13.4 Including Applets for the Java Plug-In

Early in the evolution of the Java programming language, the main application area was applets (Java programs embedded in Web pages and executed by Web browsers). Furthermore, most browsers supported the most up-to-date Java version. Now, however, applets are a very small part of the Java world, and the only major browser that supports the Java 2 platform (i.e., JDK 1.2–1.4) is Netscape 6 and later. This leaves applet developers with three choices:

- Develop the applets with JDK 1.1 or even 1.02 (to support *really* old browsers).
- Have users install version 1.4 of the Java Runtime Environment (JRE), then use JDK 1.4 for the applets.
- Have users install any version of the Java 2 Plug-in, then use Java 2 for the applets.

The first option is the one generally chosen for applets that will be deployed to the general public, because that option does not require users to install any special software. You need no special JSP syntax to use this option: just use the normal HTML `APPLET` tag. Just remember that `.class` files for applets need to go in the Web-accessible directories, not `WEB-INF/classes`, because it is the browser, not the server, that executes them. However, the lack of support for the Java 2 Platform imposes several restrictions on these applets:

- To use Swing, you must send the Swing files over the network. This process is time consuming and fails in Internet Explorer 3 and Netscape 3.x and 4.01–4.05 (which only support JDK 1.02), since Swing depends on JDK 1.1.
- You cannot use Java 2D.
- You cannot use the Java 2 collections package.
- Your code runs more slowly, since most compilers for the Java 2 platform are significantly improved over their 1.1 predecessors.

So, developers of complex applets for corporate intranets often choose one of the second two options.

The second option is best if the users all have Internet Explorer 6 (or later) or Netscape 6 (or later). With those browsers, version 1.4 of the JRE will replace the Java Virtual Machine (JVM) that comes bundled with the browser. Again, you need no special JSP syntax to use this option: just use the normal HTML `APPLET` tag. And again, remember that `.class` files for applets need to go in the Web-accessible directories, not `WEB-INF/classes`, because it is the browser, not the server, that executes them.

Core Approach

No matter what approach you use for applets, the applet `.class` files must go in the Web-accessible directories, not in `WEB-INF/classes`. The browser, not the server, uses them.

In large organizations, however, many users have earlier browser versions, and the second

choice is not a viable option. So, to address this problem, Sun developed a browser plug-in for Netscape and Internet Explorer that lets you use the Java 2 platform in a variety of browser versions. This plug-in is available at http://java.sun.com/products/plugin/ and also comes bundled with JDK 1.2.2 and later. Since the plug-in is quite large (several megabytes), it is not reasonable to expect users on the WWW at large to download and install it just to run your applets. On the other hand, it is a reasonable alternative for fast corporate intranets, especially since applets can automatically prompt browsers that lack the plug-in to download it.

Unfortunately, in some browsers, the normal `APPLET` tag will not work with the plug-in, since these older browsers are specifically designed to use only their built-in virtual machine when they see `APPLET`. Instead, you have to use a long and messy `OBJECT` tag for Internet Explorer and an equally long `EMBED` tag for Netscape. Furthermore, since you typically don't know which browser type will be accessing your page, you have to either include both `OBJECT` and `EMBED` (placing the `EMBED` within the `COMMENT` section of `OBJECT`) or identify the browser type at the time of the request and conditionally build the right tag. This process is straightforward but tedious and time consuming.

The `jsp:plugin` element instructs the server to build a tag appropriate for applets that use the plug-in. This element does not add any Java capabilities to the client. How could it? JSP runs entirely on the server; the client knows nothing about JSP. The `jsp:plugin` element merely simplifies the generation of the `OBJECT` or `EMBED` tags.

Core Note

The `jsp:plugin` element does not add any Java capability to the browser. It merely simplifies the creation of the cumbersome `OBJECT` and `EMBED` tags needed by the Java 2 Plug-in.

