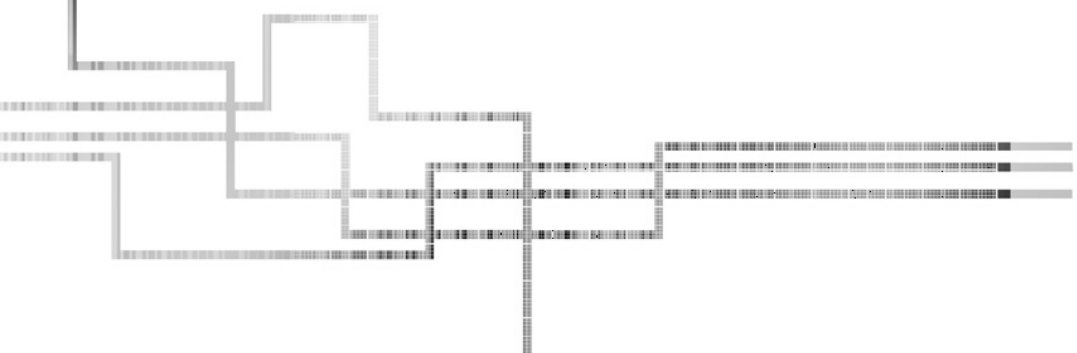
CHAPTER 1

WebLogic Server Basics



In this chapter, we will introduce you to the basics of WebLogic Server, with terms and concepts that will be referenced throughout this book. This chapter offers an overview of the following:

- Various ways to use WebLogic Server as an HTTP server
- The role of WebLogic Server as an application server
- Major Java 2 Enterprise Edition (J2EE) components such as servlets, JSPs (Java Server Pages), and EJB (Enterprise JavaBeans)
- The Resource Adapter
- Details about and differences between development and production environments
- The environment provided by WebLogic Server for application developers and administrators

WebLogic Application Server comes with an HTTP server. As an administrator, you need to be comfortable with administering the HTTP server, and you must gain the necessary skills to administer server-side Java applications with this server. Organizations have hybrid collections of Web servers installed on their sites to support enterprise applications. Therefore, you should be able to integrate WebLogic Server with other HTTP servers.

UNDERSTANDING TCP/IP AND HTTP

HTTP is a protocol that regulates the Way web browsers and servers communicate. The TCP/IP (Transmission Control Protocol/Internet Protocol) suite of protocols is the primary open standard for network communication. HTTP (HyperText Transfer Protocol), part of this suite, is the protocol of the World Wide Web. All the information that resides on the Web as documents or pages is transferred from server to client with the help of HTTP. It is a *stateless* protocol, meaning that it doesn't maintain active session state information about each client connected to the server. Rather, it is based on request and response architecture: the client contacts the server only when it needs information, and the server communicates with the client only when it needs to deliver information back to the client. Other protocols in the TCP/IP suite include UDP (User Datagram Protocol), ICMP (Internet Control Message Protocol), FTP (File Transfer Protocol), ARP (Address Resolution Protocol), RARP (Reverse Address Resolution Protocol), SMTP (Simple Mail Transfer Protocol), and NNTP (Network News Transfer Protocol).

WEBLOGIC WEB SERVER

Web server software runs HTTP services and is able to host one or more Web sites. Each site is a collection of various documents and applications that form the Web content. This content needs to be delivered to Web clients over the network using HTTP.

Various HTTP servers are available on the market today, and most are similar in terms of what they provide because HTTP service is a standard feature of the Web. The WebLogic Web server is a Java-powered server capable of delivering not only static content, but also dynamic content with the help of Java-enabled technologies such as JSP and Java servlets. WebLogic Web server hosts and delivers static HTML/HTM files, images, Java applets, XML (Extensible Markup Language) documents, JSP, Java servlets, multimedia files, and other types of files.

The way Web servers and browsers communicate is represented in Figure 1-1.

The client running the browser software sends a user request to the server using HTTP. The server software (in our case, the WebLogic Server) examines and interprets the request, prepares to locate the appropriate information, locates the information, and then sends an HTTP response to the client. This response is either an HTML document or an image file, which is then interpreted by the client's browser software and presented on the client's interface accordingly.

NOTE Web server software such as Apache, Microsoft Internet Information Services (IIS), IBM HTTP server, WebLogic Server, JRun, and many others are available to provide HTTP services. Presently, because HTTP services have become a subset of application servers, each vendor dealing with application servers has built-in support for Web content handling and management.

Let's break it down even further. The Web server serves requested contents to the client—contents that are either statically available as HTML or dynamically generated using JSP and servlets. When a client sends a request for information to the Web server, the server maps the URL (Uniform Resource Locator) to a file with the given name on the local file system. Then either it reads the content from the disk and serves it out to the network with the aid of HTTP, or the server-side program generates it dynamically.

For example, in the case of a URL that reads *http://www.softwareklinic.com/index.html*, the Web server software will serve *index.html* after locating and loading the file from disk to client. The information contained in the document is placed between HTML tags that, along with the requested information, are carried over the network using HTTP. The client software interprets these tags and presents the information in a fashion appropriate for the user.

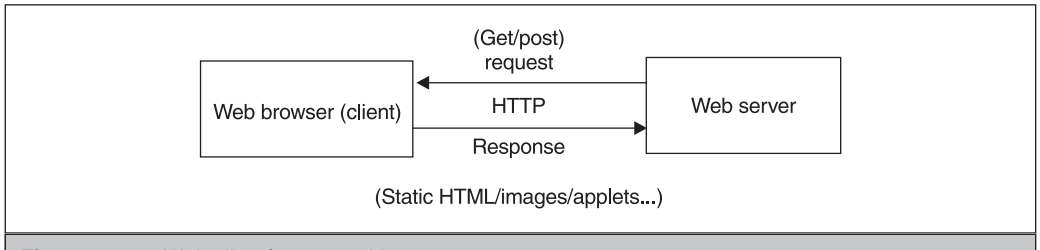


Figure 1-1. Web client/server architecture

Static Web documents are always placed in the respective folder in the appropriate directory. For example, all the static Web documents for the default Web app are placed in the default Web application folder. (Figure 1-2 shows the directory structure of WebLogic 6.1 and 7.0.)

The default Web application directory contains documents delivered when the browser doesn't specify a URI (Uniform Resource Identifier). In other words, if the server is listening on *http://server:7001*, that URL will respond with documents from the default Web application directory. If the browser requests *http://server:7001/otherdocs*, another Web application directory serves the *otherdocs* URI.

