Chapter 1

Introducing JavaServer Pages and the JSP Environment

In the early days of the Web (1995), a lot of web pages were black text on a light gray background with maybe a color logo. Then Java applets appeared. They became popular and quickly spread around the world because they attracted the viewer’s eye. Providing information took second place to providing users with entertainment. Quickly new technologies such as Flash and ActiveX and updated scripting languages such as VBScript and JavaScript jockeyed to provide dynamic presentations. However, none of the “eye candy” replacements had the Java advantage: platform independence. This advantage would later make a crucial difference in the proliferation and adoption of Java as the preferred server-side solution, and eventually it was decided that JavaServer Pages would be the official format for presenting enterprise Java on the Web.

In this chapter, we’ll discuss the role of JSP and its advantages, and we’ll compare JSP with similar technologies. We’ll also present an overview of web programming as a background for learning JSP. We’ll look at several of the available web server environments for JavaServer engines, and last we’ll look at some of the client-side Integrated Development Environments (IDEs) useful for developing JSP pages. The examples in this book will use the Tomcat Java server. (For Tomcat setup instructions, see Appendix A.)

Featured in this chapter:

- Web programming overview
- Background for learning JSP
- The various web server environments
- Client-side IDEs
The Growth of Server-Side Java and JSP’s Role

From the beginning, Java’s marketing mantra was “write once, run anywhere,” and properly written, Java could be labeled 100 percent pure Java—meaning that it could run on any platform with a Java virtual machine. Of course, poorly written code might only run on one operating system, if it ran at all.

Platform independence is an important feature of Java, and it has helped establish Java as a standard language that can unify all the operating systems serving pages to the World Wide Web. Although supporting plurality is important, statistically the Web is served overwhelmingly by Apache web servers running on Unix, or *nix.

Platform independence is an important feature of Java, and it has helped establish Java as a standard language that can unify all the operating systems serving pages to the World Wide Web. Although supporting plurality is important, statistically the Web is served overwhelmingly by Apache web servers running on Unix, or *nix.

The abbreviation *nix is often used to refer to the many flavors of Unix, for example, HP Unix, SGI Irix, Linux, Sun Solaris Unix, FreeBSD, and so on. For a long list, go to www.ugu.com.

Apache is an open-source web server, one of the most common in the world. Developed by the Apache Software Foundation, it has been around since 1995. The Apache Software Foundation also provides support for the Jakarta projects, all written in Java, with Tomcat being the foundation’s implementation of the JavaServer and JSP specification.

With such a large variety of web server operating systems, the “write once, run anywhere” mantra becomes more than a marketing buzzword—it’s a necessity. Also the creators and owners of Java, Sun Microsystems implement the Java environment on Unix (Solaris source code is available), whereas the Microsoft Windows Java environment has to function on top of the “black box” of the windows code. Fortunately, virtually all Java implementations have smoothed out their inconsistencies, so Java is now a stable, enterprise-ready environment. You can see why a Java environment is the natural choice for a majority of web servers.

What Is a JSP Page?

A JSP page is an HTML web page that has been enhanced by the addition of special tags and Java code, both of which are used by the web server and the JavaServer Engine to create a page for a client’s web browser.

JavaServer Pages (JSP) is a server-side programming language. The raw Java/JSP code never reaches the client. Java can be used on the client side in the form of applets, or applications installed on the desktop.

The JavaServer Engine strips out the special tags and Java source code and uses the code to interpret and create the page that the web server sends out; no Java code is actually sent to the client. Thus, any web browser with or without Java installed can use a JSP application! The JSP page is actually compiled into a servlet on the Java-enabled web server; thus to serve out JSP pages, a web server must have a JavaServer Engine installed or provide this functionality. Later in this chapter, we’ll look at different JavaServer Engines and JSP development environments.

Server-side Java was a late entry to the game; servlets only became possible with the release of Java 1.2. A servlet is a Java program that runs only on a Java-enabled web server servlet container. A servlet container is a JavaServer Engine that can host servlets and JSP pages. For example, Apache’s
Tomcat is a servlet container. A servlet is designed to run only on a web server inside a servlet container; you cannot run a servlet as a standalone application or in a browser window, but a browser can send requests and receive responses from a servlet.

