TRAINING & REFERENCE

murach's Java servlets and JSP

(Chapter 1)

Andrea Steelman
Joel Murach

Mike Murach & Associates
2560 West Shaw Lane, Suite 101
Fresno, CA 93711-2765
(559) 440-9071 • (800) 221-5528
murachbooks@murach.com • www.murach.com

Section 1

Introduction to servlet and JSP programming

The three chapters in this section provide the background information that you need for writing servlets and JavaServer Pages (JSPs). In chapter 1, you'll learn what web programming is and how servlets and JSPs work. In chapter 2, you'll learn how to install and use the Tomcat software for running servlets and JSPs on your own PC. And in chapter 3, you'll learn how to develop the HTML documents that pass data to the servlets and JSPs of a web application.

1

An introduction to web programming with Java

This chapter introduces you to the concepts and terms that you need for working with servlets and JavaServer Pages (JSPs) as you create web applications. In particular, this chapter introduces you to the software that you'll be working with.

An introduction to web applications

A *web application* is a set of web pages that are generated in response to user requests. The Internet has many different types of web applications, such as search engines, online stores, auctions, news sites, discussion groups, and games.

A typical web application

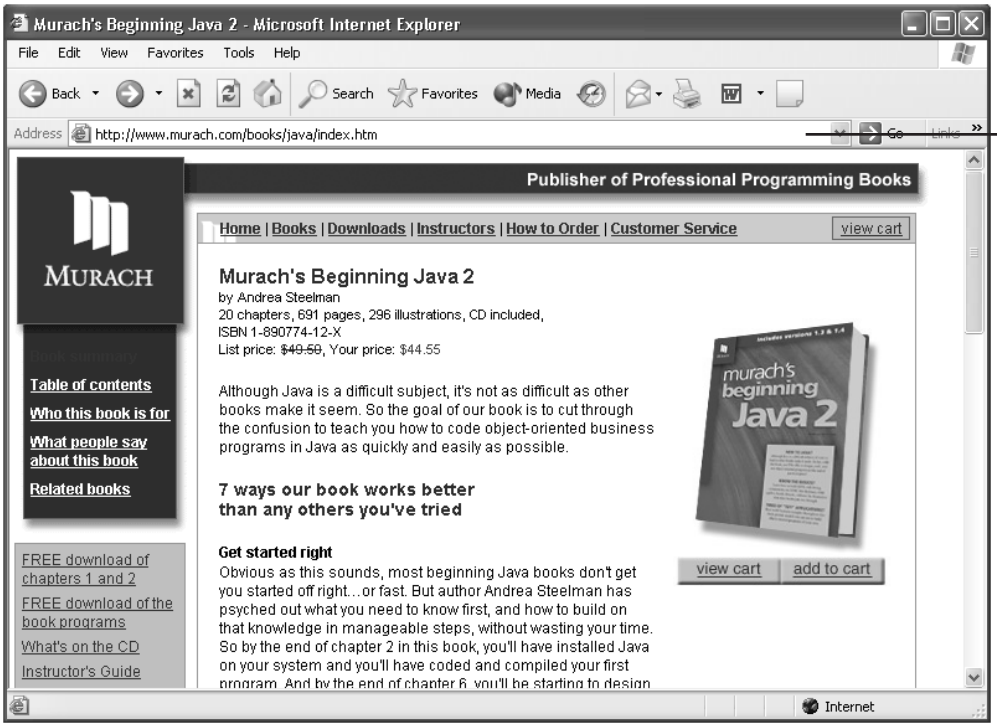
Figure 1-1 shows the first two pages of the shopping cart application that's available from *www.murach.com*. Here, the first page presents some information about our beginning Java book. This page contains two buttons: a View Cart button and an Add To Cart button. When you click the Add To Cart button, the web application adds the book to your cart and displays the second page in this figure, which shows all of the items in your cart.

The second page lets you change the quantity for an item or remove an item from the cart. It also lets you continue shopping or begin the checkout process. In this book, you'll learn all the skills you need to create a shopping cart application like this one.

If you take a closer look at these web pages, you can learn a little bit about how this application works. For the first page, the Address box of the browser shows an address that has an htm extension. This means that the page was rendered from an HTML document.

In contrast, the Address box for the second page shows the address of a servlet named ShoppingServlet that's stored in the murach.ordering package. This means that the HTML code for this page was generated by the servlet. After the servlet address, you can see a question mark and one parameter named productCode that has a value of "java". This is the parameter that was passed from the first page. You'll learn how this navigation from page to page works in chapter 3.

The first page of a shopping cart application



The second page of a shopping cart application

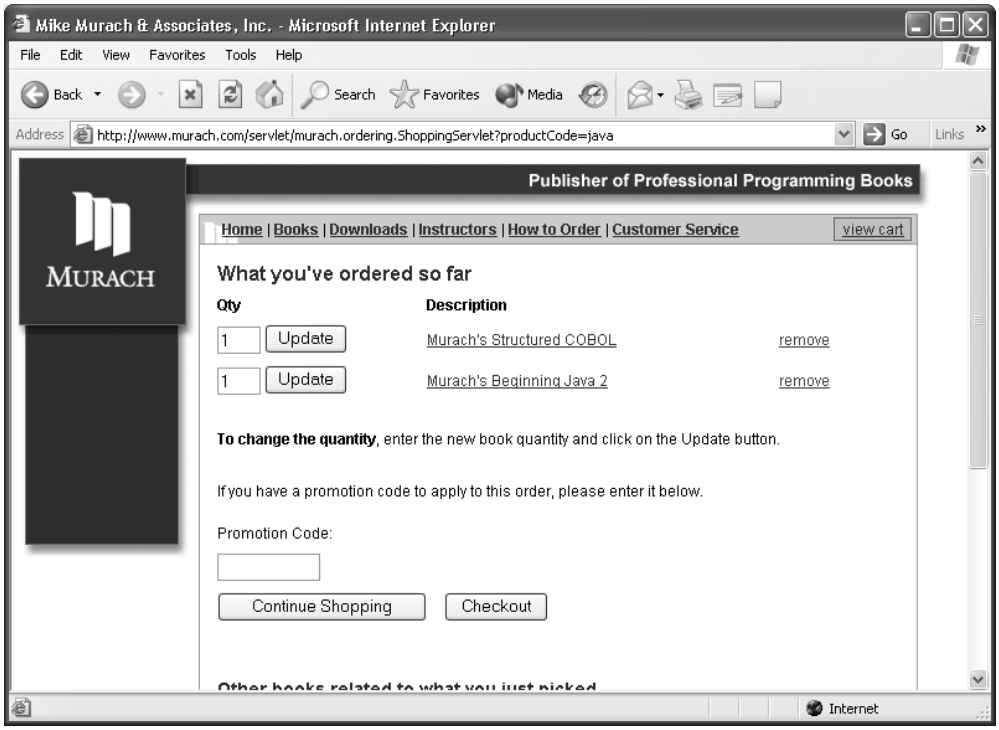


Figure 1-1 A typical web application

The components of a web application

Figure 1-2 shows the basic components that make up a web application. Because a web application is a type of *client/server application*, the components of a web application are stored on either the *client* computer or the *server* computer.