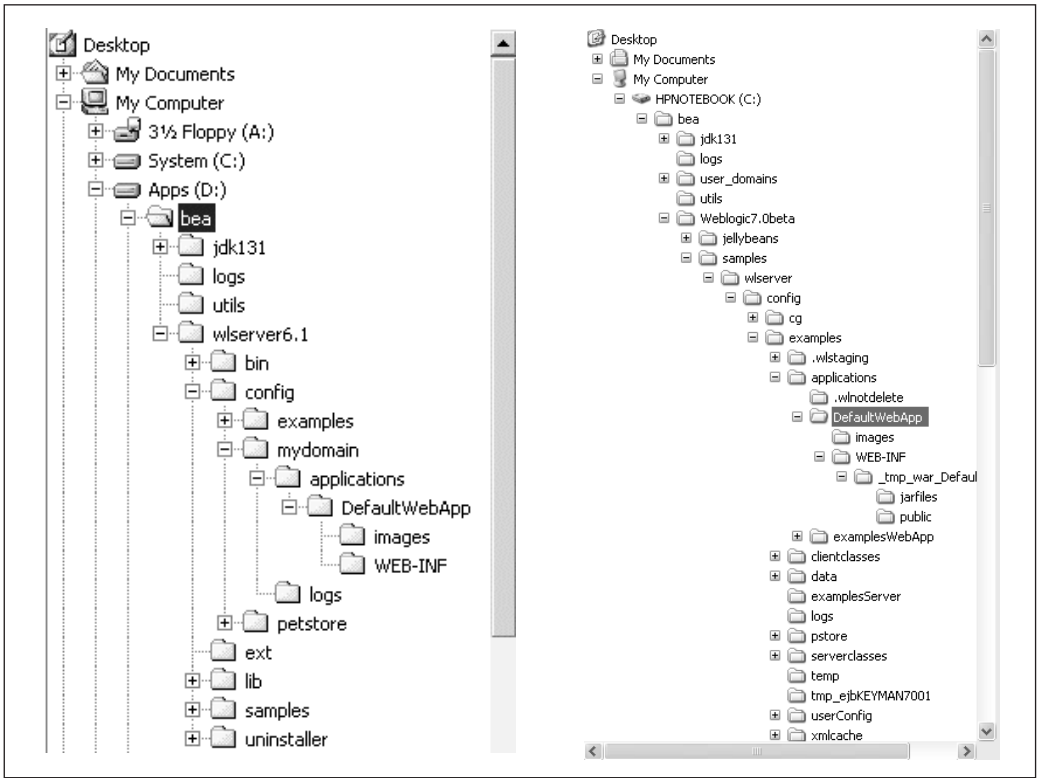


Figure 1-2. Directory structure for WebLogic 6.1 (left) and WebLogic 7.0 (right)

Ports

WebLogic Server typically listens on port 80, but it can be set to listen on port 7001 instead. You can assign the listener port address to any value ranging between 1024 and 65536 (0–1023 are reserved). Figure 1-3 demonstrates the various communication ports available.

If you already have a running server installed at port 80, WebLogic Server can be installed to listen on a port other than 80. It is not possible for WebLogic Server to listen to your client's requests on the same port as another server. For example, if you are using Windows NT/2000, Microsoft IIS will already be installed and running on port 80. In this case, you will either have to stop (shut down) IIS to free up port 80 for WebLogic Server or you must configure the server to listen on another port.

NOTE No two Web servers can be configured to listen at the same HTTP port at the same time, but every Web service can be listened to on a particular port over the network.

To access the home page of your WebLogic server, enter the following link into the browser's address bar: **http://localhost:7001**. Here, *http://* is the protocol, *localhost* is the computer on which the Web site is hosted, and *7001* is the port at which the WebLogic Web server will be ready to listen for a client's request.

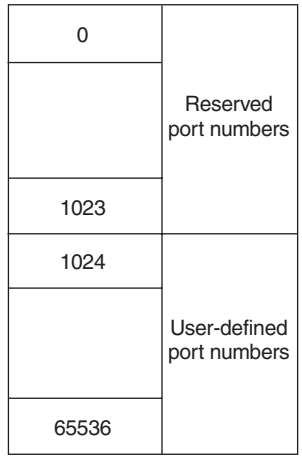


Figure 1-3. Communication ports

Other HTTP Servers

Table 1-1 provides information about other HTTP servers.

For Administrators

If you use WebLogic Server as a Web server, you should be interested in the following aspects of its functionality:

- **Security** Security is the key to keeping critical data related to systems and customers safe from hackers and pirates. It is the responsibility of administrators to ensure that systemwide security policies and profiles are constructed and implemented to discourage hackers/pirates from breaking into the application/site.
- **Virtual hosting** Virtual hosting enables WebLogic Server to host multiple Web sites on a single Web server or a cluster of Web servers.
- **Support for proxy server configuration** WebLogic Server can be integrated with other Web servers such as Microsoft IIS, Apache, and Netscape Enterprise Server. Client requests can be redirected or proxied from a WebLogic server to another Web server.
- **Load balancing** A cluster of servers can be set up to share the load and provide performance enhancements.

HTTP Server	Vendor	Description
Apache	Apache Software Foundation	The most popular Web server on the Internet since April 1996. The January 2002 Netcraft Web Server Survey found that 56 percent of Web sites use Apache.
IIS 5.0	Microsoft	A Web server from Microsoft runs on top of operating systems such as Windows NT/ 2000/ XP Professional. It supports HTTP 1.1 and SSL 3.0. Used for hosting ASP-driven Web sites.
Netscape Enterprise Server/iPlanet FastTrack Server	Sun/Netscape	A Web server from the Sun/Netscape alliance that runs on Windows NT/2000 and various UNIX flavors. Supports HTTP 1.1 and SSL 3.0.
IBM HTTP Server	IBM	An Apache-powered IBM HTTP server that runs on AIX, Linux, zSeries, iSeries, Sun Solaris, HP-UX, and Windows NT.
Oracle HTTP Server	Oracle	A simple Web HTTPD server (Web listener) based on the Apache HTTP Server (*www.apache.org*). Oracle Database Server (8.1.7 and above) and Oracle 9iAS (Oracle Internet Application Server) ship with the Oracle HTTP Server.

Table 1-1. Other HTTP Servers

- **Failover support** With the help of a cluster of servers, it is possible to redirect requests that are part of same session to another WebLogic server in the cluster.
- **Session management in Web farms** In a cluster environment, client states must be maintained elsewhere in case one of the servers in the cluster is malfunctioning. That way, the application remains intact and doesn't have to be restarted.

NOTE WebLogic provides plug-ins for Apache, Microsoft IIS, and Netscape Enterprise Server.

A WebLogic plug-in is a small piece of software that extends the boundaries and capacities of WebLogic Server implementation. It allows WebLogic Server to communicate with other Web servers, as well as access Web applications that have been deployed on those servers.

WEBLOGIC APPLICATION SERVER

Current economic demands require that Web and e-commerce applications help accelerate awareness of companies in growing markets and help them discover new means to reach customers and retain them, as well as ways to introduce new products and provide services to their customers quickly and effectively.

To achieve all these goals, solutions need to be built, developed, and deployed that target effective service to customers. This is possible with the help of proven and reliable e-commerce platforms that allow companies to integrate corporate data, legacy applications on mainframes, and other enterprise applications. That's where WebLogic Server comes into play.

WebLogic Server is an industry-leading e-commerce platform. With WebLogic, it is possible to develop and deploy applications that are reliable, scaleable, secure, manageable, and maintainable. WebLogic facilitates the complexities of system-level details, allowing the user to focus on building a business rather than running a server.

