J2EE

Servlets and JSP

Advanced topics

Presented by Bartosz Sakowicz

JSP and servlets integration

Advantages and disadvantages of servlets and JSP:

• Servlets are good when your application requires a lot of real programming.
• Generating HTML with servlets can be tedious and can yield a result that is hard to modify.
• JSP document provides a single overall presentation. Beans and custom tags, although very flexible, don't overcome the limitation that the JSP page defines a relatively fixed top-level page appearance.
• **The solution is to use both servlets and JSP**. If you have a complicated application that may require several substantially different presentations, a servlet can handle the initial request, partially process the data, set up beans, then forward the results to one of a number of different JSP pages, depending on the circumstances.

RequestDispatcher class

To let servlets forward requests or include external content you use a **RequestDispatcher.** You obtain a RequestDispatcher by calling the getRequestDispatcher method of ServletContext, supplying a URL relativeto the server root. **Example**:

String url = "/presentations/presentation1.jsp";
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher(url);

Once you have a RequestDispatcher, you use **forward** to completely transfer control to the associated URL and use **include** to output the associated URL's content.

Forwarding example

```
public void doGet(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException {
        String operation = request.getParameter("operation");
        if (operation == null) { operation = "unknown"; }
        if (operation.equals("operation1")) {
                gotoPage("/operations/presentation1.jsp",
                                request, response);
        } else if (operation.equals("operation2")) {
                gotoPage("/operations/presentation2.jsp",
                                request, response);
        } else {
        gotoPage("/operations/unknownRequestHandler.jsp",
                        request, response);
        }
}
```

Forwarding example(2)

```
private void gotoPage(String address,
HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher(address);

        dispatcher.forward(request, response);
}
```

Supplying Data to the destination page

Main reasons why the destination page shouldn't look up and process all the data itself:

• Complicated programming is easier in a servlet than in a JSP page.

• Multiple JSP pages may require the same data, so it would be wasteful for each JSP page to have to set up the same data.

A better approach is for the original servlet to set up the information that the destination pages need, then store it somewhere that the destination pages can easily access.

There are two main places for the servlet to store the data that the JSP pages will use: in the HttpServletRequest and as a bean in the location specific to the scope attribute of jsp:useBean

Supplying ... (2)

The originating servlet would store arbitrary objects in the HttpServletRequest by using:

request.setAttribute("key1", value1);

The destination page would access the value by using a JSP scripting element to call:

Type1 value1 = (Type1)request.getAttribute("key1");

Supplying ... (3)

For complex values a better approach is to represent the value as a bean and store it in the location used by jsp:useBean for shared beans. To make a bean accessible to all servlets or JSP pages in the server or Web application, the originating servlet would do the following:

Type1 value1 = computeValueFromRequest(request);
getServletContext().setAttribute("key1", value1);

The destination JSP page would normally access the previously stored value by using jsp:useBean as follows:
<jsp:useBean id="key1" class="Type1" scope="application" />
Alternatively, the destination page could use a scripting element to explicitly call *application.getAttribute("key1")* and cast the result to *Type1*.

Supplying ... (4)

The Servlet 2.2 specification adds a third way to send data to the destination page when using GET requests: append the query data to the URL. Example:

*String address = "**/path/resource.jsp?newParam=value**";*

RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher(address);

dispatcher.forward(request, response);

Differences between forwarding and sendRedirect

• sendRedirect requires the client to reconnect to the new resource, whereas the forward method of RequestDispatcher is handled completely on the server.

• sendRedirect does not automatically preserve all of the request data; forward does.

• **sendRedirect results in a different final URL whereas with forward the URL of the original servlet is maintained.**

Differences between forwarding and sendRedirect

• sendRedirect requires the client to reconnect to the new resource, whereas the forward method of RequestDispatcher is handled completely on the server.

• sendRedirect does not automatically preserve all of the request data; forward does.

• **sendRedirect results in a different final URL whereas with forward the URL of the original servlet is maintained.**

Interpreting relative URL in the destination page

If the destination page uses relative URLs for images or style sheets, it needs to make them relative to the server root, not to the destination page's actual location. **Example:**

<LINK REL=STYLESHEET HREF="my-styles.css" >

If the JSP page containing this entry is accessed by means of a forwarded request, mystyles.css will be interpreted relative to the URL of the originating servlet, not relative to the JSP page itself, almost certainly resulting in an error. The solution is to give the full server path to the style sheet file:

<LINK REL=STYLESHEET HREF="/path/my-styles.css" >

The same approach is required for addresses used in *<IMG SRC=...> and <A HREF=...>.*

Alternative ways of getting RequestDispatcher

In servers that support version 2.2 of the servlet specification, there are two additional ways of obtaining a RequestDispatcher besides the getRequestDispatcher method of ServletContext:

Since most servers let you give explicit names to servlets or JSP pages, it is possible access them by name rather than by path. Use the **getNamedDispatcher** method of ServletContext for this task.

Alternative ways of getting RequestDispatcher(2)

If you might want to access a resource by a path relative to the current servlet's location, rather than relative to the server root you can use the **getRequestDispatcher** method of **HttpServletRequest** rather than the one from ServletContext.
Example:
If the originating servlet is at *http://host/travel/TopLevel*

getServletContext().getRequestDispatcher("/travel/cruises.jsp")

could be replaced by

request.getRequestDispatcher("cruises.jsp");

Including static or dynamic content

If the servlet wants to generate some of the content itself but use a JSP page or static HTML document for other parts of the result, the servlet can use the include method of RequestDispatcher.

The process is very similar to that for forwarding requests: call the getRequestDispatcher method of ServletContext with an address relative to the server root, then call ***include*** with the HttpServletRequest and HttpServletResponse.

Including static or dynamic content (2)

Whatever request data was associated with the original request is also associated with the auxiliary request, and you can add new parameters (in version2.2 only) by appending them to the URL supplied to getRequestDispatcher.

Also supported in version 2.2 is the ability to get a requestDispatcher by name (getNamedDispatcher) or by using a relative URL (use the getRequestDispatcher method of the HttpServletRequest);

Including static or dynamic content (3)

include does one thing that ***forward*** does not: it automatically sets up attributes in the HttpServletRequestobject that describe the original request path in case the included servlet or JSP page needs that information. These attributes, available to the included resource by calling getAttribute on the HttpServletRequest, are:

- javax.servlet.include.request_uri
- javax.servlet.include.context_path
- javax.servlet.include.servlet_path
- javax.servlet.include.path_info
- javax.servlet.include.query_string

Differences between include directive and include method

- **In the JSP include directive the actual *source code* of JSP files was included in the page**

- **The include method of RequestDispatcher just includes the *result* of the specified resource.**

- The jsp:include action has behavior similar to that of the include method, except that jsp:include is available only from JSP pages, not from servlets.

Forwarding requests from JSP pages

In JSP, the **jsp:forward** action is simpler and easier to use than wrapping up RequestDispatcher code in a scriptlet. This action takes the following form:

```
<jsp:forward page="Relative URL" />
```

The page attribute is allowed to contain JSP expressions so that the destination can be computed at request time. Example:

```
<% String destination;
if (Math.random() > 0.5) {
        destination = "/examples/page1.jsp";
} else {
        destination = "/examples/page2.jsp";
}
%>
<jsp:forward page="<%= destination %>" />
```