Servers are permitted some leeway in exactly how they implement `jsp:plugin` but most simply include both `OBJECT` and `EMBED`. To see exactly how your server translates `jsp:plugin`, insert into a page a simple `jsp:plugin` element with `type`, `code`, `width`, and `height` attributes as in the following example. Then, access the page from your browser and view the HTML source. For example, Listing 13.7 shows the HTML code generated by Tomcat for the following `jsp:plugin` element.

```
<jsp:plugin type="applet"
            code="SomeApplet.class"
            width="300" height="200">
</jsp:plugin>
```

Listing 13.7 Code Generated by Tomcat for `jsp:plugin`

[View full width]

```
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        width="300" height="200"
        codebase="http://java.sun.com/products/plugin/1.2.2/jinstall-1_2_2-win
➥ .cab#Version=1,2,2,0">
  <param name="java_code" value="SomeApplet.class">
  <param name="type" value="application/x-java-applet;">
<COMMENT>
<embed type="application/x-java-applet;" width="300" height="200"
       pluginspage="http://java.sun.com/products/plugin/"
       java_code="SomeApplet.class"
>
<noembed>
```

```
</COMMENT>
</noembed></embed>
</object>
```

The jsp:plugin Element

The simplest way to use `jsp:plugin` is to supply four attributes: `type`, `code`, `width`, and `height`. You supply a value of `applet` for the `type` attribute and use the other three attributes in exactly the same way as with the `APPLET` element, with two exceptions: the attribute names are case sensitive, and single or double quotes are always required around the attribute values. So, for example, you could replace

```
<APPLET CODE="MyApplet.class"
        WIDTH=475 HEIGHT=350>
</APPLET>
```

with

```
<jsp:plugin type="applet"
            code="MyApplet.class"
            width="475" height="350">
</jsp:plugin>
```

The `jsp:plugin` element has a number of other optional attributes. Most parallel the attributes of the `APPLET` element. Here is a full list.

- **`type`**

  For applets, this attribute should have a value of `applet`. However, the Java Plug-in also permits you to embed JavaBeans components in Web pages. Use a value of `bean` in such a case.

- **`code`**

  This attribute is used identically to the `CODE` attribute of `APPLET`, specifying the top-level applet class file that extends `Applet` or `JApplet`.

- **`width`**

  This attribute is used identically to the `WIDTH` attribute of `APPLET`, specifying the width in pixels to be reserved for the applet.

- **`height`**

  This attribute is used identically to the `HEIGHT` attribute of `APPLET`, specifying the height in pixels to be reserved for the applet.

- **`codebase`**

  This attribute is used identically to the `CODEBASE` attribute of `APPLET`, specifying the base directory for the applets. The `code` attribute is interpreted relative to this directory. As with the `APPLET` element, if you omit this attribute, the directory of the current page is used as the default. In the case of JSP, this default location is the directory in which the original JSP file resided, not the system-specific location of the servlet that results from the JSP file.

- **align**

  This attribute is used identically to the ALIGN attribute of APPLET and IMG, specifying the alignment of the applet within the Web page. Legal values are left, right, top, bottom, and middle.

- **hspace**

  This attribute is used identically to the HSPACE attribute of APPLET, specifying empty space in pixels reserved on the left and right of the applet.

- **vspace**

  This attribute is used identically to the VSPACE attribute of APPLET, specifying empty space in pixels reserved on the top and bottom of the applet.

- **archive**

  This attribute is used identically to the ARCHIVE attribute of APPLET, specifying a JAR file from which classes and images should be loaded.

- **name**

  This attribute is used identically to the NAME attribute of APPLET, specifying a name to use for interapplet communication or for identifying the applet to scripting languages like JavaScript.

- **title**

  This attribute is used identically to the very rarely used TITLE attribute of APPLET (and virtually all other HTML elements in HTML 4.0), specifying a title that could be used for a tool-tip or for indexing.

- **jreversion**

  This attribute identifies the version of the Java Runtime Environment (JRE) that is required. The default is 1.2.

- **iepluginurl**

  This attribute designates a URL from which the plug-in for Internet Explorer can be downloaded. Users who don't already have the plug-in installed will be prompted to download it from this location. The default value will direct the user to the Sun site, but for intranet use you might want to direct the user to a local copy.

- **nspluginurl**

  This attribute designates a URL from which the plug-in for Netscape can be downloaded. The default value will direct the user to the Sun site, but for intranet use you might want to direct the user to a local copy.

The jsp:param and jsp:params Elements

The jsp:param element is used with jsp:plugin in a manner similar to the way that PARAM is

used with APPLET, specifying a name and value that are accessed from within the applet by getParameter. There are two main differences, however. First, since jsp:param follows XML syntax, attribute names must be lower case, attribute values must be enclosed in single or double quotes, and the element must end with />, not just >. Second, all jsp:param entries must be enclosed within a jsp:params element.