WebLogic Server is also the leader in implementing features of J2EE 1.3, a standard for developing multi-tier enterprise applications. J2EE provides a complete set of services, such as Java servlets; JSP; EJB; HTTP; Java Message Service (JMS); Java Transaction Service (JTA); Java Naming and Directory Interface (JNDI); Java Connection Architecture (JCA); Internet Inter-ORB Protocol (IIOP); Java Authentication and Authorization Service (JAAS); Java Database Connectivity (JDBC); Simple Object Access Protocol (SOAP); Extensible Markup Language (XML); Universal Description, Discovery, and Integration (UDDI); and Web Services Description Language (WSDL). Table 1-2 lists various services provided by WebLogic Server.

WebLogic Server is referred to as *middleware* because it is responsible for connecting the client with the database servers and for serving the information contained in the databases. WebLogic Server is needed in an enterprise for several reasons. For one, when companies want to decrease the size and complexity of client programs, they need to cache and control data flow for better performance and to enhance the

Services	Description
EJB	EJB specification, version 2.0, Second Public Draft; EJB provides a mechanism that contains business logic for building reusable Java components. It also helps users build component-based distributed applications.
HTTP	HTTP specification, version 1.1; WebLogic complies with the HTTP V1.1 specifications.
JAAS	A package that enables services to authenticate the users and enforce access control upon them. It is integrated into Java SDK version 1.4.
JCA	JCA specification, version 1.0; when implemented in WebLogic and Resource Adapters, JCA facilitates connectivity with Enterprise Information Systems (EIS)
JDBC	JDBC specification, version 2.0; JDBC is a Java standard for allowing Java applications to communicate with databases.
JMS	JMS, version 1.02; aids communication between applications with the help of message exchanges.
JNDI	JNDI, version 1.2.1; naming services as a means for locating objects over the network.
JSP	JSP specification, version 1.2; JSPs are used for generating dynamic Web content.
JTA	JTA, version 1.0.1; in a Distributed Transaction System (DTS), JTA is a standard Java interface between the transaction manager and the parties involved.
Servlet	Servlet specification, version 2.3; servlets are server-side Java programs that act as clients to EJB components and have the ability to generate dynamic Web content, process client requests, and communicate with databases.
SOAP	SOAP, version 1.1; a protocol providing an XML/HTTP-based solution for accessing services, objects, and servers in a platform-independent manner.
UDDI	UDDI, version 1.0; UDDI is an industry-standard initiative that enables businesses to locate and communicate with each other. UDDI allows businesses to describe their services, locate businesses that offer desired services, and integrate these services with other businesses. It opens a world of opportunities for enterprises in exchanging services.
WDSL	WSDL, version 1.0; an XML format for specifying Web services as a set of endpoints operating on messages. It's a specification for describing networked XML-based services.
XML	JAXP, version 1.0, SAX version 2.0, DOM Level 2, and W3C Schema; XML is a standard markup language for describing data in structured fashion.

Table 1-2. WebLogic Server Services

performance of the entire system, while providing security for both data and users of the system.

In client/server applications, the business logic is split across the client and server, but it usually resides in the client application. This increases the complexity of software. In addition, upgrading software or applying any changes is a huge job in itself, as these changes have to be managed with all client systems on the network. This creates the need for software that helps connect the two pieces—client application and databases—

while managing all business logic and providing seamless connectivity to the front and back ends.

The architecture of Web-based applications is both two and three tiered (see Figures 1-4 and 1-5). In a scenario that involves simply a Web client and a Web server, the architecture is two tiered: client and server. However, if the services are extended to provide the client information from various sources, such as a database, a third tier is added.

Web servers do have limitations. They cannot provide more elaborate service to the client other than static pages with static information. To resolve this, a typical piece of software and a development language is needed that helps build logical pages and that contains not only data for presentation but a way to gather information dynamically from the back-end systems and build pages on the fly to deliver to the client.

The role of the application server differs from application to application. Not every application requires the same functionality and set of services from an application server. Take scalability, for example. Smaller companies might want an application server that helps them organize their applications for the Web, that provides better control over the way business logic is contained and managed, and that makes it easier to monitor and secure the data. They don't need multiple servers. On the other hand, large corporations or enterprises may need to manage multiple servers. For them, the scalability of an application server is crucial because they are expecting a huge number of users to visit their Web sites and do business online. WebLogic Server provides everything that's needed for such business needs; it's up to the user to make appropriate use of what is provided.

Before deciding on an application server, an organization must conduct an in-depth and accurate study of its requirements. Look into factors such as security, scalability, business logic management, and database connectivity to decide which server is appropriate.

Keep in mind that not all products from the same family of application servers are written using the same programming framework. While many—though not all—

