errors before you submit it to your server.

Normally, when you submit an HTML form, you do it by specifying the next form to load in the action field of the form

```
<form action= "search.jsp">
<input type=submit name="search" value="Search">
```

If you do not specify a new JSP, the same one gets reloaded with whatever changes the script specifies after the data in the form are processed. A more powerful way to submit pages is to have the button call a JavaScript function which then itself executes the submit, usually after some input validation:

```
<input type=button name="search" value="Search"
            onClick="searchData(this.form)">
```

The corresponding JavaScript function could be as simple as

```
  function showData(form) {
    form.action= "search.jsp";
    form.submit();
 }
```

JavaScript Browser Variations

If you want to make an interesting client page without using Java, you can frequently do it using just JavaScript. For example, you might have a set of floating menu items along the side of your page that you want to highlight when the mouse moves over them. Floating items are placed in a special division or layer that you can position by pixel and even move while the page is displaying.

In Netscape browsers, you position this floating objects inside <layer> tags.

```
<layer name=”ourmenu” left=50 top=100>
Hi there
</layer>
```

Layers are accessible through a layers[] array of the Document object, and the array is in the same order as they appear in the document.

```
Document.layers[0].top = 75;
```

You can also make layers highlight and un-highlight when the mouse moves over them by setting the onmouseover property to a function to be called

```
   var mLayer = new Layer(menuWidth);
   mLayer.onmouseover = hiLight;
   mLayer.onmouseout = unLight;


//--------------------------------
function hiLight() {
   this.bgColor = menuForeColor;
}
//--------------------------------
function unLight() {
   this.bgColor = menuBackColor;
}
```

While layers are not part of any W3C standard, as far as I am aware, they are definitively a part of Netscape browsers. Note that this changes somewhat in Navigator 6, but at this writing the final version has not been released.

In Explorer, you can assign dynamic positions to any elements, and you typically use a <DIV> tag to create a block to be positioned:

```
<div id="ourmenu" style="position:absolute">
Hi there
</div>
document.all.ourmenu.style.pixeltop = 75 ;
```

If you want to have an anchor tag which is not underlined and changes color with mouseover, you set it using style sheet tags:

```
<style>
  a:hover{
      color:#ff00ff;
      font-weight:bold;
      background-color:#FFFF00;
}
```

```
</style>

<style>
 A {
      text-decoration:none
}
</style>
```
This latter text-decoration style declaration works for both Netscape and IE.

How to Support Multiple Browsers

Now, if we want to be able to draw our menus irrespective of which browser we are using, we need to encapsulate these differences. Typically, you write a lot of JavaScript with if blocks in it.

```
if (NetscapeBrowser)
      do this;
else
      do that;
```

A better way is to use JavaServer pages to generate just the page code we need for a particular browser. We start with a splash screen page which figures out what kind of browser we have.

```
<%@ page language="java"
    import="SMenu.MenuBean" %>
<jsp:useBean id="smenu" scope = "session"
    class="SMenu.MenuBean"/>
<jsp:setProperty name="smenu" property="*"/>

<html><head>
<title>Opener Page</title>
<!-- -----------------  -->
<script language ="JAVASCRIPT">
//detects which browser we are using
 function showData(form) {
    form.action= "ShowMenu.jsp";
    var browser = navigator.appName;
    //sets string into hidden input variables
    form.browserType.value = browser;
    form.browserVersion.value =
        navigator.appVersion;
    form.submit();
 }
</script>
</head>
<!-- -----------------  -->
<body>
<center>
<h2>Welcome to the Opener</h2>

<form>
<input type=button name="enter"
    value="Enter site"
    onClick="showData(this.form)">
<INPUT TYPE="HIDDEN" NAME="browserType" valu e="foo">
```

```
<INPUT TYPE="HIDDEN" NAME="browserVersion" value="foo2">
</form>

</center>
</body>
</html>
```

The two hidden input variable fields are set to the browser name and version number when you click on the button labeled “Enter Site.” Then these two values are sent to the JSP bean named MenuBean where the following two methods are called automatically:

```
package SMenu;
public class MenuBean {
    private String browserVersion;
    private String browserType;
    //saves the browser name
//saves the browser name
    public void setBrowserType(
            String btype) {
       browserType = btype;
    }
    public String getBrowserType() {
        return browserType;
    }
    //saves the browser version
    public void setBrowserVersion(
            String bversion) {
       browserVersion = bversion;
    }
    public String getBrowsVersion() {
        return browserVersion;
    }
```

The other thing we do is that we encapsulate the generation of the menu JavaScript code into two files with identical methods but different implementations, called NSMenus.js and IEMenus.js. They each have an addMenu, drawMenu functions and also contain hilight and unhighlight functions. Our main program is simply

```
<script language="JavaScript">
 function setMenus() {
  menuBackColor = "#ffffff";
  menuForeColor = "#FFFF00";
  menuHighColor = "#ffc000";
  menuX = 50;
  menuY = 100;
  menuWidth = 200;
  menuHeight = 20;
  menuBottom = 300;

  var index = 0;
  addMenu("Lab Software",
      index++, "http://labsoftware.com");
  addMenu("Connecticut Swimming",
      index++,"http://ctswim.org");
  addMenu("Amazon",
      index++, "http://amazon.com");
  addMenu("US Postal Service",
      index++, "http://usps.gov");
```

```
  drawMenus();
}
</script>
<style>
 a:hover{color:#ff00ff;
 font-weight:bold;
 background-color:#FFFF00; }
</style>

<style>
A {text-decoration:none}
</style>
<body onLoad="setMenus()">

<h1> Favorite Sites</h1>
</body></html>
```

The critical part, however, is the JSP part that decides which menu code to include. We need to have a statement that gets the correct menu from the JSP Bean:

```
    <script src=
    <% out.println("\"" +
      smenu.getBrowserMenu() +
      "\""); %>
     language="JavaScript">
    </script>
```

The corresponding Java in the JSP bean implements that getBrowserMenu function:

```
public String getBrowserMenu() {
  int i =
      browserType.indexOf ("Netscape");

  if( i >=0 )
    return "NSMenus.js";
  else
    return "IEMenus.js";
}
```

So the generated page code will then contain either

```
<html>
<head>
    <TITLE>Favorite sites</TITLE>
    <script src=
    "NSMenus.js"
    language="JavaScript">
```

or the corresponding statement for IEMenus.js. The menu page is shown in Figure 1.

