10.3 Advantages of JSP Over Competing Technologies

A number of years ago, the lead author of this book (Marty) was invited to attend a small 20-person industry roundtable discussion on software technology. Sitting in the seat next to Marty was James Gosling, inventor of the Java programming language. Sitting several seats away was a high-level manager from a very large software company in Redmond, Washington. During the discussion, the moderator brought up the subject of Jini, which at that time was a new Java technology. The moderator asked the manager what he thought of it, and the manager responded that it was too early to tell, but that it seemed to be an excellent idea. He went on to say that they would keep an eye on it, and if it seemed to be catching on, they would follow his company's usual "embrace and extend" strategy. At this point, Gosling lightheartedly interjected "You mean *dis*grace and *dis*tend."

Now, the grievance that Gosling was airing was that he felt that this company would take technology from other companies and suborn it for their own purposes. But guess what? The shoe is on the other foot here. The Java community did not invent the idea of designing pages as a mixture of static HTML and dynamic code marked with special tags. For example, ColdFusion did it years earlier. Even ASP (a product from the very software company of the aforementioned manager) popularized this approach before JSP came along and decided to jump on the bandwagon. In fact, JSP not only adopted the general idea, it even used many of the same special tags as ASP did.

So, the question becomes: why use JSP instead of one of these other technologies? Our first response is that we are not arguing that everyone should. Several of those other technologies are quite good and are reasonable options in some situations. In other situations, however, JSP is clearly better. Here are a few of the reasons.

Versus .NET and Active Server Pages (ASP)

.NET is well-designed technology from Microsoft. ASP.NET is the part that directly competes with servlets and JSP. The advantages of JSP are twofold.

First, JSP is portable to multiple operating systems and Web servers; you aren't locked into deploying on Windows and IIS. Although the core .NET platform runs on a few non-Windows platforms, the ASP part does not. You cannot expect to deploy serious ASP.NET applications on multiple servers and operating systems. For some applications, this difference does not matter. For others, it matters greatly.

Second, for some applications the choice of the underlying language matters greatly. For example, although .NET's C# language is very well designed and is similar to Java, fewer programmers are familiar with either the core C# syntax or the many auxiliary libraries. In addition, many developers still use the original version of ASP. With this version, JSP has a clear advantage for the dynamic code. With JSP, the dynamic part is written in Java, not VBScript or another ASP-specific language, so JSP is more powerful and better suited to complex applications that require reusable components.

You could make the same argument when comparing JSP to the previous version of ColdFusion; with JSP you can use Java for the "real code" and are not tied to a particular server product. However, the current release of ColdFusion is within the context of a J2EE server, allowing developers to easily mix ColdFusion and servlet/JSP code.

Versus PHP

PHP (a recursive acronym for "PHP: Hypertext Preprocessor") is a free, open-source, HTML-embedded scripting language that is somewhat similar to both ASP and JSP. One advantage of

JSP is that the dynamic part is written in Java, which already has an extensive API for networking, database access, distributed objects, and the like, whereas PHP requires learning an entirely new, less widely used language. A second advantage is that JSP is much more widely supported by tool and server vendors than is PHP.

Versus Pure Servlets

JSP doesn't provide any capabilities that couldn't, in principle, be accomplished with servlets. In fact, JSP documents are automatically translated into servlets behind the scenes. But it is more convenient to write (and to modify!) regular HTML than to use a zillion `println` statements to generate the HTML. Plus, by separating the presentation from the content, you can put different people on different tasks: your Web page design experts can build the HTML by using familiar tools and either leave places for your servlet programmers to insert the dynamic content or invoke the dynamic content indirectly by means of XML tags.

Does this mean that you can just learn JSP and forget about servlets? Absolutely not! JSP developers need to know servlets for four reasons:

1. JSP pages get translated into servlets. You can't understand how JSP works without understanding servlets.

2. JSP consists of static HTML, special-purpose JSP tags, and Java code. What kind of Java code? Servlet code! You can't write that code if you don't understand servlet programming.

3. Some tasks are better accomplished by servlets than by JSP. JSP is good at generating pages that consist of large sections of fairly well structured HTML or other character data. Servlets are better for generating binary data, building pages with highly variable structure, and performing tasks (such as redirection) that involve little or no output.

4. Some tasks are better accomplished by a *combination* of servlets and JSP than by *either* servlets or JSP alone. See Chapter 15 for details.

Versus JavaScript

JavaScript, which is completely distinct from the Java programming language, is normally used to dynamically generate HTML on the *client*, building parts of the Web page as the browser loads the document. This is a useful capability and does not normally overlap with the capabilities of JSP (which runs only on the *server*). JSP pages still include `SCRIPT` tags for JavaScript, just as normal HTML pages do. In fact, JSP can even be used to dynamically generate the JavaScript that will be sent to the client. So, JavaScript is not a competing technology; it is a complementary one.

It is also possible to use JavaScript on the server, most notably on Sun ONE (formerly iPlanet), IIS, and BroadVision servers. However, Java is more powerful, flexible, reliable, and portable.

Versus WebMacro or Velocity

JSP is by no means perfect. Many people have pointed out features that could be improved. This is a good thing, and one of the advantages of JSP is that the specification is controlled by a community that draws from many different companies. So, the technology can incorporate improvements in successive releases.

However, some groups have developed alternative Java-based technologies to try to address these deficiencies. This, in our judgment, is a mistake. Using a third-party tool like Apache Struts (see Volume 2 of this book) that *augments* JSP and servlet technology is a good idea when that tool adds sufficient benefit to compensate for the additional complexity. But using a

nonstandard tool that tries to *replace* JSP is a bad idea. When choosing a technology, you need to weigh many factors: standardization, portability, integration, industry support, and technical features. The arguments for JSP alternatives have focused almost exclusively on the technical features part. But portability, standardization, and integration are also very important. For example, as discussed in Section 2.11, the servlet and JSP specifications define a standard directory structure for Web applications and provide standard files (`.war` files) for deploying Web applications. All JSP-compatible servers must support these standards. Filters (Volume 2) can be set up to apply to any number of servlets or JSP pages, but not to nonstandard resources. The same goes for Web application security settings (see Volume 2).

Besides, the tremendous industry support for JSP and servlet technology results in improvements that mitigate many of the criticisms of JSP. For example, the JSP Standard Tag Library (Volume 2) and the JSP 2.0 expression language (Chapter 16) address two of the most well-founded criticisms: the lack of good iteration constructs and the difficulty of accessing dynamic results without using either explicit Java code or verbose `jsp:useBean` elements.