11.12 Using Predefined Variables

When you wrote a `doGet` method for a servlet, you probably wrote something like this:

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  PrintWriter out = response.getWriter();
  out.println(...);
  ...
}
```

The servlet API told you the types of the arguments to `doGet`, the methods to call to get the session and writer objects, and their types. JSP changes the method name from `doGet` to `_jspService` and uses a `JspWriter` instead of a `PrintWriter`. But the idea is the same. The question is, who told you what variable names to use? The answer is, nobody! You chose whatever names you wanted.

For JSP expressions and scriptlets to be useful, you need to know what variable names the autogenerated servlet uses. So, the specification tells you. You are supplied with eight automatically defined local variables in `_jspService`, sometimes called "implicit objects." Nothing is special about these; they are merely the names of the local variables. *Local* variables. Not constants. Not JSP reserved words. Nothing magic. So, if you are writing code that is not part of the `_jspService` method, these variables are not available. In particular, since JSP declarations result in code that appears outside the `_jspService` method, these variables are not accessible in declarations. Similarly, they are not available in utility classes that are invoked by JSP pages. If you need a separate method to have access to one of these variables, do what you always do in Java code: pass the variable along.

The available variables are `request`, `response`, `out`, `session`, `application`, `config`, `pageContext`, and `page`. Details for each are given below. An additional variable called `exception` is available, but only in error pages. This variable is discussed in Chapter 12 (Controlling the Structure of Generated Servlets: The JSP page Directive) in the sections on the `errorPage` and `isErrorPage` attributes.

- **`request`**

  This variable is the `HttpServletRequest` associated with the request; it gives you access to the request parameters, the request type (e.g., `GET` or `POST`), and the incoming HTTP headers (e.g., cookies).

- **`response`**

  This variable is the `HttpServletResponse` associated with the response to the client. Since the output stream (see `out`) is normally buffered, it is usually legal to set HTTP status codes and response headers in the body of JSP pages, even though the setting of headers or status codes is not permitted in servlets once any output has been sent to the client. If you turn buffering off, however (see the `buffer` attribute in Chapter 12), you must set status codes and headers before supplying any output.

- **`out`**

This variable is the `Writer` used to send output to the client. However, to make it easy to set response headers at various places in the JSP page, `out` is not the standard `PrintWriter` but rather a buffered version of `Writer` called `JspWriter`. You can adjust the buffer size through use of the `buffer` attribute of the `page` directive (see Chapter 12). The `out` variable is used almost exclusively in scriptlets since JSP expressions are automatically placed in the output stream and thus rarely need to refer to `out` explicitly.

- **`session`**

  This variable is the `HttpSession` object associated with the request. Recall that sessions are created automatically in JSP, so this variable is bound even if there is no incoming session reference. The one exception is the use of the `session` attribute of the `page` directive (Chapter 12) to disable automatic session tracking. In that case, attempts to reference the `session` variable cause errors at the time the JSP page is translated into a servlet. See Chapter 9 for general information on session tracking and the `HttpSession` class.

- **`application`**

  This variable is the `ServletContext` as obtained by `getServletContext`. Servlets and JSP pages can store persistent data in the `ServletContext` object rather than in instance variables. `ServletContext` has `setAttribute` and `getAttribute` methods that let you store arbitrary data associated with specified keys. The difference between storing data in instance variables and storing it in the `ServletContext` is that the `ServletContext` is shared by all servlets and JSP pages in the Web application, whereas instance variables are available only to the same servlet that stored the data.

- **`config`**

  This variable is the `ServletConfig` object for this page. In principle, you can use it to read initialization parameters, but, in practice, initialization parameters are read from `jspInit`, not from `_jspService`.

- **`pageContext`**

  JSP introduced a class called `PageContext` to give a single point of access to many of the page attributes. The `PageContext` class has methods `getRequest`, `getResponse`, `getOut`, `getSession`, and so forth. The `pageContext` variable stores the value of the `PageContext` object associated with the current page. If a method or constructor needs access to multiple page-related objects, passing `pageContext` is easier than passing many separate references to `request`, `response`, `out`, and so forth.

- **`page`**

  This variable is simply a synonym for `this` and is not very useful. It was created as a placeholder for the time when the scripting language could be something other than Java.