# Overview of JSP Technology

**Topics in This Chapter**

- Understanding the need for JSP
- Evaluating the benefits of JSP
- Comparing JSP to other technologies
- Avoiding JSP misconceptions
- Installing JSP pages
- Surveying JSP syntax

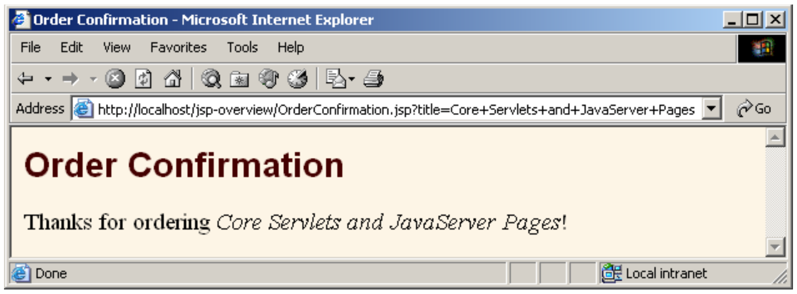
# Chapter 10

JavaServer Pages (JSP) technology enables you to mix regular, static HTML with dynamically generated content. You simply write the regular HTML in the normal manner, using familiar Web-page-building tools. You then enclose the code for the dynamic parts in special tags, most of which start with `<%` and end with `%>`.

For example, Listing 10.1 (Figure 10–1) presents a very small JSP page that uses a request parameter to display the title of a book. Notice that the listing is mostly standard HTML; the dynamic code consists entirely of the half line shown in bold in the listing.

**Listing 10.1** OrderConfirmation.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Order Confirmation</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H2>Order Confirmation</H2>
Thanks for ordering <I><%= request.getParameter("title") %></I>!
</BODY></HTML>
```


Order Confirmation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/jsp-overview/OrderConfirmation.jsp?title=Core+Servlets+and+JavaServer+Pages
Go
Done
Local intranet

Order Confirmation

Thanks for ordering *Core Servlets and JavaServer Pages*!

**Figure 10–1** Result of OrderConfirmation.jsp.

You can think of servlets as Java code with HTML inside; you can think of JSP as HTML with Java code inside. Now, neither servlets nor JSP pages are restricted to using HTML, but they usually do, and this over-simplified description is a common way to view the technologies.

Now, despite the large apparent differences between JSP pages and servlets, behind the scenes they are the same thing. JSP pages are translated into servlets, the servlets are compiled, and at request time it is the compiled servlets that execute. So, writing JSP pages is really just another way of writing servlets.

Even though servlets and JSP pages are equivalent behind the scenes, they are not equally useful in all situations. Separating the static HTML from the dynamic content provides a number of benefits over servlets alone, and the approach used in JavaServer Pages offers several advantages over competing technologies. This chapter explains the reasons for using JSP, discusses its benefits, dispels some misconceptions, shows you how to install and execute JSP pages, and summarizes the JSP syntax discussed in the rest of the book.

## 10.1 The Need for JSP

“Hey!” you say, “You just spent several chapters extolling the virtues of servlets. I like servlets. Servlets are convenient to write and efficient to execute. They make it simple to read request parameters and to set up custom code to handle missing and malformed data. They can easily make use of HTTP request headers and can flexibly manipulate HTTP response data. They can customize their behavior based on cookies, track user-specific data with the session-tracking API, and talk to relational databases with JDBC. What more do I need?”

Well, this is a good point. Servlets are indeed useful, and JSP by no means makes them obsolete. However, look at the list of topics for which servlets excel. They are all tasks related to *programming* or data *processing*. But servlets are not so good at *presentation*. Servlets have the following deficiencies when it comes to generating the output:

- **It is hard to write and maintain the HTML.** Using `print` statements to generate HTML? Hardly convenient: you have to use parentheses and semicolons, have to insert backslashes in front of embedded double quotes, and have to use string concatenation to put the content together. Besides, it simply does not look like HTML, so it is harder to visualize. Compare this servlet style with Listing 10.1 where you hardly even notice that the page is not an ordinary HTML document.