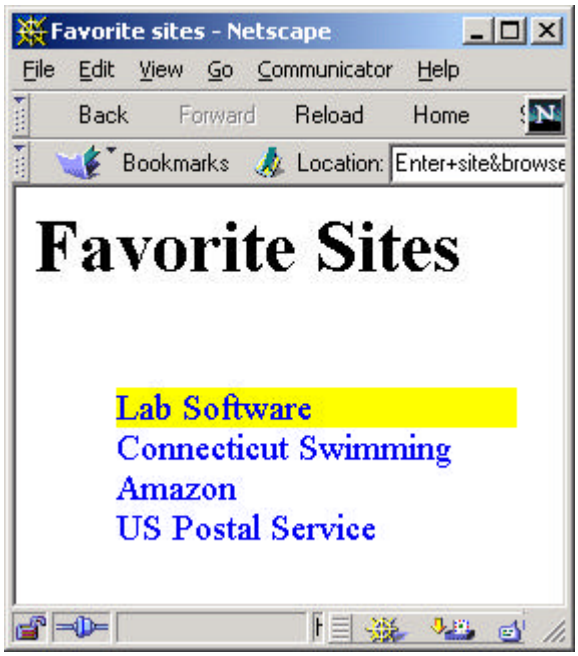

Figure 1 – The menu page in Netscape. The IE version is essentially identical except for minor differences in font size.

You can of course extend this dynamic JavaScript generation to any level of detail, but with the advantage that no page contains JavaScript code for more than one browser. Such code is far easier to write and maintain than code containing all those if tests. In addition, if you need to specify different behavior for Netscape 6, some newer version of IE or the Opera browser, you can add as many new versions as you want without having to tangle up your JavaScript code.

Persistence of Values Between Pages

In our above example, we declared that our JSP connection had session score:

```
<jsp:useBean id="smenu" scope = "session"
    class="SMenu.MenuBean"/>
```

This means that the values for the browserType and browserVersion variables persist throughout the session. Under the covers, the JSP engine manages this persistence using cookies. To prove this, set your browser to warn you about cookies, and you will get the warning shown in Figure 2.

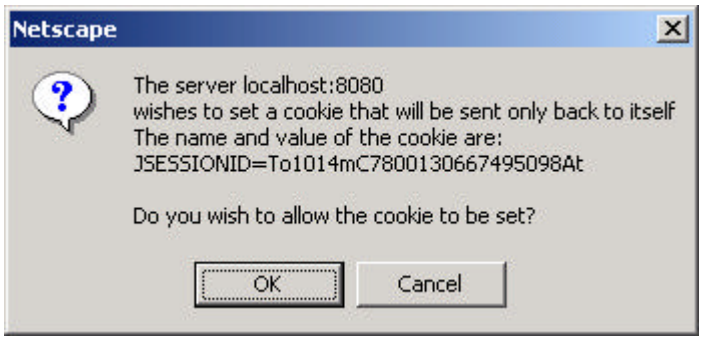

Figure 2 – The Netscape cooking warning message. The IE message is similar.

You do not have to use cookies for this, however. You can include these same two hidden fields on each client page and query the bean for their values (which are set as each page is submitted). We just have to include the JSP code to generate these two hidden fields with the values filled in.

```
<form>
<INPUT TYPE="HIDDEN"
      NAME="browserType" value=
 <% out.println("\"" +
      smenu.getBrowserType() +"\"");
 %>
>
<INPUT TYPE="HIDDEN"
      NAME="browserVersion" value=
 <% out.println("\"" +
      smenu.getBrowserVersion() +"\"");
 %>
>
</form>
```

For Netscape, this gets filled in as:

```
<form>
<INPUT TYPE="HIDDEN"
 NAME="browserType" value=
 "Netscape"
>
<INPUT TYPE="HIDDEN"
  NAME="browserVersion" value=
 "4.74 [en] (Windows NT 5.0; U)"
>
</form>
```

and for Internet Explore the page contains:

```
<form>
<INPUT TYPE="HIDDEN"
 NAME="browserType" value=
 "Microsoft Internet Explorer"
 >
<INPUT TYPE="HIDDEN"
 NAME="browserVersion" value=
 "4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
>
</form>
```

Summary

In essence, we have used JSPs to create a JavaScript factory. The JSP Bean generates just the JavaScript we need for each browser we want to support. In cases where Java is not appropriate for the client page, this gives you a great deal of flexibility in providing JavaScript implementations of your web pages.

References

1. David Flanagan, *JavaScript – The Definitive Guide*, O'Reilly, Sebastopol, 1998.
2. Danny Goodman, *Dynamic HTML – The Definitive Reference*, O'Reilly, Sebastopol, 1998.
3. James Cooper, *Java Design Patterns:A Tutorial*, Addison-Wesley, Boston, 2000.