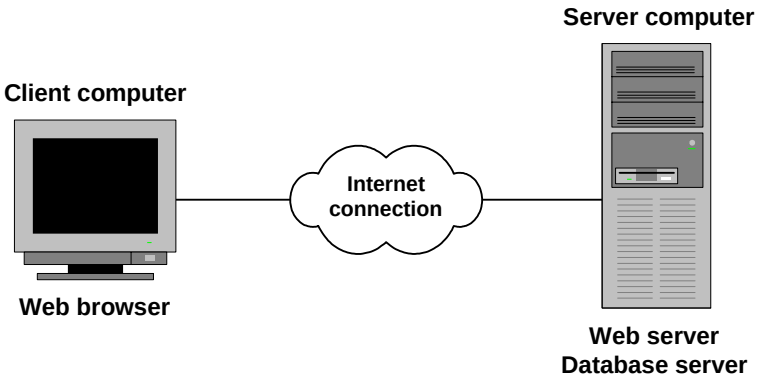
To access a web application, you use a *web browser* that runs on a client computer. It is the browser that converts HTML code to the user interface. The most widely used web browser is Microsoft's Internet Explorer, and the most popular alternative is Netscape's Navigator (commonly known as Netscape).

The web application itself is stored on the server computer. This computer runs *web server* software that enables it to send web pages to web browsers. Although there are many web servers, the most popular one for Java web applications is the Apache Software Foundation's *Apache HTTP Server*, which is usually just called *Apache*.

Because most web applications work with data that's stored in a database, most servers also run a *database management system* (or *DBMS*). Two of the most popular for Java development are *Oracle* and *MySQL*. Note, however, that the DBMS doesn't have to run on the same server as the web server software. In fact, a separate database server is often used to improve an application's overall performance.

Although this figure shows the client and server computers connected via the Internet, this isn't the only way a client can connect to a server in a web application. If the client and the server are on the same *Local Area Network*, or *LAN*, they function as an *intranet*. Since an intranet uses the same protocols as the Internet, a web application works the same on an intranet as it does on the Internet.

Components of a web application



Description

- Web applications are a type of *client/server application*. In a client/server application, a user at a *client* computer accesses an application at a *server* computer. For a web application, the client and server computers are connected via the Internet or an intranet.
- In a web application, the user works with a *web browser* at the client computer. The web browser provides the user interface for the application. The most commonly used web browser is Microsoft's Internet Explorer, but other web browsers such as Netscape's Navigator are also in use.
- A web application runs on the server computer under the control of *web server* software. For servlet and JSP applications, the *Apache* server is the most widely used web server.
- For most web applications, the server computer also runs a *database management system* (*DBMS*). For servlet and JSP applications, *Oracle* and *MySQL* are two of the most popular database management systems.

Figure 1-2 The components of a web application

How static web pages work

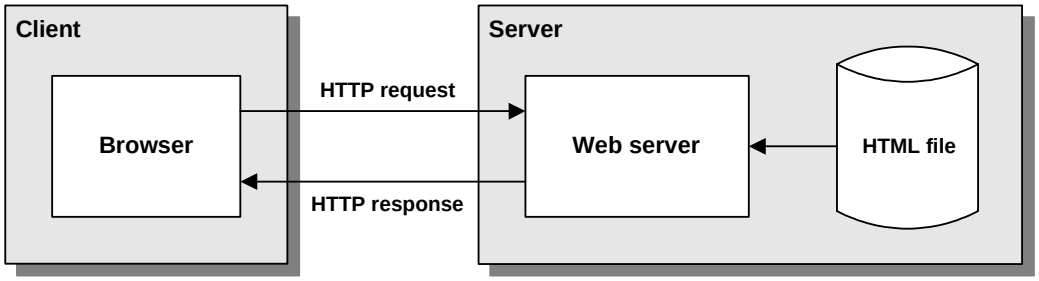
HTML, or *Hypertext Markup Language*, is the language that the browser converts to the web pages that make up a web application's user interface. Many of these web pages are *static web pages*, which are the same each time they are viewed. In other words, they don't change in response to user input.

Figure 1-3 shows how a web server handles static web pages. The process begins when a user at a web browser requests a web page. This can occur when the user enters a web address into the browser's Address box or when the user clicks a link that leads to another page. In either case, the web browser uses a standard Internet protocol known as *Hypertext Transfer Protocol*, or *HTTP*, to send a request known as an *HTTP request* to the web site's server.

When the web server receives an HTTP request from a browser, the server retrieves the requested HTML file from disk and sends the file back to the browser in the form of an *HTTP response*. The HTTP response includes the HTML document that the user requested along with any other resources specified by the HTML code such as graphics files.

When the browser receives the HTTP response, it formats and displays the HTML document. Then, the user can view the content. If the user requests another page, either by clicking a link or typing another web address in the browser's Address box, the process begins again.

How a web server processes static web pages



Description

- *Hypertext Markup Language*, or *HTML*, is the language that the web browser converts into the web pages of a web application.
- A *static web page* is an HTML document that's stored in a file and does not change in response to user input. Static web pages have a filename with an extension of .htm or .html.
- *Hypertext Transfer Protocol*, or *HTTP*, is the protocol that web browsers and web servers use to communicate.
- A web browser requests a page from a web server by sending the server a message known as an *HTTP request*. For a static web page, the HTTP request includes the name of the HTML file that's requested.
- A web server replies to an HTTP request by sending a message known as an *HTTP response* back to the browser. For a static web page, the HTTP response includes the HTML document.

Figure 1-3 How static web pages work

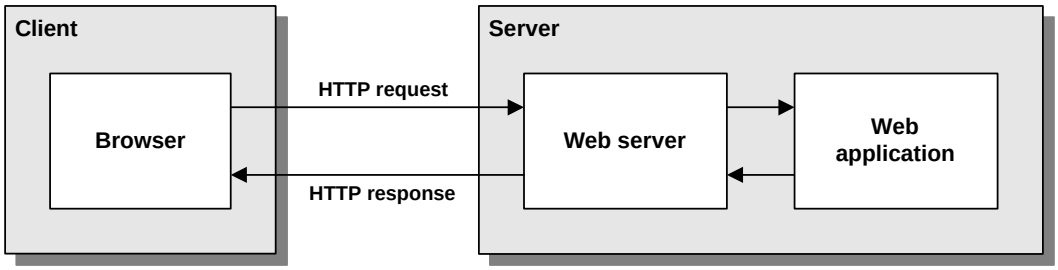
How dynamic web pages work

In contrast to a static web page, a *dynamic web page* changes based on the parameters that are sent to the web application from another page. For instance, when the Add To Cart button in the first page in figure 1-1 is clicked, the static web page calls the web application and sends one parameter to it. Then, the web application generates the HTML for a new web page and sends it back to the browser.