- **You cannot use standard HTML tools.** All those great Web-site development tools you have are of little use when you are writing Java code.
- **The HTML is inaccessible to non-Java developers.** If the HTML is embedded within Java code, a Web development expert who does not know the Java programming language will have trouble reviewing and changing the HTML.

## 10.2 Benefits of JSP

JSP pages are translated into servlets. So, fundamentally, any task JSP pages can perform could also be accomplished by servlets. However, this underlying equivalence does not mean that servlets and JSP pages are equally appropriate in all scenarios. The issue is not the power of the technology, it is the convenience, productivity, and maintainability of one or the other. After all, anything you can do on a particular computer platform in the Java programming language you could also do in assembly language. But it still matters which you choose.

JSP provides the following benefits over servlets alone:

- **It is easier to write and maintain the HTML.** Your static code is ordinary HTML: no extra backslashes, no double quotes, and no lurking Java syntax.
- **You can use standard Web-site development tools.** For example, we use Macromedia Dreamweaver for most of the JSP pages in the book. Even HTML tools that know nothing about JSP can be used because they simply ignore the JSP tags.
- **You can divide up your development team.** The Java programmers can work on the dynamic code. The Web developers can concentrate on the presentation layer. On large projects, this division is very important. Depending on the size of your team and the complexity of your project, you can enforce a weaker or stronger separation between the static HTML and the dynamic content.

Now, this discussion is not to say that you should stop using servlets and use only JSP instead. By no means. Almost all projects will use both. For some requests in your project, you will use servlets. For others, you will use JSP. For still others, you will combine them with the MVC architecture (Chapter 15). You want the appropriate tool for the job, and servlets, by themselves, do not complete your toolkit.

## 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.

## 10.4 Misconceptions About JSP

In this section, we address some of the most common misunderstandings about JSP.

### Forgetting JSP Is Server-Side Technology

The book's Web site lists the lead author's email address: hall@coreservlets.com. Furthermore, Marty teaches JSP and servlet training courses for various companies and at public venues. Consequently, he gets a lot of email with servlet and JSP questions. Here are some typical questions he has received (most of them repeatedly).

- Our server is running JDK 1.4. So, how do I put a Swing component in a JSP page?
- How do I put an image into a JSP page? I do not know the proper Java I/O commands to read image files.
- Since Tomcat does not support JavaScript, how do I make images that are highlighted when the user moves the mouse over them?
- Our clients use older browsers that do not understand JSP. What should we do?
- When our clients use "View Source" in a browser, how can I prevent them from seeing the JSP tags?

All of these questions are based upon the assumption that browsers know something about the server-side process. But they do not. Thus:

- For putting applets with Swing components into Web pages, what matters is the browser's Java version—the server's version is irrelevant. If the browser supports the Java 2 platform, you use the normal `APPLET` (or Java plug-in) tag and would do so even if you were using non-Java technology on the server.
- You do not need Java I/O to read image files; you just put the image in the directory for Web resources (i.e., two levels up from `WEB-INF/classes`) and output a normal `IMG` tag.
- You create images that change under the mouse by using client-side JavaScript, referenced with the `SCRIPT` tag; this does not change just because the server is using JSP.
- Browsers do not "support" JSP at all—they merely see the output of the JSP page. So, make sure your JSP outputs HTML compatible with the browser, just as you would do with static HTML pages.
- And, of course you need not do anything to prevent clients from seeing JSP tags; those tags are processed on the server and are not part of the output that is sent to the client.

## Confusing Translation Time with Request Time

A JSP page is converted into a servlet. The servlet is compiled, loaded into the server's memory, initialized, and executed. But which step happens when? To answer that question, remember two points:

- The JSP page is translated into a servlet and compiled only the first time it is accessed after having been modified.
- Loading into memory, initialization, and execution follow the normal rules for servlets.