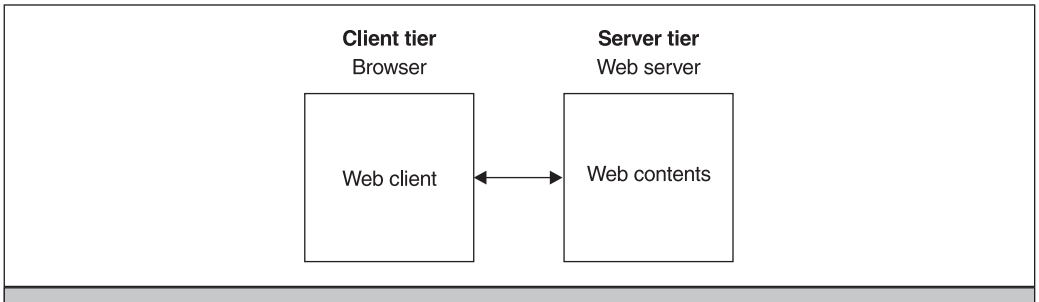


Figure 1-4. Two-tier client/server architecture of Web-based applications

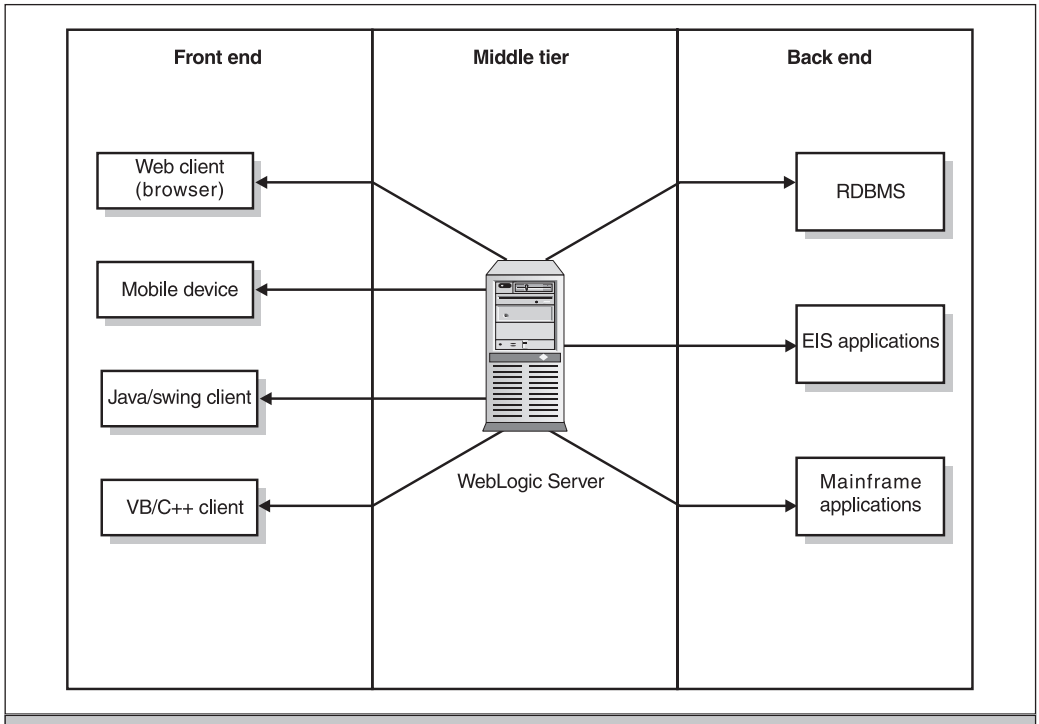


Figure 1-5. Three-tier client/server architecture of Web-based applications

products are written in Java, some are Microsoft friendly and others are not. However, there is room for all, including support for Java, CORBA, or Microsoft COM+ and the .NET Framework of distributed application development infrastructures.

If you are working for an organization that's looking to run enterprise Java applications in n-tier architecture, you're going to need to work with an application server, and a place for it in the infrastructure is an inevitable necessity. The application server is the cornerstone of a software architecture designed to tie together different components of a complex application, yet maintain a fundamental modularity in the software. First and foremost, application servers provide the glue that connects information from a database with the end user or client program, which is often running on a Web browser.

WebLogic Server provides the means to cache and control data flow for better overall performance and scalability of applications in production. It has the potential to provide security for both data and user traffic. WebLogic Server extracts data from the database to individual applications instead of requiring that each of those applications make a

call to the database directly, thereby reducing direct database hits, which adds to the overall performance of the entire system.

The Web is automatically three tiered, with a client-centered application, a Web server, and one or more databases. Therefore, managing data along with application functionality is not only an exercise in better application design, but also a downright necessity.

WEBLOGIC 7 FEATURES

With each new version of WebLogic Server, new features help ease development, deployment, and administration tasks. In addition, facilities and enhancements help developers make applications more secure, robust, and reliable. Versions of WebLogic Server since 6.*x* have initiated support for Web services, enhanced and improved the security infrastructure, provided new tools useful to application developers such as EJBGen and Deployer, brought about major enhancements to ease administration of various J2EE components, and made application deployment a more comfortable process.

WebLogic Server 7 also includes enhancements concerning administration of caching and clustering. With the help of WebLogic Server 7 *cache tags*, administrators can configure caching for entire pages, URLs, and file types. The most exciting aspect of the cache tags enhancement is its ease of use—there is no need for administrators to make any changes to the application, and the system can realize an immediate gain in performance.

With WebLogic Server 6.*x*, a multi-home environment was necessary, in which each server within the cluster had its own IP address. So, for example, even if you wanted to set up a cluster environment on a single computer, you needed to set up the multi-home environment, in which multiple IPs exists on a single computer.

WEBLOGIC SYSTEM ADMINISTRATOR INFRASTRUCTURE

In the early days of application servers, computer networks within an organization grew as a result of incorporating additional components and systems. Typically, such components as active/intelligent hubs, routers, switches, gateways, PCs, network printers, and storage area networks (SANs) were added to the network, making it complex and expensive. This was tackled by incorporating standards such as management information bases (MIBs) and Simple Network Management Protocol (SNMP) within the products, as has been done with Tivoli. Challenges faced in managing a software deployment task are just as complicated as any network challenges in the enterprise. These issues exist not only in the Java world, but also in the entire Web community.

What do network management solutions have to do with Java applications? With Java Management Extensions (JMX), Sun has come up with a standard that allows Java

developers to integrate their applications with existing network management solutions and infrastructure without using proprietary software. JMX is an API that models system administration functions with the help of Java objects known as *MBeans* (Management Beans). WebLogic Server implements 100-percent Java standards. System administration in WebLogic Server is implemented from the ground up using the JMX specification.

J2EE COMPONENTS

The most basic and necessary components of J2EE applications are

- Servlets
- JSPs
- EJBs
- Resource Adapter

Servlets

A Java *servlet* is a Java program (a class file) that runs in the environment provided by a Java-enabled server, such as WebLogic Server. The most common use of Java servlets on WebLogic Server is to create interactive applications using standard Web browsers for the client-side presentation. WebLogic Server lets users develop and deploy business logic as a server-side process within servlets. Servlet capabilities extend from user interface (UI) generation to access databases, EJBs, messaging APIs, HTTP sessions, and other facilities exposed by WebLogic Server.

The Java servlet is simply a Java class (a file containing bytecodes) that is loaded in the Java Virtual Machine (JVM) environment on the WebLogic Server. After the servlet is loaded in the memory, one of its class methods is called to service the client request. A servlet may remain in memory for subsequent client requests and may serve multiple clients; the method is merely accessed once more, without the overhead of reloading and initializing the servlet again. This significantly improves the efficiency of the server. Figure 1-6 demonstrates the structure of a typical Java servlet, where you typically hard-code HTML tags within Java program code.

NOTE A Java servlet is a program that runs when the client requests specific information from the server. Consider a Java servlet as a server-side applet designed to work on the server and then return the information to the browser through the Web server. In much the same way that an applet extends the functionality of a browser, the Java servlet extends a server. The applet is an amazing client program, and the servlet is an amazing server program.

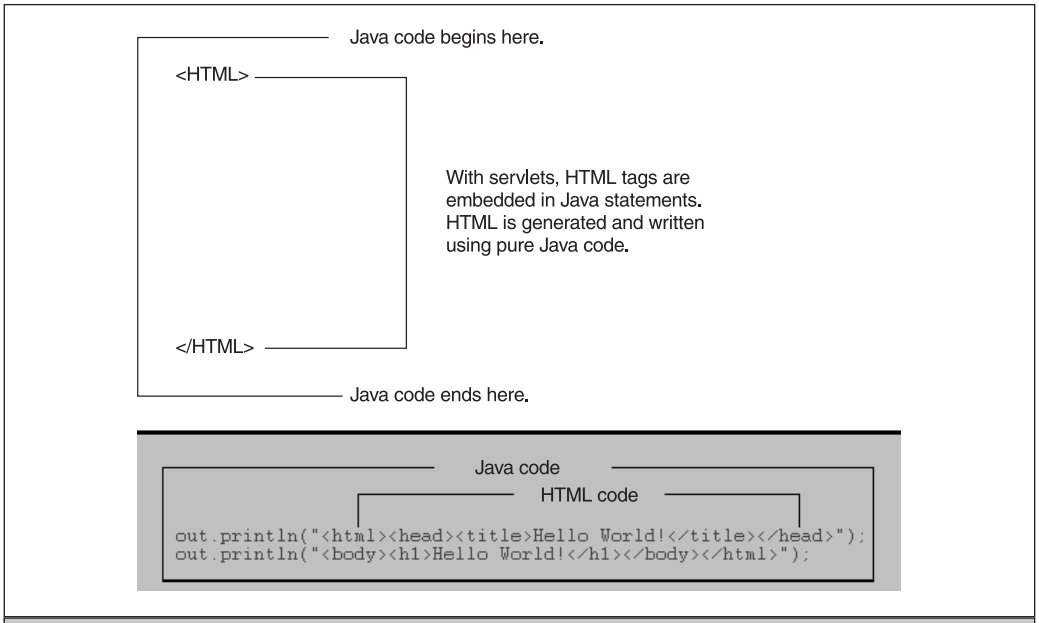


Figure 1-6. Servlets coding architecture

WebLogic Server fully supports HTTP servlets as defined in the Servlet 2.3 Specification from Sun Microsystems. HTTP servlets form an integral part of J2EE.