Figure 1-4 shows how this works. When a user enters data into a web page and clicks the appropriate button, the browser sends an HTTP request to the server. This request contains the address of the next web page along with any data entered by the user. Then, when the web server receives this request and determines that it is a request for a dynamic web page, it passes the request back to the web application.

When the web application receives the request, it processes the data that the user entered and generates an HTML document. Next, it sends that document to the web server, which sends the document back to the browser in the form of an HTTP response. Then, the browser displays the HTML document that's included in the response so the process can start over again.

How a web server processes dynamic web pages



Description

- A *dynamic web page* is an HTML document that's generated by a web application. Often, the web page changes according to parameters that are sent to the web application by the web browser.
- When a web server receives a request for a dynamic web page, the server passes the request to the web application. Then, the application generates a response, which is usually an HTML document, and returns it to the web server. The web server, in turn, wraps the generated HTML document in an HTTP response and sends it back to the browser.
- The browser doesn't know or care whether the HTML was retrieved from a static HTML file or was dynamically generated by the web application. Either way, the browser displays the HTML document that is returned.

Figure 1-4 How dynamic web pages work

An introduction to Java web programming

In the early days of Java, Java received much attention for its ability to create *applets*, which are Java applications that can be downloaded from a web site and run within a web browser. However, once Microsoft's Internet Explorer stopped supporting new versions of Java, applets lost much of their appeal. As a result, many developers switched their attention to *servlets* and *JavaServer Pages*, or *JSPs*. These technologies allow developers to write Java web applications that run on the server.

The components of a Java web application

Figure 1-5 shows the primary software components for a Java web application. By now, you should understand why the server must run web server software. To run a Java application, though, the server must also run a software product known as a *servlet and JSP engine*, or *servlet and JSP container*. This software allows a web server to run servlets and JSPs.

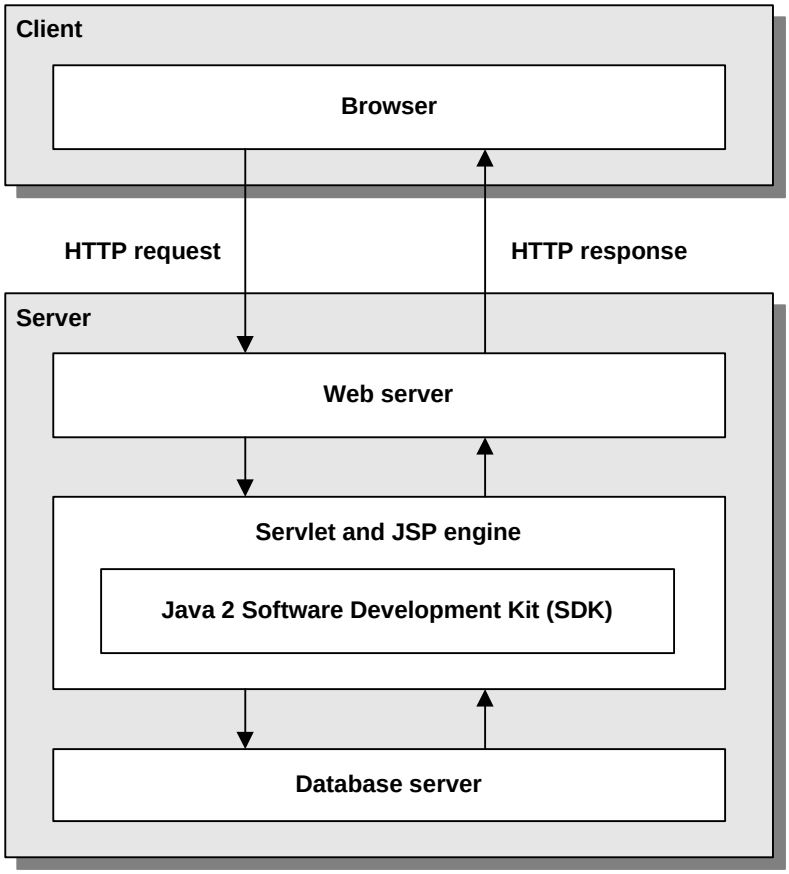
Sun's *Java 2 Platform, Enterprise Edition*, or *J2EE*, specifies how a servlet and JSP engine should interact with a web server. Since this specification is available to all developers, most modern web servers support any servlet and JSP engine that implements this specification. In addition, all servlet and JSP engines should work similarly. In theory, this makes servlet and JSP code portable between servlet and JSP engines and web servers. In practice, though, you may need to make some modifications to your code when switching servlet and JSP engines or web servers.

Tomcat is a free, open-source servlet and JSP engine that was developed by the Jakarta project at the Apache Software Foundation. This engine is the official reference implementation of the servlet and JSP specification set forth by Sun, and it's one of the most popular servlet and JSP engines. In the next chapter, you'll learn how to install and use Tomcat on your own PC.

For a servlet and JSP engine to work properly, the engine must be able to access Java's *Software Development Kit*, or *SDK*, that comes as part of the *Java 2 Platform, Standard Edition*, or *J2SE*. The SDK contains the Java compiler and the core classes for working with Java. It also contains the *Java Runtime Environment*, or *JRE*, that you need for running the Java classes. Since this book assumes that you already have some Java experience, you should already be familiar with the SDK and the JRE.

Many large websites also use a Java technology known as *Enterprise JavaBeans*, or *EJBs*. To use EJBs, the server must run an additional piece of software known as an *EJB server*, or *EJB container*. Although there are some benefits to using EJBs, they're more difficult to use when you're first learning how to code Java web applications. That's why this book focuses on developing web applications without using EJBs. In chapter 17, though, you'll get an introduction to EJBs.

The components of a Java web application



Description

- Java web applications consist of JavaServer Pages and servlets. You'll learn more about them in the next two figures.
- A *servlet and JSP engine*, or *servlet and JSP container*, is the software that allows the web server to work with servlets and JSPs.
- The *Java 2 Platform, Enterprise Edition*, or *J2EE*, specifies how web servers can interact with servlet and JSP engines. *Tomcat* is one of the most popular servlet and JSP engines. It was developed by the Jakarta project at the Apache Software Foundation.
- For a servlet and JSP engine to work, it must have access to Java's *Software Development Kit*, or *SDK*, which comes as part of the *Java 2 Platform, Standard Edition*, or *J2SE*. Among other things, the SDK contains the Java classes, the Java compiler, and a *Java Runtime Environment*, or *JRE*.
- Java web applications that use *Enterprise JavaBeans*, or *EJBs*, require an additional server component known as an *EJB server*, or *EJB container*. For more information about working with EJBs, see chapter 17.