Table 10.1 gives some common scenarios and tells whether or not each step occurs in that scenario. The most frequently misunderstood entries are highlighted. When referring to the table, note that servlets resulting from JSP pages use the `_jspService` method (called for both `GET` and `POST` requests), not `doGet` or `doPost`. Also, for initialization, they use the `jspInit` method, not the `init` method.

**Table 10.1** JSP Operations in Various Scenarios

| | JSP page translated into servlet | Servlet compiled | Servlet loaded into server's memory | `jspInit` called | `_jspService` called |
|---|---|---|---|---|---|
| | | | ***Page first written*** | | |
| Request 1 | Yes | Yes | Yes | Yes | Yes |
| Request 2 | **No** | **No** | **No** | **No** | Yes |
| | | | ***Server restarted*** | | |
| Request 3 | **No** | **No** | Yes | Yes | Yes |
| Request 4 | No | No | No | No | Yes |
| | | | ***Page modified*** | | |
| Request 5 | Yes | Yes | Yes | Yes | Yes |
| Request 6 | No | No | No | No | Yes |

## Thinking JSP Alone Is Sufficient

There is a small community of developers that are so enamored with JSP that they use it for practically everything. Most of these developers *never* use servlets; many never even use auxiliary helper classes—they just build large complex JSP pages for each and every task.

This is a mistake. JSP is an excellent tool. But the fundamental problem it addresses is *presentation*: the difficulty of creating and maintaining HTML to represent the result of a request. JSP is a good choice for pages with relatively fixed formats and lots of static text. JSP, by itself, is less good for applications that have a variable structure, is poor for applications that have mostly dynamic data, and is totally unsuitable for applications that output binary data or manipulate HTTP without generating explicit output (as with the search engine servlet of Section 6.4). Still other applications are best solved with neither servlets alone nor JSP alone, but with a combination of the two (see Chapter 15).

JSP is a powerful and widely applicable tool. Nevertheless, other tools are sometimes better. Choose the right tool for the job.

### Thinking Servlets Alone Are Sufficient

At the other end of the spectrum from the JSP-only camp is the servlets-only camp. Adherents of this camp state, quite rightly, that JSP pages are really just dressed up servlets, so JSP pages cannot accomplish anything that could not also be done with servlets. From this, they conclude that you should stick with servlets, where you have access to the full underlying power, have complete control, and can see exactly what is happening. Hmm, have you heard this argument before? It sounds a lot like the position of the "don't be a wimp; write all your code in assembly language" crowd.

Yes, you could use servlets for any task for which JSP is used. But it is not always equally convenient to do so. For tasks that involve a lot of static HTML content, use of JSP technology (or a combination of JSP and servlets) simplifies the creation and maintenance of the HTML, permits you to use industry standard Web site creation tools, and lets you "divide and conquer" by splitting your effort between the Java developers and the Web developers.

Servlets are powerful and widely applicable tools. Nevertheless, other tools are sometimes better. Choose the right tool for the job.

## 10.5 Installation of JSP Pages

Servlets require you to set your `CLASSPATH`, use packages to avoid name conflicts, install the class files in servlet-specific locations, and use special-purpose URLs. Not so with JSP pages. JSP pages can be placed in the same directories as normal HTML pages, images, and style sheets; they can also be accessed through URLs of the same form as those for HTML pages, images, and style sheets. Here are a few examples of default installation locations (i.e., locations that apply when you aren't using custom Web applications) and associated URLs. Where we list *SomeDirectory*, you can use any directory name you like, except that you are never allowed to use WEB-INF or META-INF as directory names. When using the default Web application, you also have to avoid a directory name that matches the URL prefix of any other Web application. For information on defining your own Web applications, see Section 2.11.

