11.10 Using Declarations

A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (*outside* the `_jspService` method that is called by `service` to process the request). A declaration has the following form:

```
<%! Field or Method Definition %>
```

Since declarations do not generate output, they are normally used in conjunction with JSP expressions or scriptlets. In principle, JSP declarations can contain field (instance variable) definitions, method definitions, inner class definitions, or even static initializer blocks: anything that is legal to put inside a class definition but outside any existing methods. In practice, however, declarations almost always contain field or method definitions.

One caution is warranted, however: do not use JSP declarations to override the standard servlet life-cycle methods (`service`, `doGet`, `init`, etc.). The servlet into which the JSP page gets translated already makes use of these methods. There is no need for declarations to gain access to `service`, `doGet`, or `doPost`, since calls to `service` are automatically dispatched to `_jspService`, which is where code resulting from expressions and scriptlets is put. However, for initialization and cleanup, you can use `jspInit` and `jspDestroy`—the standard `init` and `destroy` methods are guaranteed to call these two methods in servlets that come from JSP.

Core Approach

For initialization and cleanup in JSP pages, use JSP declarations to override `jspInit` or `jspDestroy`, not `init` or `destroy`.

Aside from overriding standard methods like `jspInit` and `jspDestroy`, the utility of JSP declarations for defining methods is somewhat questionable. Moving the methods to separate classes (possibly as static methods) makes them easier to write (since you are using a Java environment, not an HTML-like one), easier to test (no need to run a server), easier to debug (compilation warnings give the right line numbers; no tricks are needed to see the standard output), and easier to reuse (many different JSP pages can use the same utility class). However, using JSP declarations to define instance variables (fields), as we will see shortly, gives you something not easily reproducible with separate utility classes: a place to store data that is persistent between requests.

Core Approach

Define most methods with separate Java classes, not JSP declarations.

JSP/Servlet Correspondence

JSP declarations result in code that is placed inside the servlet class definition but outside the `_jspService` method. Since fields and methods can be declared in any order, it does not matter whether the code from declarations goes at the top or bottom of the servlet. For instance,

Listing 11.10 shows a small JSP snippet that includes some static HTML, a JSP declaration, and a JSP expression. Listing 11.11 shows a servlet that might result. Note that the specific name of the resultant servlet is not defined by the JSP specification, and in fact, different servers have different conventions. Besides, as already stated, different vendors will produce this code in slightly different ways, and we oversimplified the `out` variable (which is a `JspWriter`, not the slightly simpler `PrintWriter` that results from a call to `getWriter`). Finally, the servlet will never implement `HttpJspPage` directly, but rather will extend some vendor-specific class that already implements `HttpJspPage`. So, don't expect the code your server generates to look *exactly* like this.

Listing 11.10 Sample JSP Declaration

```
<H1>Some Heading</H1>
<%!
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>");
  }
%>
<%= randomHeading() %>
```

Listing 11.11 Representative Resulting Servlet Code: Declaration

```
public class xxxx implements HttpJspPage {
 private String randomHeading() {
   return("<H2>" + Math.random() + "</H2>");
 }
 public void _jspService(HttpServletRequest request,
                         HttpServletResponse response)
    throws ServletException, IOException {
  response.setContentType("text/html");
  HttpSession session = request.getSession();
  JspWriter out = response.getWriter();
  out.println("<H1>Some Heading</H1>");
  out.println(randomHeading());
  ...
 }

  ...
}
```

XML Syntax for Declarations

The XML equivalent of `<%! Field or Method Definition %>` is

```
<jsp:declaration>Field or Method Definition</jsp:declaration>
```

In JSP 1.2 and later, servers are required to support this syntax as long as authors don't mix the XML version (`<jsp:declaration> ... </jsp:declaration>`) and the standard ASP-like version (`<%! ... %>`) in the same page. The entire page must follow XML syntax if you are going to use the XML form, so most developers stick with the classic syntax except when they are using XML anyhow. Remember that XML elements are case sensitive; be sure to use `jsp:declaration` in lower case.