Figure 1-5 The components of a Java web application

An introduction to JavaServer Pages

One of the goals of this book is to teach you how to develop the *JavaServer Pages*, or *JSPs*, that are part of a web application. To give you a better idea of what they are, figure 1-6 shows part of a simple JSP that generates the web page shown in the browser.

If you look at the Address box of the browser in this figure, you can see that the address of a JSP ends with a jsp extension. After that, you can see a question mark followed by the parameters that are passed to the JSP.

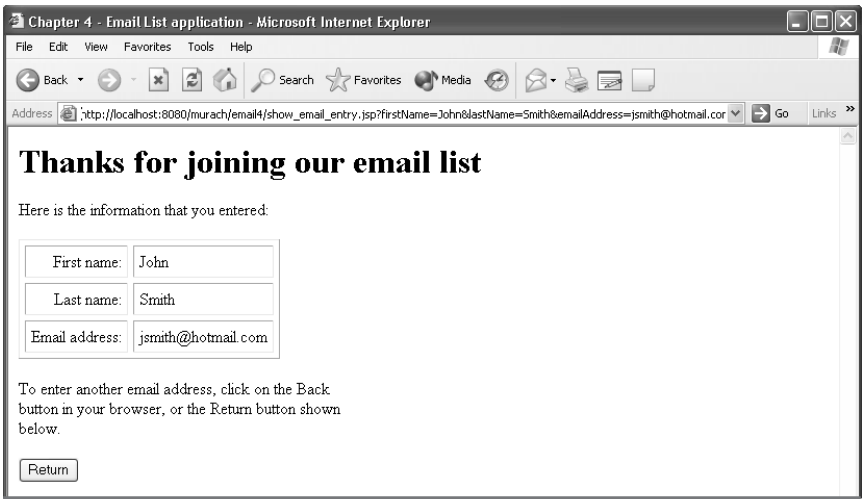
If you're already familiar with HTML, you can see that most of this JSP consists of HTML code. In fact, the only Java code in this JSP is shaded. That's why JSPs are relatively easy to write if you know HTML and keep the Java to a minimum. If a JSP requires extensive Java programming, though, it's easier to write the Java code with a servlet. In practice, web designers often write the HTML portions of the JSPs, while web programmers write the Java portions.

In case you're interested, the first three lines of Java code in this JSP get three parameters from the request object that has been passed to it. To do that, the code uses the getParameter method of the object. Then, the three Java expressions that are used later in the JSP refer to the String variables that store the parameters.

When a JSP is requested for the first time, the *JSP engine* (which is part of the servlet and JSP engine) converts the JSP code into a servlet and compiles the servlet. Then, the JSP engine loads that servlet into the servlet engine, which runs it. For subsequent requests, the JSP engine runs the servlet that corresponds to the JSP.

In chapter 3, you'll get a crash course in HTML that will teach you all the HTML you need to know for writing JSPs. Then, in chapter 4, you'll learn how to combine HTML code with Java code as you write JSPs. When you're done with those chapters, you'll know how to write significant JSPs of your own.

A web page that's returned from a JSP



Partial code for the JSP

```
<head>
  <title>Chapter 4 - Email List application</title>
</head>
<body>
<%
  String firstName = request.getParameter("firstName");
  String lastName = request.getParameter("lastName");
  String emailAddress = request.getParameter("emailAddress");
%>
<h1>Thanks for joining our email list</h1>
<p>Here is the information that you entered:</p>
  <table cellspacing="5" cellpadding="5" border="1">
    <tr>
      <td align="right">First name:</td>
      <td><%= firstName %></td>
    </tr>
    <tr>
      <td align="right">Last name:</td>
      <td><%= lastName %></td>
    </tr>
    <tr>
      <td align="right">Email address:</td>
      <td><%= emailAddress %></td>
    </tr>
  </table>
```

Description

- A *JavaServer Page*, or *JSP*, consists of Java code that is embedded within HTML code. This makes it easy to write the HTML portion of a JSP, but harder to write the Java code.
- When a JSP is first requested, the JSP engine translates it into a servlet and compiles it. Then, the servlet is run by the servlet engine.
- In chapter 3, you'll learn how to create HTML documents. In chapter 4, you'll learn how to create JSPs.

Figure 1-6 An introduction to JavaServer Pages

An introduction to servlets

A second goal of this book is to teach you how to develop the *servlets* that are part of a Java web application. To give you a better idea of what they are, figure 1-7 shows part of a servlet that generates the same web page as the JSP in figure 1-6.

In contrast to the JSP, the servlet is a Java class that runs on a server and does the processing for the dynamic web pages of a web application. That's why servlets for a web application are written by web programmers, not web designers. After the processing is done, the servlet returns HTML code to the browser by using the println method of an out object. Note, however, that this makes it more difficult to code the HTML.

In case you're interested, each servlet for a web application is a Java class that extends (or inherits) the HttpServlet class. Then, each servlet can override the doGet method of the inherited class, which receives both a request and a response object from the web server, and the servlet can get the parameters that have been passed to it by using the getParameter method of the request object. After that, the servlet can do whatever processing is required by using normal Java code.

In chapter 5, you'll get the details on how servlets work. When you complete that chapter, you'll be able to write significant servlets of our own. But as you'll learn next, the real trick as you develop Java web applications is to use a combination of servlets and JSPs so you get the benefits of both.

How to combine servlets and JSPs in a web application

As you have seen, servlets are actually Java classes so it makes sense to use them for the processing requirements of the web pages in an application. Similarly, JSPs are primarily HTML code so it makes sense to use them for the design of the web pages in an application. But how can you do that in an efficient way?

The solution is for a servlet to do the processing for each web page, and then forward the request and response objects to the JSP for the page. That way, the servlet does all the processing, and the JSP does all of the HTML. With this approach, the JSP requires a minimum of embedded Java code. And that means that the web designer can write the JSPs with little or no help from the Java programmer, and the Java programmer can write the servlets without worrying about the HTML.

In chapter 6, you'll learn how to use this approach for developing web applications. You'll also learn how to use the Model-Controller-View pattern to structure your applications so they're easy to manage and maintain. When you finish that chapter, you'll know how to develop Java web applications in a thoroughly professional manner.