So, for example, you would replace

```
<APPLET CODE="MyApplet.class"
        WIDTH=475 HEIGHT=350>
  <PARAM NAME="PARAM1" VALUE="VALUE1">
  <PARAM NAME="PARAM2" VALUE="VALUE2">
</APPLET>
```

with

```
<jsp:plugin type="applet"
            code="MyApplet.class"
            width="475" height="350">
  <jsp:params>
    <jsp:param name="PARAM1" value="VALUE1" />
    <jsp:param name="PARAM2" value="VALUE2" />
  </jsp:params>
</jsp:plugin>
```

The jsp:fallback Element

The jsp:fallback element provides alternative text to browsers that do not support OBJECT or EMBED. You use this element in almost the same way as you would use alternative text placed within an APPLET element. So, for example, you would replace

```
<APPLET CODE="MyApplet.class"
        WIDTH=475 HEIGHT=350>
  <B>Error: this example requires Java.</B>
</APPLET>
```

with

```
<jsp:plugin type="applet"
            code="MyApplet.class"
            width="475" height="350">
  <jsp:fallback>
    <B>Error: this example requires Java.</B>
  </jsp:fallback>
</jsp:plugin>
```

A jsp:plugin Example

Listing 13.8 shows a JSP page that uses the jsp:plugin element to generate an entry for the Java 2 Plug-in. Listing 13.9 shows the code for the applet itself (which uses Swing, Java 2D, and the auxiliary classes of Listings 13.10 through 13.12). Figure 13-3 shows the result.

Listing 13.8 PluginApplet.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Using jsp:plugin</TITLE>
<LINK REL=STYLESHEET
```

```
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">
      Using jsp:plugin</TABLE>

<P>
<jsp:plugin type="applet"
            code="PluginApplet.class"
            width="370" height="420">
</jsp:plugin>
</CENTER></BODY></HTML>
```

Listing 13.9 PluginApplet.java

```
import javax.swing.*;

/** An applet that uses Swing and Java 2D and thus requires
 *  the Java Plug-in.
 */

public class PluginApplet extends JApplet {
  public void init() {
    WindowUtilities.setNativeLookAndFeel();
    setContentPane(new TextPanel());
  }
}
```

Listing 13.10 TextPanel.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/** JPanel that places a panel with text drawn at various angles
 *  in the top part of the window and a JComboBox containing
 *  font choices in the bottom part.
 */

public class TextPanel extends JPanel
                       implements ActionListener {
  private JComboBox fontBox;
  private DrawingPanel drawingPanel;

  public TextPanel() {
    GraphicsEnvironment env =
      GraphicsEnvironment.getLocalGraphicsEnvironment();
    String[] fontNames = env.getAvailableFontFamilyNames();
    fontBox = new JComboBox(fontNames);
    setLayout(new BorderLayout());
    JPanel fontPanel = new JPanel();
    fontPanel.add(new JLabel("Font:"));
    fontPanel.add(fontBox);
    JButton drawButton = new JButton("Draw");
    drawButton.addActionListener(this);
    fontPanel.add(drawButton);
    add(fontPanel, BorderLayout.SOUTH);
    drawingPanel = new DrawingPanel();
    fontBox.setSelectedItem("Serif");
    drawingPanel.setFontName("Serif");
```

```
    add(drawingPanel, BorderLayout.CENTER);
  }

  public void actionPerformed(ActionEvent e) {
    drawingPanel.setFontName((String)fontBox.getSelectedItem());
    drawingPanel.repaint();
  }
}
```