A servlet receives a request, processes some information, and returns a result. That may sound vague, but it has to be said that way because servlets have a certain quality of vague flexibility. For example, you could design an application so that a client could request information about a race horse. One servlet could then return the statistics, while another servlet could return a picture of the horse. Another servlet might call the database to record the popularity of the horse and check the client’s connection speed. If the client happens to have a small screen or a slow connection, another servlet could pass the information on to an image-formatting servlet, which would then return the final processed page. Servlets can be chained together so that they go far beyond the typical web paradigm of one page request returning one page view.

HTTP servlets behave similarly to CGI programs: they accept raw input from a client’s browser, process the data, and return a result. The idea was that they could provide additional functionality at a crucial stage. For example, using regular HTML, a website can direct users to a selection of products, cache the selections (albeit somewhat awkwardly), send the user to an order submit page, process the final order by checking the parameters, call a database to log a charge against the customer’s account, and remove the item(s) from inventory.

CGI programs were written in the C programming language, whereas servlets take advantage of Java’s structured exception handling. As you will see, the JSP environment has further improved the handling of errors. Beyond mimicking the behavior of CGI programs, Java servlets added Session and Servlet Context objects. We’ll look at these in depth in Chapter 2.

Why Use JSP?
The following are some of the advantages associated with JSP:

- JSP encourages platform independence: conscientiously written code can be written once, run anywhere.
- JSP is the preferred Presentation layer for enterprise services.
- JSP provides stability by allowing developers to use and integrate established, thoroughly tested Java code.
- JSP can encourage a clean separation of the Data layer from the Presentation layer of an application.
- JSP’s seamless integration with the Java2 Enterprise Edition services encourages the development of extended architectures, not just a series of web pages linked together.

In addition to serving JSP pages, the JavaServer Engine provides functionality and memory for creating objects that persist across the user’s session state. Thus, a user can visit a page, begin an order, look at some unrelated pages, perhaps browse through similar products— with the JavaServer Engine patiently holding onto the user’s shopping cart selections—and then return to complete the order. All the behind-the-scenes caching of the user selections happens without extra code. Numerous time-savers and opportunities for code reuse make JSP popular for developers, especially compared with the options.
Learning JSP: How Hard Is This Going to Be?
From our experience of observing and teaching people new to JSP, we have found that given enough enthusiasm and effort anyone can become proficient in JSP and web programming. Different backgrounds, however, present different challenges.

Getting into JSP with a Background in HTML
Coming from an HTML background centers the developer on the user’s experience. HTML coders usually know what works for users, and usually they can give a fair estimate of what’s possible and how long it will take. The danger is that they might get hypnotized by flashy graphics instead of getting their application to talk to the database.

The most difficult part for someone with heavy HTML experience is that Java is not JavaScript; although many features of the languages are similar, they are two different languages. The challenge depends on how much Java you want to learn. Just enough Java to create good JSP pages is within ready grasp; proficiency in the fine points of graphic user components, for example GUI applets that enhance a JSP, will take longer. Depending on your situation and needs, the learning curve may not be a great barrier; the more complex features of Java and JSP are getting easier to use all the time through pre-packaged JavaBeans, Java classes, and JSP tag libraries. We'll look at these technologies in depth in the coming chapters.

Getting into JSP with a Background in C/C++ or Java Application Programming
Sooner or later, every application programmer longs to put up a web page or even get a web page to run their program. This desire arises because the web is everywhere and because these days HTML documentation of source code is a standard feature of many tools. For example, javadoc, a program bundled with the Java Development Kit (JDK), produces hyperlinked HTML files from Java source code. Many other IDE vendors have followed suit, adding this functionality to generate HTML road maps in C, C++, and other languages.

The most difficult part for application programmers is remembering and understanding the end user. This viewpoint is deceptively simple, and many engineers often quickly forget that a web application lives or dies by the user’s approval. In a way, it’s not surprising that they don’t put the user first because until recently university computer science courses often didn’t give due emphasis to designing for end users.

Many companies deal with this apparent skills gap by including a graphic artist and sometimes a creative director on a project team. This approach is a good management technique for remedying imbalances; however, it’s far more efficient to anticipate the problems and deal with them at the source. The more you understand, the more you will be valued as a developer. Although there may be a practical limit on how much technical knowledge you want to acquire, interpersonal understanding—such as the insight into what users want in a web page form and how they want to be able to find help—will often play a more important role in projects and promotions than technical know-how.