Partial code for a servlet that works the same as the JSP in figure 3-6

```
public class EmailServlet extends HttpServlet{

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws IOException, ServletException{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String firstName = request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String emailAddress = request.getParameter("emailAddress");

        out.println(
          "<html>\n"
        + "<head>\n"
        + "  <title>Chapter 5 - Email List application</title>\n"
        + "</head>\n"
        + "<body>\n"
        + "<h1>Thanks for joining our email list</h1>\n"
        + "<p>Here is the information that you entered:</p>\n"
        + "  <table cellspacing=\"5\" cellpadding=\"5\" border=\"1\">\n"
        + "  <tr><td align=\"right\">First name:</td>\n"
        + "      <td>" + firstName + "</td>\n"
        + "  </tr>\n"
        + "  <tr><td align=\"right\">Last name:</td>\n"
        + "      <td>" + lastName + "</td>\n"
        + "  </tr>\n"
        + "  <tr><td align=\"right\">Email address:</td>\n"
        + "      <td>" + emailAddress + "</td>\n"
        + "  </tr>\n"
        + "  </table>\n"
        + "</html>);
```

Description

- A *servlet* is a Java class that runs on a server. Although servlets are commonly used for web applications, they can also be used for other types of applications like mail or FTP server applications.
- A servlet for a web application extends the HttpServlet class. This makes it easy for a Java programmer to write the code for the processing that a web page requires. Once a servlet is compiled, it can be run by the servlet engine.
- To return HTML code to the browser, a servlet uses the println method of an out object. This makes it more difficult to write the HTML portion of the code.
- To get the best results from servlets and JSPs, you use a combination of the two as you develop web pages. In particular, you use servlets for the processing that's required by the pages, and JSPs for the HTML that's required by the pages.
- In chapter 5, you'll learn how to create servlets. And in chapter 6, you'll learn how to structure and code your web pages so they take advantage of the best features of both servlets and JSPs.

Figure 1-7 An introduction to servlets

An introduction to Java web development

This topic introduces you to servlet and JSP development. In particular, it presents some of the hardware and software options that you have as you develop web applications.

Three platforms for developing servlets and JSPs

Figure 1-8 shows the three types of platforms that you can use for developing servlets and JSPs. First, you can use a single PC. Second, you can use a Local Area Network (or LAN). Third, you can use the Internet.

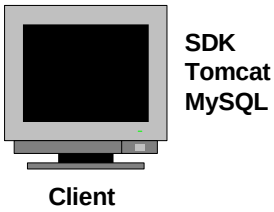
When you use a single PC, you of course need to install all of the required software on that PC. That includes the Java SDK, the web server software, the servlet and JSP engine, and the database management system. To make this easy for you, the CD for this book includes everything that you need. In particular, it includes Tomcat, which functions as both a web server and a servlet and JSP engine. It also includes MySQL, which is a DBMS. In the next chapter, you'll learn how to install Tomcat, and you can learn how to install the other components in appendix A.

When you work on a LAN, you can use the same software components, but you divide them between client and server. To compile and run servlets on the server, the server requires the Java SDK, a combined web server and servlet and JSP engine like Tomcat, and a DBMS like MySQL. To compile servlets on a client, the client requires the Java SDK and the servlet.jar file, which contains all of the classes required for servlet development in a compressed format. This JAR file comes with Tomcat, but you can also download it for free from the Sun web site. When you run a web application on a LAN like this, it functions as an intranet.

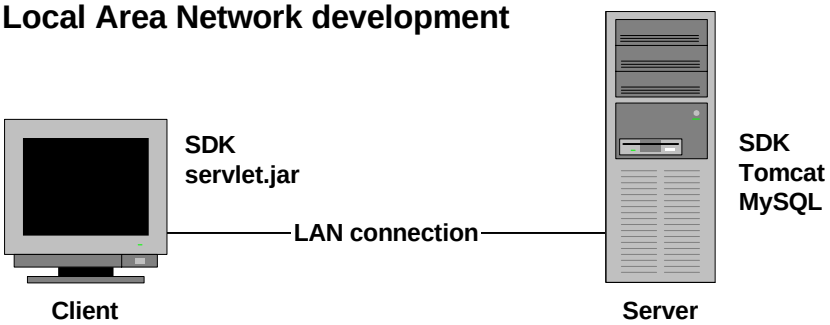
When you work over the Internet, you use the same general components as you do when you work over an intranet. To improve performance, though, you normally have a separate web server like Apache in addition to a servlet and JSP engine like Tomcat. If necessary, you can also improve the performance of an intranet application by using Apache as the web server.

Since the SDK, Apache, Tomcat, and MySQL can be run by most operating systems, Java web developers aren't tied to a specific operating system. In fact, the Windows operating system is commonly used for both the client and server computers during development. But when the applications are ready for use, they are often deployed on a Unix or Solaris server.

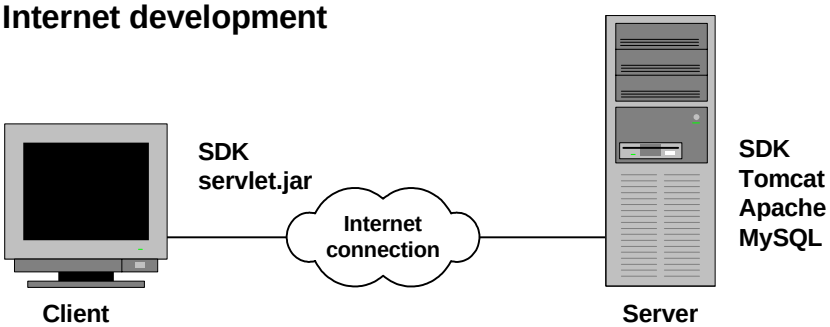
Stand-alone development



Local Area Network development



Internet development



Description

- When you develop web applications, you can work on platforms at three levels.
- If you want to develop web applications on your own PC, you need to install the Java SDK, a web server, a servlet and JSP engine, and a DBMS. To make that easy for you, the CD that comes with this book includes everything you need including the MySQL DBMS and Tomcat 4.0, which functions as both a web server and a servlet and JSP engine.
- If you're working in a small group on a Local Area Network (LAN), the server can run Tomcat as both the web server and the servlet and JSP engine, and MySQL as the DBMS. Then, the client just needs the Java SDK and the servlet.jar file, which isn't part of the SDK. When you run web applications on a LAN, it functions as an intranet.
- If you're working in a group over the Internet, you normally use a product like Apache as the web server and a product like Tomcat as just the servlet and JSP engine. Otherwise, this works the same as when you're working over a LAN.