JSP

JSP is Sun's specification for embedding Java with HTML to provide on-the-fly content for Web pages. When you create dynamic content, JSPs are more convenient to write than HTTP servlets because they allow you to embed Java code directly into your HTML pages, in contrast with HTTP servlets, in which you embed HTML inside Java code. JSP is part of the J2EE.

JSP enables you to separate the dynamic content of a Web page from its static counterpart. It renders services to two different types of developers: HTML developers, who are responsible for the graphic design of the page, and Java developers, who handle the development of software to create the dynamic content.

Because JSP is part of the J2EE standard, JSPs can be deployed on a variety of platforms, including WebLogic Server. In addition, third-party vendors and application developers can provide JavaBean components and define custom JSP tags, which can be referenced from a JSP page to provide dynamic content.

NOTE Java servlets and JSP reside in a special environment on the application server known as a *Web container*.

In which areas of applications can Java servlets and JSPs be used? Figure 1-7 delivers the concept of coding architecture of JSP, in which the Java scriptlets are embedded within HTML tags. Figure 1-8 provides a JSP example as a proof to the concept demonstrated in Figure 1-8. A Java servlet is a pure Java program written using Java Programming Language (HTML is embedded in the Java code), whereas a scriptlet is a Java code snippet that is embedded within HTML code. Servlets and JSP have inside-out architecture (exactly opposite to each other).

Areas of Application

The following shows where servlets can be applied:

- **HTML form processing** This is the processing of Web-based forms, including those used to store the data to a file or package it and e-mail it to an administrator. Common Gateway Interface (CGI) scripts have conventionally performed these functions. One of the most common CGI scripts in use for this task is the `form2email.cgi` program. However, servlets provide better performance and scalability than CGI.
- **HTML page counters** Popular additions to many Web pages, page counters can track the number of times a user has accessed a given page by incrementing a counter file somewhere on the server. Although popular, they tend to place significant overhead on the server when implemented using CGI scripts.

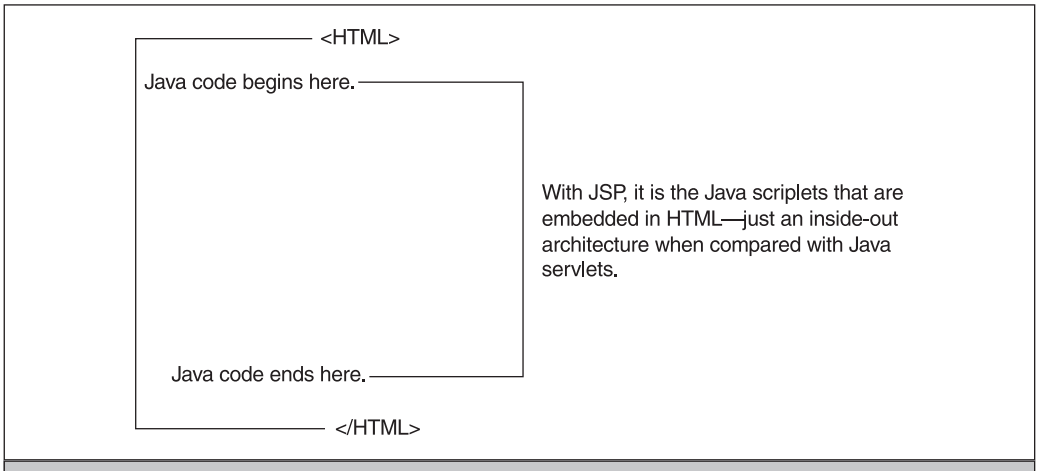
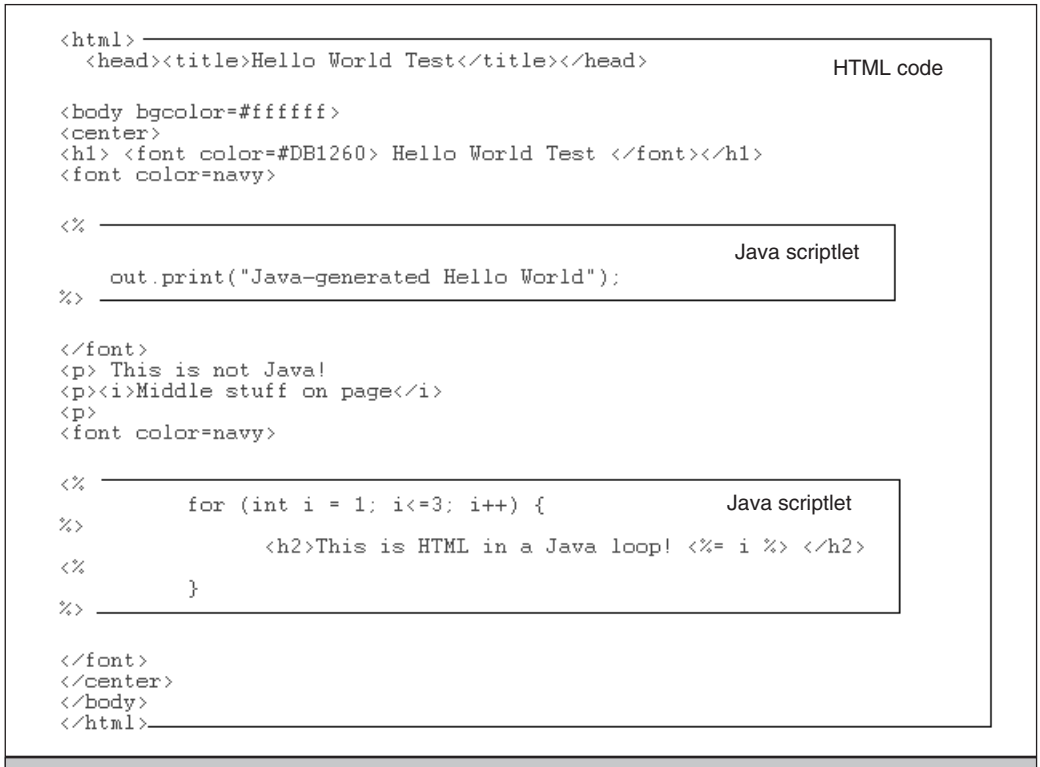


Figure 1-7. JSP coding architecture




```
<html>
  <head><title>Hello World Test</title></head>

<body bgcolor=#ffffff>
<center>
<h1> <font color=#DB1260> Hello World Test </font></h1>
<font color=navy>

<%
    out.print("Java-generated Hello World");
%>

</font>
<p> This is not Java!
<p><i>Middle stuff on page</i>
<p>
<font color=navy>

<%
        for (int i = 1; i<=3; i++) {
%>
              <h2>This is HTML in a Java loop! <%= i %> </h2>
<%
        }
%>

</font>
</center>
</body>
</html>
```