Many books can help you understand the sometimes inscrutable demands of users and managers. We recommend Designing Web Usability, by Jakob Nielsen (New Riders Publishing, 1999), for excellent insights and more than common sense, and The Inmates Are Running the Asylum, by Alan Cooper (Sams, 1999), for a detailed look at how to harmonize software design and user needs.
### Web Programming Languages Compared

As the web has grown, the demand for flexible ways to design it has only increased. Web programming languages have made difficult tasks easier and turned impossible ideas into attainable goals. In this section, we'll look at some of these languages. We'll point out their advantages and disadvantages and, as appropriate, discuss the advantage of using JSP instead of one of these languages.

#### Perl

Perl (Practical Extraction and Report Language) handles the processing and formatting of text exceptionally well. It can be expanded by the use of modules of code and is supported by many operating systems.

CGI (Common Gateway Interface) is the use of Perl in a web server environment. Perl can be resource expensive because there’s no built-in functionality for sharing application resources, for example, session tracking, database connection pooling, and so on. CGI/Perl has limited error-handling capabilities. The Web was first programmed with CGI scripts, and Perl remains largely procedural, which is excellent for simple tasks but poses special problems for code reuse. Often Perl is difficult to learn because of its many idiosyncratic language constructions.

The JSP advantage over Perl is that JSP provides special, customized error handling, which we’ll look at in detail in Chapters 4 and 8.

#### ASP (Active Server Pages)

ASP is a scripting language similar to JSP that provides native support for documents created with Microsoft products, for example, an Excel spreadsheet. ASP is proprietary, however, and runs only on Windows web servers.

*Note* Some software companies have/are developing ASP engines/interpreters for non-Windows platforms; however, developers must nevertheless be proficient in the proprietary Microsoft languages, VBS, C#, and Microsoft’s ASP framework leaves significant portions of the architecture components in limbo. ActiveX data objects, COM (Component Object Model) DLLs, and COM servers must be served up from a Windows Registry and a Service Control Manager (SCM).

Both ASP and IIS (Internet Information Services), Microsoft’s web server, rely on the COM/COM+ enterprise application framework being hosted on Microsoft servers. This arrangement can limit the flexibility of an application as it places demands on a business’s hardware and software choices.

The advantage of using JSP rather than ASP is that JSP is cross-platform ready, without any functionality being crippled. JSP’s backbone Java architecture allows supporting components to be distributed in source form or near equivalents, allowing greater user-ownership of the product and the potential for more thorough debugging (the source can be decompiled); whereas ActiveX and the COM/COM+ architecture models aim to provide discreet “black boxes” of software tied to a Windows machine.
ColdFusion

ColdFusion is fairly unique among web application tools because it uses a tag-based scripting language. When you view the source files of an application created in ColdFusion, you see tag sets that behave much the same as HTML tags. Thus, ColdFusion has an edge in projects in which programmers and graphic designers collaborate. Programmers can build some fairly sophisticated web applications that nevertheless allow designers to find their way around the tag-based source code even if they aren’t familiar with programming. ColdFusion is also extensible, meaning that if you find a task it can’t handle, you can create a solution with another programming language such as Perl or C and then seamlessly integrate it with ColdFusion.

ColdFusion’s tag-based language may also be its biggest drawback. Although the tags provide shortcuts to common programming tasks, they can also create unwieldy code that’s difficult for anyone but the original programmer to troubleshoot. Also, unlike many of the tools described here, ColdFusion Server is an application that must be purchased and installed in conjunction with your existing web server. Many web-hosting services support ColdFusion, but you can expect higher monthly rates.

PHP: Hypertext Preprocessor

PHP is an open-source scripting language, and as such it has the potential to change faster; individuals can participate in the development and maturation of the language.

PHP is a late entry in the arena of web server programming languages. It has an unusual scripting syntax, which is a mix of Perl, C++, and Java. Although PHP was originally designed for small sites, it’s now striving to grow and achieve speed and efficiency. Many web programmers select PHP because a number of free, open-source portal and bulletin board software modules are available.