Figure 1-8 Three platforms for developing servlets and JSPs

The architecture for servlet and JSP applications

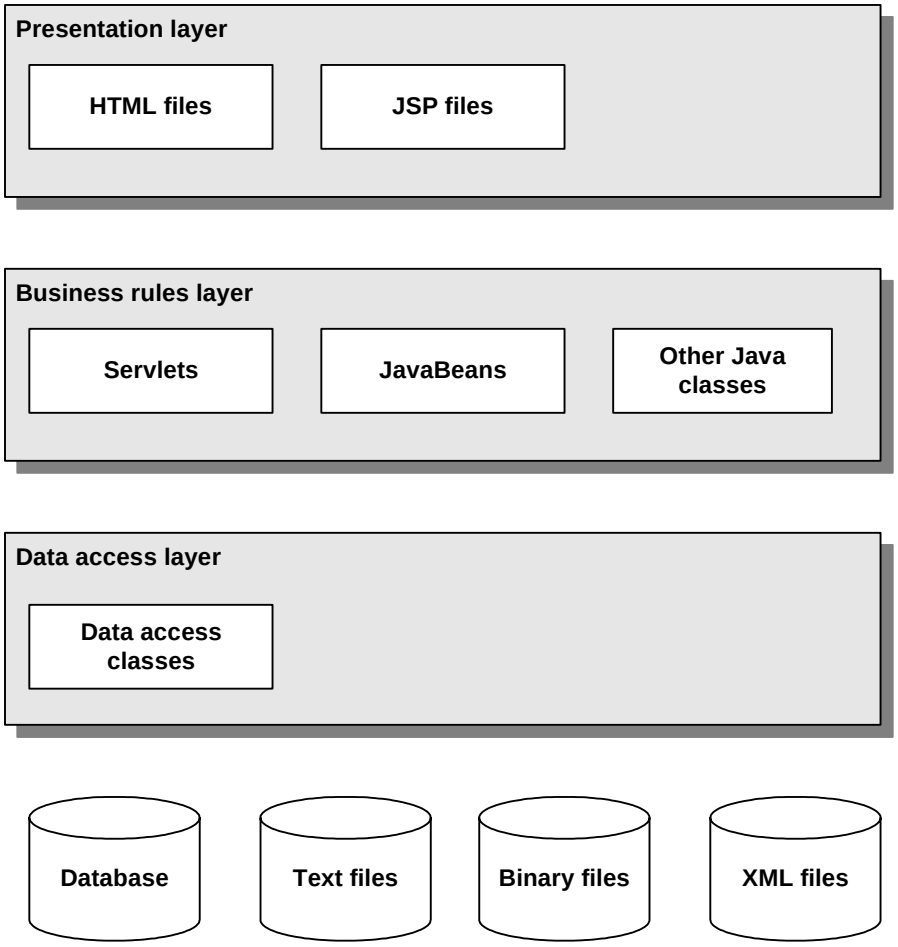
Figure 1-9 shows the architecture for a typical web application that uses servlets and JSPs. This architecture uses three layers: (1) the *presentation layer*, or *user interface layer*, (2) the *business rules layer*, and (3) the *data access layer*. In theory, the programmer tries to keep these layers as separate and independent as possible. In practice, though, these layers are often interrelated, and that's especially true for the business and data access layers.

The presentation layer consists of HTML pages and JSPs. Typically, a web designer will work on the HTML stored in these pages to create the look and feel of the user interface. Later, a Java programmer may need to edit these pages so they work properly with the servlets of the application.

The business rules layer uses servlets to control the flow of the application. These servlets may call other Java classes to store or retrieve data from a database, and they may forward the results to a JSP or to another servlet. Within the business layer, Java programmers often use a special type of Java class known as a *JavaBean* to temporarily store and process data. A JavaBean is typically used to define a business object such as a User or Invoice object.

The data layer works with data that's stored on the server's disk. For a serious web application, this data is usually stored in a relational database. However, this data can also be stored in text files and binary files. In addition, the data for an application can be stored in a special type of file that uses *Extensible Markup Language*, or *XML*. You'll learn more about working with XML in chapter 16.

The architecture for a typical Java web application



Description

- The *presentation layer* for a typical Java web application consists of HTML pages and JSPs.
- The *business rules layer* for a typical Java web application consists of servlets. These servlets may call other Java classes including a special type of Java class known as a *JavaBean*. Chapter 8 will show how to work with JavaBeans.
- The *data access layer* for a typical Java web application consists of classes that read and write data that's stored on the server's disk drive.
- For a serious web application, the data is usually stored in a relational database. However, it may also be stored in binary files, in text files, or in *Extensible Markup Language* (or *XML*) files. Chapter 16 will show how to work with XML.

Figure 1-9 The architecture for servlet and JSP applications

Tools for writing servlets and JSPs

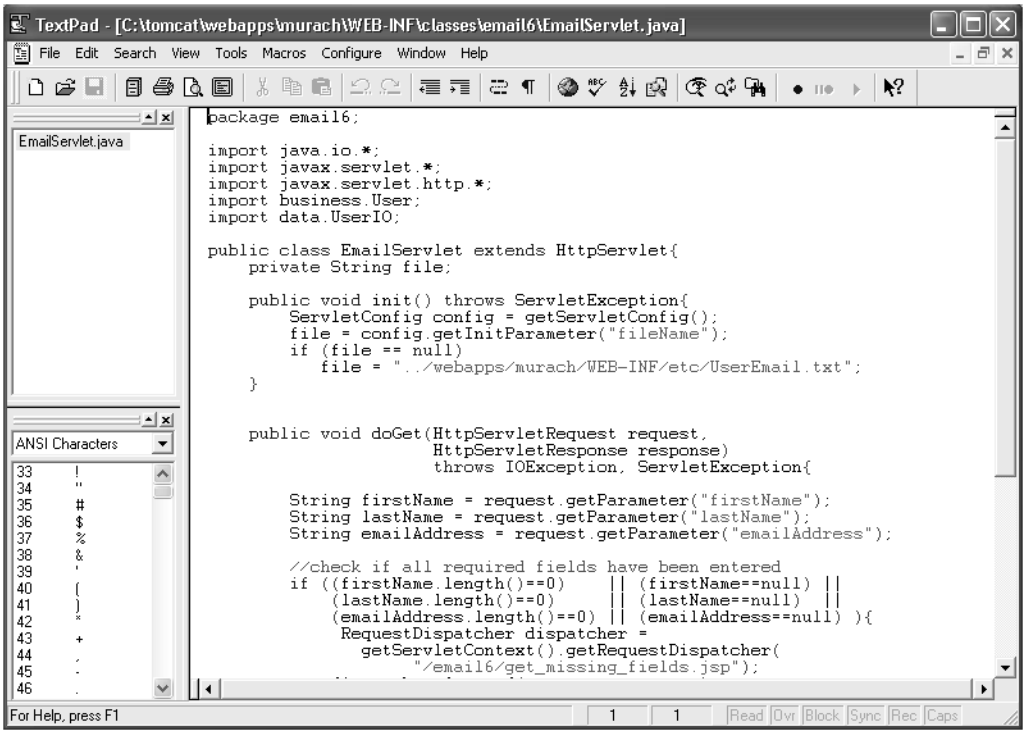
When you write servlets, you normally use a text editor that's designed for entering, editing, compiling, and testing Java classes because that can save you a considerable amount of time. In figure 1-10, for example, you can see an editor named TextPad that is being used to edit a servlet. In case you don't already have an editor like this on your PC, a trial version of TextPad is included on the CD that comes with this book and appendix A shows how to install it. Then, if you decide that you want to use TextPad beyond the trial period, you can pay the fee of about $27, which we think is a bargain.

When you write JSPs, you can use a text editor like TextPad, but you can also use an editor that is designed specifically for working with HTML and JSPs. One of these editors is Macromedia's HomeSite, which is shown in this figure as it's used to edit a JSP. Here again, a trial version is included on the CD in case you want to experiment with it, and appendix A shows how to install it. Then, if you decide to use this software beyond the trial period, you can pay the required fee of about $85 to register your copy.