### JSP Directories for Tomcat (Default Web Application)

- **Main Location.**
  *install_dir*/webapps/ROOT

- **Corresponding URL.**
  http://*host*/*SomeFile*.jsp
- **More Specific Location (Arbitrary Subdirectory).**
  *install_dir*/webapps/ROOT/*SomeDirectory*
- **Corresponding URL.**
  http://*host*/*SomeDirectory*/*SomeFile*.jsp

## JSP Directories for JRun (Default Web Application)

- **Main Location.**
  *install_dir*/servers/default/default-ear/default-war
- **Corresponding URL.**
  http://*host*/*SomeFile*.jsp
- **More Specific Location (Arbitrary Subdirectory).**
  *install_dir*/servers/default/default-ear/default-war/*SomeDirectory*
- **Corresponding URL.**
  http://*host*/*SomeDirectory*/*SomeFile*.jsp

## JSP Directories for Resin (Default Web Application)

- **Main Location.**
  *install_dir*/doc
- **Corresponding URL.**
  http://*host*/*SomeFile*.jsp
- **More Specific Location (Arbitrary Subdirectory).**
  *install_dir*/doc/*SomeDirectory*
- **Corresponding URL.**
  http://*host*/*SomeDirectory*/*SomeFile*.jsp

Note that, although JSP pages *themselves* need no special installation directories, any Java classes called *from* JSP pages still need to go in the standard locations used by servlet classes (e.g., .../WEB-INF/classes/*directoryMatchingPackageName*; see Section 2.10). Note that the Java classes used by JSP pages should *always* be in packages; this point is discussed further in later chapters.

## 10.6 Basic Syntax

Here is a quick summary of the various JSP constructs you will see in this book.

### HTML Text

- **Description:**
  HTML content to be passed unchanged to the client
- **Example:**
  `<H1>Blah</H1>`
- **Discussed in:**
  Section 11.1

### HTML Comments

- **Description:**
  HTML comment that is sent to the client but not displayed by the browser
- **Example:**
  `<!-- Blah -->`
- **Discussed in:**
  Section 11.1

### Template Text

- **Description:**
  Text sent unchanged to the client. HTML text and HTML comments are just special cases of this.
- **Example:**
  *Anything other than the syntax of the following subsections*
- **Discussed in:**
  Section 11.1

### JSP Comment

- **Description:**
  Developer comment that is not sent to the client
- **Example:**
  `<%-- Blah --%>`

- **Discussed in:**
  Section 11.1

## JSP Expression

- **Description:**
  Expression that is evaluated and sent to the client each time the page is requested
- **Example:**
  `<%= Java Value %>`
- **Discussed in:**
  Section 11.4

## JSP Scriptlet

- **Description:**
  Statement or statements that are executed each time the page is requested
- **Example:**
  `<% Java Statement %>`
- **Discussed in:**
  Section 11.7

## JSP Declaration

- **Description:**
  Field or method that becomes part of class definition when page is translated into a servlet
- **Examples:**
  `<%! Field Definition %>`
  `<%! Method Definition %>`
- **Discussed in:**
  Section 11.10

## JSP Directive

- **Description:**
  High-level information about the structure of the servlet code (`page`), code that is included at page-translation time (`include`), or custom tag libraries used (`taglib`)

- **Example:**
  `<%@ directive att="val" %>`
- **Discussed in:**
  `page`: Chapter 12
  `include`: Chapter 13
  `taglib` and `tag`: Volume 2

## JSP Action

- **Description:**
  Action that takes place when the page is requested
- **Example:**
  `<jsp:blah>...</jsp:blah>`
- **Discussed in:**
  `jsp:include` and related: Chapter 13
  `jsp:useBean` and related: Chapter 14
  `jsp:invoke` and related: Volume 2

## JSP Expression Language Element

- **Description:**
  Shorthand JSP expression
- **Example:**
  `${ EL Expression }`
- **Discussed in:**
  Chapter 16

## Custom Tag (Custom Action)

- **Description:**
  Invocation of custom tag
- **Example:**
  ```
  <prefix:name>
  Body
  </prefix:name>
  ```
- **Discussed in:**
  Volume 2

## Escaped Template Text

- **Description:**
  Text that would otherwise be interpreted specially. Slash is removed and remaining text is sent to the client
- **Examples:**
  `<\%`
  `%\>`
- **Discussed in:**
  Section 11.1