Figure 1-8. JSP coding example

- **Newsgroups** This is a Web-based version of the UseNet groups found on the Internet. A newsgroup is a mechanism that allows users to exchange data by leaving messages for everyone to read. These messages can also be replied to. Threads of conversations take place, in which people reply to replies.
- **Guest books** A guest book is a place for guests to a Web site to leave their comments or suggestions for the webmaster. This differs from merely sending the webmaster an e-mail, as other visitors can also see the posted comments.
- **Search engines** A site search engine allows the user to find information quickly and easily from within the site, without having to root around for it. Although CGI-based search engines do exist, it has been left to the built-in capabilities of the Web server to provide such functionality.
- **Banner advertising** Online advertising is becoming increasingly popular with the more highly accessed sites that sell banner space to advertisers. A banner allows the advertiser to display a small image that, when clicked, will take the user to an alternative site.

- **Quote generators** A quote generator is a small program that, when run, generates a new line of text. For example, UNIX fortune cookies run every time a user logs into the system, presenting the user with a new pearl of wisdom. Web-based generators operate in somewhat the same way, by inserting a new line of text into the Web page every time it is accessed.
- **Random links** It is both common and courteous to have a place on a Web site that offers a list of additional places that the user may wish to visit. These lists can sometimes get very long. Instead of having the user wade through lists, webmasters can create one link that users can click to take them to a new link that's randomly chosen from a list of possible links.
- **Chat programs** Chat programs allow users to talk to each other in real time on the Internet. Internet Relay Chat (IRC) is one of the most common protocols for conversing on the Net. A separate program that runs outside the Web environment is required to use IRC (e.g., MSN Messenger or Yahoo Messenger). However, CGI was one of the tools used to bring an HTML version of IRC to the user.

Future Applications

When server-side processes are implemented using Java, a whole new world of applications can be realized. Applications that had previously required a sophisticated language solution can be easily implemented for a multiple-platform environment in Java servlets. These are listed here:

- **Advanced database accesses** Providing access to databases via CGI scripts was never much of a problem; however, controlling the number of sessions and security issues sometimes was.
- **Virtual shopping baskets** Virtual shopping baskets allow users to browse a site, adding items to be purchased to a list (or "cart") as they go. Once the shopper is finished, he or she can visit the virtual checkout for payment and delivery details.
- **Online quizzes** In Web-based quizzes, users answer multiple-choice questions as they race against a clock. The server must keep track of the answers as it also keeps an accurate record of the time. All this happens without users having to log into the game beforehand.
- **Dynamic images** Dynamic images are generated by drawing on a virtual canvas in the server's memory. Everything on the canvas is then converted to an image. The image can then be transmitted to the client browser via a .gif or .jpg image file.
- **Advanced HTML filters** Before a Web page is delivered to a client, it can be preprocessed, removing any references to words that may be deemed unsuitable by the Web or site administrator. This process can also be extended to replace terms, as opposed to search-and-destroy–type applications.

- **Advanced HTML form processing** Users have the ability to send files or data in a secure format to the server from a Web-based form.
- **E-mail transmitting servers** More sophisticated e-mail distribution list applications are available. Users can sign up to receive e-mail, such as a new joke every day or a different passage from a book.
- **Site analysis** In addition to providing weekly and daily statistical information regarding the number of visitors to a Web site, up-to-the-second information can be made available to site administrators. They then have the ability to see who is viewing the pages at any one point in time.
- **News feeds** In broadcast systems, the user is informed of an event the second it happens, as opposed to the user having to look for the information. With news feeds, the information finds the user.

EJB

EJBs are network-aware components distributed for developing secure, scaleable, transactional, and multi-user components in a J2EE environment of WebLogic Server. These components are then deployed and rolled out on a J2EE-enabled WebLogic Server. This description describes EJBs from a functional point of view—that is, what they do. A more structural explanation would be that EJBs are collections of Java classes, interfaces, and XML files adhering to given rules.

It's worth noting that in J2EE, all the components run inside their own containers. JSP, servlets, and JavaBeans run under Web containers. Similarly EJBs run inside an EJB container, as shown in Figure 1-9. The EJB container provides a standard set of services, such as transaction management, persistence, security, component pooling, resource management, and concurrency. EJBs are the standard for working with server-side components on Java-enabled servers. WebLogic Server has implemented the EJB architecture based on Sun's EJB specification.

EJB Types

EJBs come in four different types:

- **Stateless session beans** A stateless session bean does not maintain state between method calls. *State* is considered the result of a prior method call, or setting of an attribute, which is made available in a later part of the application on the Web. With stateless session beans, you cannot depend upon the result of a prior method call when making a new method call. Stateless session beans are often used to access a database or service directly where prior knowledge of events is not required or desirable. Other stateless behavior—for example, returning a list of currently logged-in users—might be modeled with a stateless session bean.

- **Stateful session beans** Stateful session beans remember what happened from prior method calls. A stateful session bean may act stateless, but in most cases, it "remembers" what happened before.
- **Entity beans** Entity beans, added into the EJB specification at version 1.1, are essentially stateful session beans with persistent behavior.
- **Message-driven beans** Message-driven beans were added into the EJB specification with version 2.0. These are effectively stateful session beans that operate asynchronously. A message bean sits idle, waiting for a message; when one arrives, the bean processes it. Message beans can remember prior state; however, because users have no control over which particular copy of a message bean received their message (many copies of the same bean could be held in memory), prudence dictates avoiding taking advantage of state. Stateful behavior in a message bean should be constrained to reading and working with startup-type information. Message-driven beans are similar to everyday mail handlers. You drop a message into a mailbox, and a mail carrier picks it up and forwards it to someone to read. The reader may send back a message in return, but then again, she may not.