Although this figure doesn't show one, you can also use an *Integrated Development Environment*, or *IDE*, for developing servlets and JSPs. For instance, Sun ONE Studio 4.0 and Macromedia's JRUN Studio 4 are two IDEs that are designed for developing servlets and JSPs. As a result, they not only provide custom text editors, but also visual tools for designing and debugging servlets and JSPs. Similarly, Macromedia's Dreamweaver is an IDE that has special features for developing HTML documents and JSPs, but not for developing servlets.

Because IDEs can help you work more productively, professional programmers often use them. However, IDEs also have their own learning curves. That's why it's probably best to use a text editor like TextPad while you learn the skills presented in this book. Then, when you're done, you'll be ready to take full advantage of an IDE.

TextPad with the code for a servlet



HomeSite with the code for a JSP

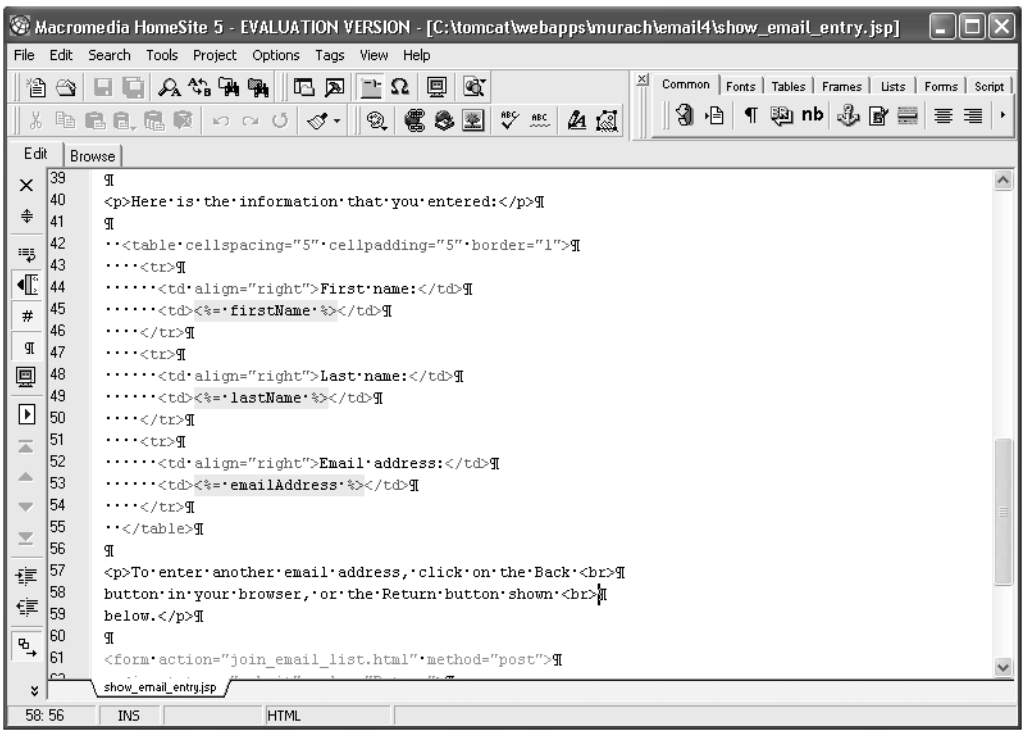


Figure 1-10 Tools for writing servlets and JSPs

Tools for deploying servlets and JSPs

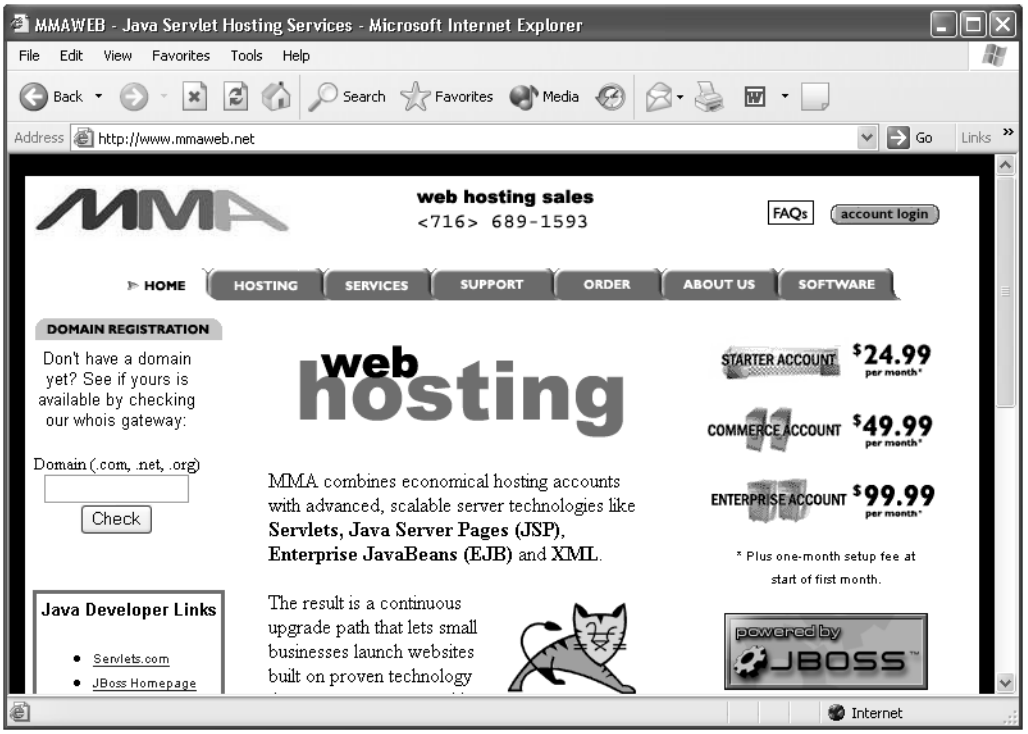
Once you've tested your servlets and JSPs on your own PC or an intranet, you may want to deploy your web application on the Internet. To do that, you need to get a *web host*. One way to do that is to find an *Internet service provider*, or *ISP*, that provides web hosting that supports servlets and JSPs. If you read the text for the ISP on the web page shown in figure 1-11, for example, you can see that this ISP supports servlets and JSPs starting at just $24.99 per month.

If you search the web, you'll be able to find many other ISPs and web hosts. Just make sure that the one you choose not only supports servlet and JSP development, but also the database management system that your application requires.

When you select a web host, you get an *IP address* like 64.71.179.86 that uniquely identifies your web site (IP stands for Internet Protocol). Then, you can get a *domain name* like *www.murach.com* from a company named VeriSign. To go to the web page for getting a domain name, you can enter *www.netsol.com* or *www.verisign.com*. Until you get your domain name, you can use the IP address to access your site.