Listing 13.11 DrawingPanel.java

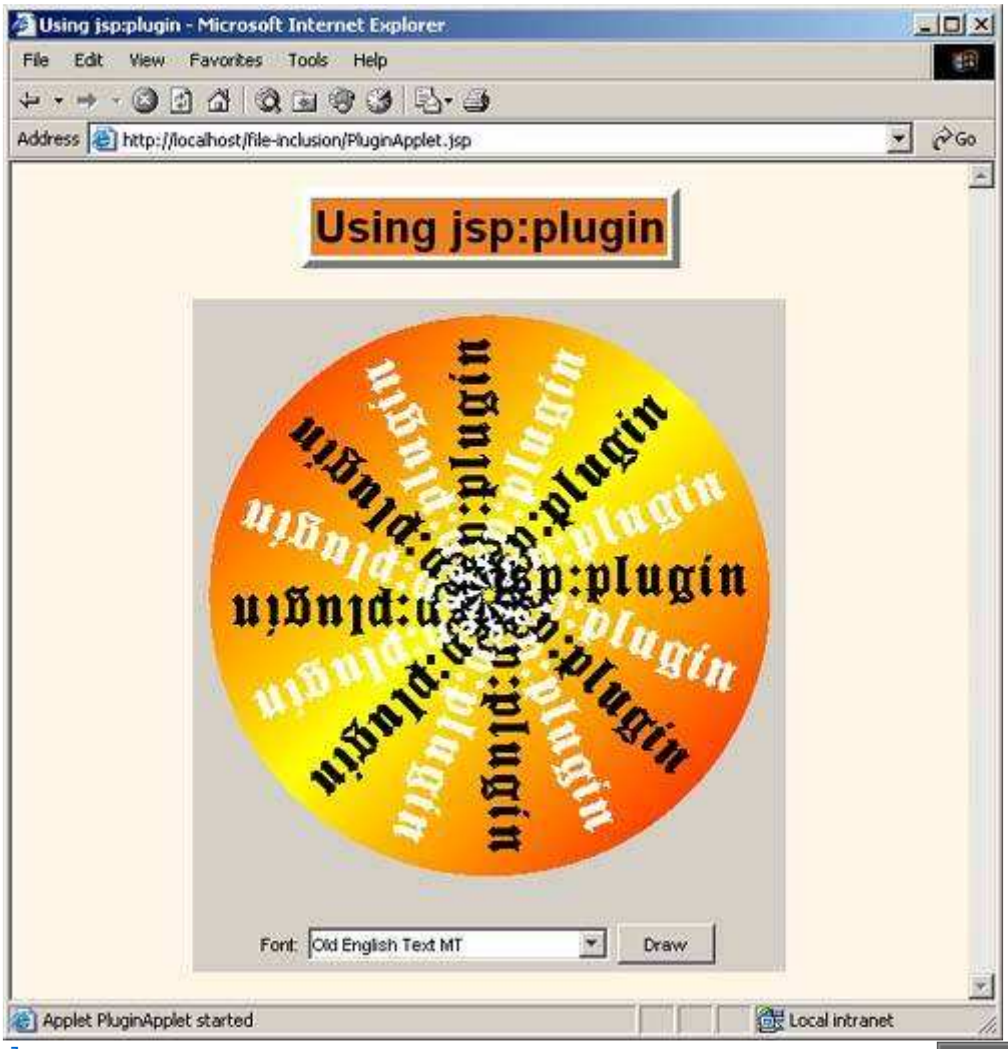
```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

/** A window with text drawn at an angle. The font is
 *  set by means of the setFontName method.
 */

class DrawingPanel extends JPanel {
  private Ellipse2D.Double circle =
    new Ellipse2D.Double(10, 10, 350, 350);
  private GradientPaint gradient =
    new GradientPaint(0, 0, Color.red, 180, 180, Color.yellow,
                      true); // true means to repeat pattern
  private Color[] colors = { Color.white, Color.black };


  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;
    g2d.setPaint(gradient);
    g2d.fill(circle);
    g2d.translate(185, 185);
    for (int i=0; i<16; i++) {
      g2d.rotate(Math.PI/8.0);
      g2d.setPaint(colors[i%2]);
      g2d.drawString("jsp:plugin", 0, 0);
    }
  }

  public void setFontName(String fontName) {
    setFont(new Font(fontName, Font.BOLD, 35));
  }
}
```

Listing 13.12 WindowUtilities.java

```
import javax.swing.*;
import java.awt.*;

/** A few utilities that simplify using windows in Swing. */

public class WindowUtilities {

  /** Tell system to use native look and feel, as in previous
   *  releases. Metal (Java) LAF is the default otherwise.
   */

  public static void setNativeLookAndFeel() {
    try {
      UIManager.setLookAndFeel
        (UIManager.getSystemLookAndFeelClassName());
```

```
    } catch(Exception e) {
      System.out.println("Error setting native LAF: " + e);
    }
  }

  ... // See www.coreservlets.com for remaining code.
}
```

Figure 13-3. Result of `PluginApplet.jsp` in Internet Explorer with the JDK 1.4 plug-in.



[Team LiB]
PREVIOUS NEXT