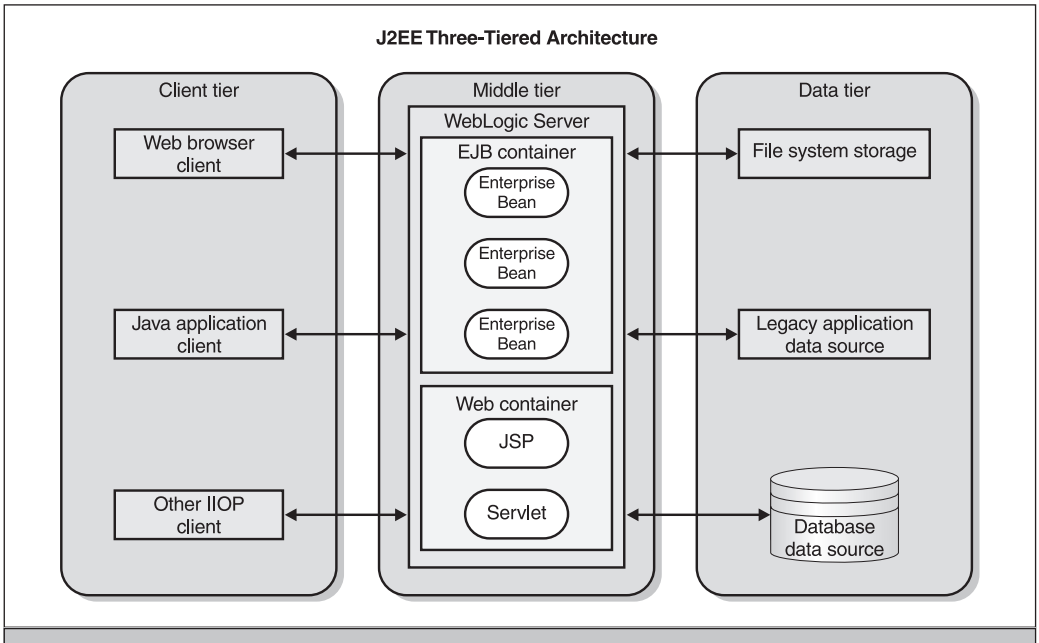


Figure 1-9. Web and EJB containers

EJB Container Services

The WebLogic Server container provides certain built-in services to EJBs, which the EJBs use to function. The services that the EJB container provides are shown here:

- Component pooling
- Resource management
- Transaction management
- Security
- Persistence
- Concurrency

At times, EJB is used to map database entities to Java objects and encapsulate all functionality within an EJB component. These objects are then used by Java servlets that act as consumers of services that EJBs have to expose.

Resource Adapter

To understand the purpose of a Resource Adapter, you first need to know about the *connector architecture*. With the latest version of J2EE is a new architecture for integration of a J2EE-compliant application server, such as WebLogic with EIS.

Figure 1-10 demonstrates that the central component within the WebLogic J2EE connector architecture is the Resource Adapter, which serves as the "connector." Connector architecture enables both EIS vendors and third-party application developers to join hands and develop Resource Adapters, which can be deployed in any application server that provides support for J2EE 1.3 specifications from Sun Microsystems. Resource Adapters contain the Java components, and if necessary, also the native components Required for interacting with the EIS.

When a Resource Adapter is deployed in the WebLogic Server environment, it enables the development of robust J2EE applications that have access to a remote EIS system. Developers of WebLogic Server applications can use HTTP servlets, JSPs, EJBs, and other APIs to develop integrated applications that use the data and business logic of the EIS.

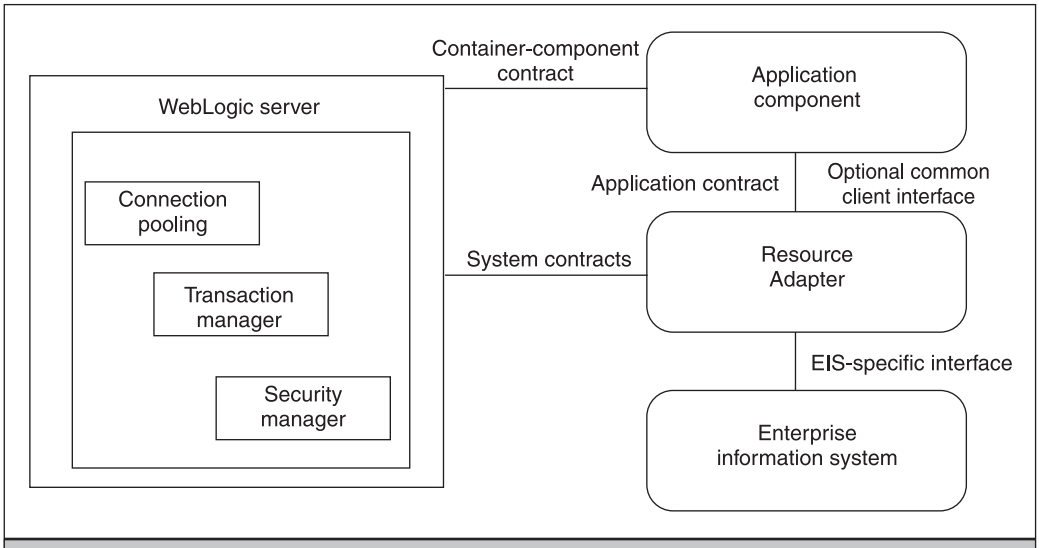


Figure 1-10. Resource Adapters

DEVELOPMENT ENVIRONMENT VS. PRODUCTION ENVIRONMENT

Development of an application and carrying it to production are two distinctly difficult activities. When the system is under development, typically various groups/teams are working on each piece of the system in an independent, rather disconnected manner. The teams do coordinate if some piece is complete and is deemed usable by other teams, but the development process is often not a well-integrated and connected process. Ideally, a well-constructed system emphasizes integration and tight coupling between various components of the system; but the reality is that it is difficult to attain this in a development environment.

In the traditional project development environment, a wide variety of tools exist for different activities during the project's life cycle. Mostly, these tools do not coexist on the same platform, thereby disconnecting various project activities.

In the typical development and production process, some teams design front ends; others build front-end logic; others work at various business layers (middle-tier); some teams are responsible for database administration; and some are responsible for database

development, business analysis, and so on. Such an environment can prove difficult for keeping teams working in a truly integrated manner.

When developing in Java, you should always make sure that you're working in a controlled development environment. To avoid Java class conflicts and other problems that can be difficult to diagnose, you need to be aware of all of the environment settings that are in use during development. A common set of tools must be used by all the teams participating in development for source code control and versioning, test frameworks, source code editors, and coding conventions and guidelines.

Having a deployment environment preconfigured in development to match the production environment helps ensure high-quality deployments and helps to streamline the application life cycle. You should develop your application by utilizing the J2EE model for application deployment. J2EE applications provide a consistent, portable, and easily maintainable way to deploy your applications.

WebLogic Server has a provision for *auto-deployment*, which is viable for quickly deploying the application on the administration server. We recommend that you use this method of deployment only in a single-server development environment. It is not advisable in a production environment or while deploying components on managed servers.

CAUTION You must ensure that auto-deployment is turned off in the production environment.

Another mechanism of deploying the applications is known as *hot-deployment*, or *dynamic deployment*. In this case, whenever the application undergoes change, it is automatically deployed in the sense that changes are immediately impacted. In addition, this setting is, by default, enabled.

CAUTION Dynamic deployment should not be used in a production environment. Instead, a manual or static deployment of applications should occur.

In the development environment, dynamic deployment or auto-deployment can be afforded because it is just internal—that is, it's internal to the teams concerned. However, when in the production environment, great care must be taken to not initiate any steps that can lead to unexpected behavior and user experiences and that can harm organizations' reputations.

Typically, Web applications undergo constant changes, and users don't want to see these changes every time the contents are added, subtracted, or manipulated on the Web site. Even the functionality of the Web applications can change, which at times involves new business rules. This doesn't mean that the only point at which we can and should deploy applications is when the server boots, however. At times, bug fixes

need to be dropped without bringing down the server and without affecting the availability and functionality of the Web site. In such a case, we must perform dynamic deployment. Hence, specific features that we need to apply to specific content are crucial.