After you get a web host, you need to transfer your files to the web server. To do that, you can use *File Transfer Protocol*, or *FTP*. One of the most popular programs for doing that is CuteFTP, which is illustrated in this figure. It lets you upload files from your computer to your web server and download files from your web server to your computer.

An ISP that provides web hosting that supports servlets and JSPs



The CuteFTP program

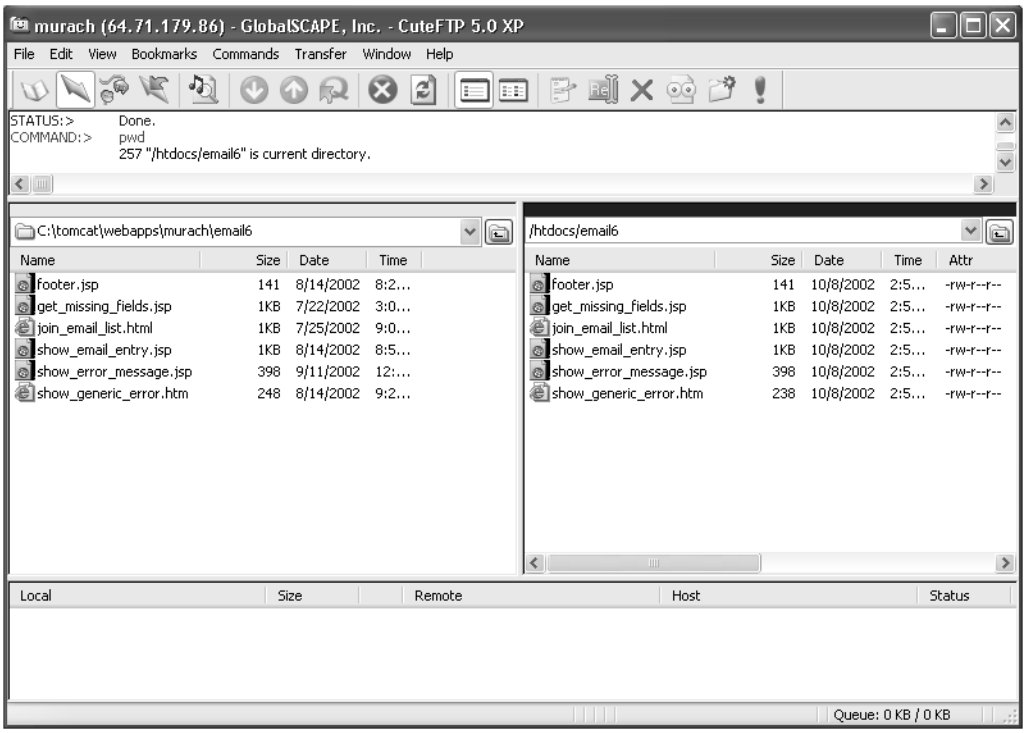


Figure 1-11 Tools for deploying servlets and JSPs

Perspective

The goal of this chapter has been to provide the background that you need for developing servlets and JSPs. Now, if this chapter has succeeded, you should be ready to install Tomcat on your own PC as shown in the next chapter. Once you've done that, you should be ready for rapid progress as chapters 3 through 5 show you how to develop HTML documents, JSPs, and servlets.

Summary

- A *web application* is a set of web pages that are generated in response to user requests.
- To run a web application, the *client* requires a web browser and the *server* requires *web server* software. The server may also require a *database management system*, or *DBMS*.
- *Hypertext Markup Language*, or *HTML*, is the language that the browser converts into the user interface, while *Hypertext Transfer Protocol* (or *HTTP*) is the protocol that web browsers and web servers use to communicate.
- A web browser requests a page from a web server by sending an *HTTP request*. A web server replies by sending an *HTTP response* back to the browser.
- A *static web page* is generated from an HTML document that doesn't change, while a *dynamic web page* is generated by a web application based on the parameters that are included in the HTTP request.
- To run Java web applications, the server requires the Java 2 *Software Development Kit*, or *SDK*, plus a *servlet and JSP engine* like Tomcat.
- A *JavaServer Page*, or *JSP*, consists of HTML with embedded Java code. When it is requested, it is converted into a servlet and compiled by the JSP engine. Then, it is run by the servlet engine.
- A *servlet* is a Java class that runs on a server. For web applications, a servlet extends the HttpServlet class. To pass HTML back to the browser, it uses the println method of the out object.
- When you develop a Java web application, you can use servlets for the processing that's required and JSPs for the page design.
- You can develop servlets and JSPs on your own PC, on a network that functions as an *intranet*, and on the Internet. When you use the Internet, you usually use a web server that's separate from the servlet and JSP engine.
- As you develop a Java web application, you try to divide its classes into three layers: *presentation*, *business rules*, and *data access*. This makes it easier to manage and maintain the application.

Terms

web application
client/server application
client
server
web browser
web server
Apache
database management system (DBMS)
MySQL
local area network (LAN)
intranet
Hypertext Markup Language (HTML)
static web page
Hypertext Transfer Protocol (HTTP)
HTTP request
HTTP response
dynamic web page
applet
servlet and JSP engine
servlet and JSP container
Java 2 Platform, Enterprise Edition (J2EE)
Tomcat
Software Development Kit (SDK)
Java 2 Platform, Standard Edition (J2SE)
Java Runtime Environment (JRE)
Enterprise JavaBean (EJB)
EJB server
EJB container
JavaServer Page (JSP)
servlet
presentation layer
user interface layer
business rules layer
data access layer
JavaBean
Extensible Markup Language (XML)
Integrated Development Environment (IDE)
web host
Internet Service Provider (ISP)
IP address
domain name
File Transfer Protocol (FTP)

Objectives

- Name the software component that is required on the client of any Internet application, and name the two software components that are usually required on the server of any Internet application.
- Distinguish between HTML and HTTP.
- Distinguish between static web pages and dynamic web pages.
- Describe the extra software components that are required for developing servlet and JSP applications.
- In general terms, distinguish between the code for a servlet and a JSP. Then, explain why you use both servlets and JSPs in a Java web application.
- Describe the three types of platforms that can be used for developing web applications.
- List the software components that you need for running servlets and JSPs on your own PC.
- Distinguish between an intranet application and an Internet application.
- List the three layers of a typical Java web application.

Self-study questions

If you're using this book for a course, the tests will be based on the objectives. In other words, the tests try to determine whether you can do what the objectives call for. For this chapter, then, you should be able to answer questions like the ones that follow.

1. What software is required on the clients of an Internet application?
2. What two software components are usually required on the server of any Internet application?
3. What's the difference between HTML and HTTP?
4. What's the difference between static and dynamic web pages?
5. What other software components are required on the server for a Java web application?
6. What is the major coding difference between JSPs and servlets?
7. Why are both servlets and JSPs used in a Java web application?
8. List the three types of platforms that can be used for developing Java web applications?
9. What software do you need for developing servlets and JSPs on your own PC?
10. What's the difference between an intranet and an Internet application?
11. Name the three layers of a Java web application.