JSP has an advantage in that its underlying language is Java, one of the world’s most popular programming languages. PHP’s language is powerful, but many managers and companies may be discouraged from using PHP because it can be more difficult to find developers and support personnel to write and maintain the code.

JSP in Various Server Environments

JSP can’t be run standalone for a couple of reasons, but for each reason JSP gains an advantage:

- JSP pages must be compiled before they are served. The JSP source files are changed from ASCII text into Java byte code, which speeds up their execution and allows the web server to cache them in memory.

**Note** The only disadvantage of the compiling and caching mechanism is that, in most cases, during debugging, a developer must stop the web server, manually delete the compiled pages, upload the new source files, and restart the web server. Otherwise, the old JSP pages will be served up, often even in the presence of newer source files!

- JSP pages run inside a servlet container that is usually part of a larger JavaServer Engine. That engine is usually part of a larger J2EE architecture. Because JSP pages must be hosted by a JavaServer Engine, they are always ready to integrate with other J2EE services as they come online.
Which Java Environment?

Now that you need a JavaServer environment, which one is best? Well that depends. At most companies, the Java environment is decided for you by issues of cost, availability of support, and the experiences of other developers and managers. However, because you might be called on to give an educated evaluation, and because you might have to work with one or more of these systems, it’s important to learn their names and something about their features.

**Tomcat**

Tomcat (see Figure 1.1) is the Java Servlet container implementation for the Java Servlet specification and the JavaServer Pages specification. It’s different from the other Java environments described here in that by itself it doesn’t provide all the features of a Sun-certified J2EE environment. Regardless, Tomcat has been and will continue to be the backbone of Java development.

**What Is J2EE?**

J2EE (officially, Java 2 Platform, Enterprise Edition) is a set of standards that allows for a unified back end for business architecture, including the following services:

- Middleware: EJB (Enterprise JavaBeans) and JMS (Java Message Service)
- Data management: JDBC (Java Database Connectivity) and the extensions that some database vendors provide in their JDBC drivers
- Transactions: JTS (Java Transaction Service)
- Directory services: JNDI (Java Naming and Directory Interface)
- Presentation layer: servlets and JSP

The items in this list account for the information management needs of most businesses. Indeed, the scope of J2EE is so large that most people specialize in a particular area. Sections of the rest of this book will deal with two of the major services, EJB and JDBC.
Tomcat is free software from the Apache Software Foundation, which readily provides the source code and binaries (see [www.apache.org](http://www.apache.org) and [http://jakarta.apache.org](http://jakarta.apache.org)). Tomcat and other Apache projects are maintained and developed by open-source developers, people contributing to the Apache software community by working in their spare time. These developers do not earn money for their work but share in the successes of their contributions.

Apache is one of the leading web servers on the internet. An October 2001 survey discovered that 56 percent of the sites on the Internet were being served up by Apache.

Jakarta is the group name for Java server software developed by the Apache Software Foundation. Their list of web server extensions continues to grow and includes valuable tools that enhance Java, notably the following:

- **Ant 1.4.1**, a tool for automating the building of applications; extremely helpful for compiling complex applications composed of numerous files
- **Cactus 1.2**, a framework for the automatic testing of server-side code
- **ECS 1.4.1**, a Java package and API (applications programming interface) for generating and formatting the elements for HTML 4, XML, and others
- **James 1.2.1**, the Java Apache Mail Enterprise Server
- **Jetspeed 1.3a2**, an open-source information portal
- **JMeter 1.5**, a pure Java desktop application for testing
- **log4j 1.1.3**, software that allows logging to be turned on at runtime
- **ORO 2.0.4**, a library that provides text-processing Java classes similar to Perl, AWK, and more
- **Slide 1.0.16**, WebDAV (WWW Distributed Authoring and Versioning) modules; for remote website maintenance and authoring
- **Struts 1.0.1**, a revolutionary application framework for providing clean Model-View-Controller designs
- **Turbine 2.1**, a secure servlet framework

Just as the Apache web server has revolutionized the Web, we expect the Jakarta subprojects to engender major, industry-wide changes.

**iPlanet/Netscape**

Netscape merged with iPlanet, and both companies are owned by Sun Microsystems, the creators of Java. iPlanet provides development environments and enterprise application servers. iPlanet is an excellent choice when funding is available. Their products make extensive use of XML.

**WebLogic**

WebLogic was one of the first companies to make a big name in the J2EE application server market. Even though it can be prohibitively expensive, it has remained popular with developers and businesses largely due to its reputation for providing a robust environment.
**Web-Sphere**

Web-Sphere is IBM’s J2EE system. IBM is a major Java player, and WebSphere has proven itself by supporting some extremely large sites, such as eBay.

**Oracle9i Application Server**

With the release of its 9i application server, heavyweight competitor Oracle is offering J2EE support to augment its position as one of the premier enterprise database companies. Oracle is only continuing to expand its support of Java and XML, and the future looks very promising.

**JBoss**

Sun and IBM have declared J2EE development an open-industry process. Both companies compete with their own proprietary (closed-source) J2EE application servers. An excellent open-source J2EE solution is emerging: JBoss. JBoss is a complete J2EE solution that provides an EJB container, JMS messaging, mail services, JTS services, security services, connectivity, and persistence. Additionally, JBoss has, unbelievably, several features that aren’t available in the other J2EE application servers. The only problem with JBoss is that it awaits Sun certification. As a result, the business rules of some companies will not yet allow them to support their enterprise architecture using it.

**Fast-Brewing Java on the Client Side**

Compiling a small Java application from a command-line window is easy enough, but even talented programmers have difficulty guessing how a graphical layout will look. Running the application after every visible change is a horribly inefficient use of time. An IDE eases the pain of nontrivial development tasks by providing a visual interface for developers to navigate their projects, dragging and dropping components onto visual layout managers.

Most IDEs provide all the features a developer needs to code, debug, test, package, and deploy standalone applications, web applications, libraries, and anything else a developer would want.

**Warning** An IDE lets you work faster and potentially learn more; however, it’s important not to lose sight of what the IDE is doing and which commands it is running. Otherwise, your understanding might become superficial, your visualization skills might be impaired, and you might feel helpless if you are called on to debug your application with a minimum of tools, as in a server room without the comfortable, cute graphics of your favorite IDE.

Because developers spend so much time with IDEs, often looking at them with the same microscopic vision they train on obscure passages of code, it’s inevitable that annoying bugs will crop up and be noticed.

Initially, Sun released Java as a language and encouraged other companies to create tools and development environments. Although all the IDEs contributed helpful features, many of them surreptitiously bundled in and encouraged the use of additional, semi-proprietary code, all to encourage loyalty to and dependency on the parent company. For example, one early IDE allowed users to build fairly complex applets by drag-and-drop and using interactive wizards. It all worked beautifully in the IDE, but when users tried to package and deploy the applet, they found that proprietary mouse-handler classes and a slew of other unnecessary class libraries had to be included. The applet usually became too large for an impatient user waiting with a modem. Many
of these horrible “planted” bugs should have been worked out by now, but a clear lesson remains: be aware of the possible hidden price of time-saving routines, and always check to see what your IDE is doing under the covers.

Often in the working world, the choice of an IDE is moot because businesses standardize their development on a particular product. However, businesses do change their minds, and sometimes management listens to developers. Since so much of your time will probably be spent in front of an IDE, it’s important to watch for changes and new features that, given some effort to learn, might make your work easier. By all means, download some of the following, even just the trial versions, to keep yourself familiar with what is possible.

**NOTE** Since the URLs for downloading sample IDEs often change, we’ll reference only the vendor. Search for the vendor’s site on the Web, and choose the appropriate IDE for your needs.

**JBuilder**

JBuilder is a great IDE because it is flexible and gives users many choices. For example, the web server it uses can be upgraded to a newer version of Tomcat. By default, JBuilder uses Tomcat as its web server. Figure 1.2 shows the JBuilder IDE.

**NOTE** You can easily run all the book’s sample project files on the CD-ROM under JBuilder, and almost all the illustrations in the book will use JBuilder. Of course, you can also run the projects and code on a standalone installation of Tomcat. We’ll show you how to package web applications for individual Tomcat deployments in Appendix A.
You can use the JBuilder IDE to preview basic HTML layout, as shown in Figure 1.3. However, JavaScript code doesn’t always work in the window, and server-side code won’t display properly since it requires a running Java environment to provide the dynamic content.

The JBuilder IDE has some useful features. For example, each source file has integrated graphical source control, as shown in Figure 1.4.
JBuilder also provides extensive debugging information, as shown in Figure 1.5.

**Figure 1.5**
JBuilder debugging information

Notice the Query String parameter in Figure 1.5. If the page had HTML inputs inside a pair of `<FORM>` tags, the inputs and their values would also be shown, as in Figure 1.6. Here, you can see that the JBuilder debug window shows everything the user submits and what happens to those values along the way.

**Figure 1.6**
JBuilder debugging information in detail
**JDeveloper**

JDeveloper is a free IDE that Oracle released to promote the use of Java and Oracle. Oracle is a big supporter of Java, and in fact the Oracle software installers and the GUIs are written in Java.

The JDeveloper IDE is similar to an older version of Borland’s JBuilder, but whatever its origins, the latest release encompasses all aspects of JavaServer programming, and the IDE is tailored with built-in functionality to interact with Oracle applications and databases. Also, the IDE comes with an impressive number of supporting libraries.

**Forte for Java**

Forte is Sun’s late entry into the Java IDE arena. Sun bought Forte from a private company and released a version to the public. You can choose between using a single document interface (SDI) and a multiple document interface (MDI). Figure 1.7 shows the windows docked together and attached to the toolbar.

![The Forte IDE](image)

**Figure 1.7**
The Forte IDE

Unlike Sun’s Forte for C++, Forte for Java is free. A Forte Internet edition, with many useful features for programming JavaServer code, is available for a considerable fee. The free version comes with many useful options found in more expensive IDEs. Figure 1.8 shows the Forte Object Browser, useful for finding method names inside packages.
The expanse of Forte is impressive; however, many users find it slow unless run on a computer with lots of memory, so your mileage may vary. Under similar stresses, JBuilder seems to fair better.

**Visual Café**

Visual Café is another Java IDE. Over several releases, it has maintained a high-quality user interface with extensive assistance for developers. Unfortunately, its price has been prohibitively high.

**Visual J+/Visual J++/Visual J# .net**

Visual J is Microsoft’s Java IDE. We mention it here cursorily because it still seems to persist, albeit in some sort of mangled form, and JSP developers are likely to run into it somewhere on an old machine. There is still great suspicion about what Microsoft is trying to do with Java. The Sun vs. Microsoft Java lawsuit, after years of litigation, was settled out of court with Microsoft paying somewhere in the neighborhood of 20 million dollars. Fortunately, you can now choose from many alternative IDEs.

**Other IDEs**

The previous list is not exhaustive, and new, helpful additions will surely appear. If you ever find yourself frustrated with your current IDE, give one of the others a try, or surf around and see what’s new.
What You Need to Get Started

For the examples in this book, we'll use Apache's Tomcat. Tomcat has been chosen as Sun's reference implementation for the latest JSP specification (1.2). Tomcat is also free and open source, so you can't complain about the price, and if you want to know what's really going on, you can browse its source. Better yet, if you have any spare time, you can contribute to the Tomcat project or to any of the other Jakarta projects.

NOTE  Apache is perhaps the most successful free software endeavor, with its web server dominating the World Wide Web. Its Java projects lead the industry in setting standards. Time spent with Apache software is an excellent investment.

To install Tomcat, you need about 34MB of disk space and an operating system that includes a Java 2 release (Windows, Linux, Solaris, Macintosh, and so on). It's recommended that you also install the Java Development Kit (JDK) for your system so that you'll have access to the additional tools, javadoc, the debugger, and so on.

Summary

Java grew out of its initial appearance in cute applets to become the ideal solution for server-side web programming. Compared with other scripting and web programming languages, Java is as good or better. JSP can leverage the use of server-side objects and even embed Java code right into the script of the web page—and it will still run on any platform.

JSP pages and servlets are inextricably linked. JSP pages make the Presentation layer more manageable, and servlets shore up the processing and facilitate lean, economical communication with other enterprise entities such as databases, messaging services, transaction services, and just about anything else related to the information needs of a business.

Today, you can choose from several JavaServer environments, and more are becoming available every day. Price is no longer an issue, because open-source development has made Tomcat web server available free, and there's even a free J2EE Java environment—JBoss. And with Forte from Sun, it's possible to get a Java IDE for free.