One more issue crucial in a production environment is the *discovery of managed servers*. WebLogic Server has an administration server and a managed server. An administration server manages the managed servers. Therefore, you must deploy your applications in the production environment on the managed servers and not use an administration server for that purpose. You can restart the administration server without affecting the clients connected to managed servers, even when the managed servers are running.

If you restart the WebLogic administration server while the managed servers are still running, the administration server will detect the presence of the running managed servers if instructed to do so. To instruct the administration server to look for running managed servers, enter the following argument on the command line when starting the administration server:

```
-Dweblogic.management.discover=true
```

The default value of this attribute is `true`. Why is it, then, that we have to set it to `true`? To ensure that either this parameter is not set or at least not set to `false`. The configuration directory for the domain contains the file *running-managed-servers.xml*, which is a list of the managed servers that the administration server knows about. When the administration server is instructed to perform discovery upon startup, it uses this list to check for the presence of running managed servers.

From a hardware perspective, the way we configure our development servers is a lot different from the way we configure our production servers. Development servers are never under stress and don't need to be scaleable in terms of supporting a large number of users, whereas the production servers do function under stress and must be scaleable.

WEB SERVICES

Web services is a blanket term used for defining the infrastructure required to link applications in a business-to-business (B2B) world. Web services go beyond those things traditionally provided by Web applications and provide a standard mechanism for linking disparate systems in a uniform and well-defined manner. Web services provide a common protocol that Web applications can use to connect to each other over the Internet.

Web services will change the way the industry and companies view their applications. Applications that previously were difficult or impossible to combine can be exposed and connected quickly and easily using Web services.

A Web service is made up of a number of the following parts and services:

- Web Services Description Language, or WSDL, is used to define the external view of a service. Applications use WSDL to understand how to talk to existing Web services and how to expose functionality as a Web service. WSDL works

much like a Remote Procedure Call (RPC) mechanism and is written completely in XML.

- UDDI and Electronic Business XML Initiative (ebXML) provide a mechanism that both registers and searches for a given service. Using WSDL, a Web service makes itself known in the global "marketplace" via the UDDI publish service (by publishing your XML Web service to the UDDI registry). Other Web services can then find an existing service by using the UDDI Inquiry API. UDDI represents simple, typically point-to-point Web services. ebXML provides a mechanism similar to UDDI but with a much broader list of query APIs. It is typically found in more complex applications that require multiple services to interact at one time.
- SOAP provides the final portion of a Web service, using a mechanism to invoke a Web service that we have found using UDDI and understand via WSDL.

Web services are an interesting new area that focuses on exposing enterprise services through the Web. A point to note is that Web services are a number of interconnected protocols, defined using the Java community process (JCP), but technically not J2EE services.

WLS ENVIRONMENT AND TOOLS

Development and production environments differ in the way their applications and availability are managed. The environment that most closely resembles the production environment is that of the User Acceptance Test (UAT). With WebLogic Server, it is possible to change the configuration attributes of domain resources dynamically—that is, while servers are running. One of the strongest features is that, in most cases, the WebLogic Server does not have to be restarted for changes to take effect. When an attribute is reconfigured (when the value is changed), the new value is immediately reflected in both the current runtime value of the attribute and the persistent value stored in the XML configuration file.

Setting the classpath Option

To execute various command-line administration tools, you will be required to set the `CLASSPATH` variable, or the following must be included as values to the `-classpath` option on the `java` command line:

```
/weblogic/lib/weblogic_sp.jar
/weblogic/lib/weblogic.jar
```

WebLogic Server provides a default database management system (DBMS) called Cloudscape. To use this DBMS, the following needs to be included in `classpath` setup:

```
/weblogic/samples/eval/cloudscape/lib/cloudscape.jar
```

If you will be using WebLogic Enterprise Connectivity, you will need to include the following:

```
/weblogic/lib/poolorb.jar
```

where *weblogic* is the directory in which you installed WebLogic Server.

To set the CLASSPATH variable on the command line, specify the following:

```
SET CLASSPATH=C:\bea\weblogic700b\server\lib\weblogic.jar
```

Starting WebLogic Server

To start WebLogic Server from the Start menu, choose Programs | BEA WebLogic Enterprise Platform | WebLogic Platform Beta 7.0 | User Domains | My Domain | My Server. Alternatively, follow these steps:

1. Access the command prompt by choosing Start | Run | CMD.
2. Change the working directory to **C:\BEA\User_Domains\MyDomain**.
3. Run the *setenv.bat* file on Microsoft Windows platform, or *setenv.sh* on the UNIX platform. This file internally calls another file from *C:\BEA\WebLogic700b\Server\Bin* called *startWebLogic.cmd*. (*StartWebLogic.cmd* is the file that declares necessary environment variables. The environment variables specified in *startWebLogic.cmd* are used by WebLogic Server as input while starting up.) It further assigns few more variables, such as WL_HOME and JAVA_HOME, sets the CLASSPATH, appends PATH for WebLogic Server *bin* and Java *bin* folders, and finally executes *weblogic.Server*. It also makes sure that the *weblogic.jar* file is available and that CLASSPATH points to it.

Sample *startWebLogic.cmd* (C:\BEA\User_Domains\MyDomain)

```
@rem ************************************************************
@rem This script is used to start WebLogic Server for the domain in
@rem the current working directory.  All this script does is set the
@rem DOMAIN_NAME and SERVER_NAME variables, then calls the
@rem startWebLogic.cmd script under %WL_HOME%\server\bin.
@rem
@rem To create your own start script for your domain, all you need to
@rem set is DOMAIN_NAME and SERVER_NAME, then call
@rem %WL_HOME%\server\bin\startWebLogic.cmd
@rem
@rem Other variables that startWebLogic takes are:
@rem
@rem WLS_USER     - cleartext user for server startup
@rem WLS_PW       - cleartext password for server startup
```

```
@rem STARTMODE    - true for production mode servers, false for
@rem                development mode
@rem JAVA_OPTIONS - Java command-line options for running the server.
@rem               (These will be tagged on to the end of the JAVA_VM
@rem                and MEM_ARGS)
@rem JAVA_VM      - The java arg specifying the VM to run.
@rem                (i.e. -server,
@rem                -hotspot, etc.)
@rem MEM_ARGS     - The variable to override the standard memory
@rem                arguments
@rem                passed to java
@rem
@rem For additional information, refer to Installing and
@rem                Setting up WebLogic
@rem Server (http://e-docs.bea.com/wls/docs70/install/index.html).
@rem ***************************************************************

echo off
SETLOCAL

@rem Set DOMAIN_NAME to the name of the domain you wish to run.
set DOMAIN_NAME=mydomain

@rem Set SERVER_NAME to the name of the server you wish to start up.
set SERVER_NAME=myserver

@rem Set WLS_USER equal to your system username and WLS_PW equal
@rem to your system password for no username and password prompt
@rem during server startup.  Both are required to bypass the startup
@rem prompt.
set WLS_USER=installadministrator
set WLS_PW=installadministrator

@rem Set Production Mode.  When this is set to true,
@rem the server starts up in
@rem production mode.  When set to false, the server starts
@rem up in development
@rem mode.  If it is not set, it will default to false.
set STARTMODE=

@rem Set JAVA_OPTIONS to the java flags you want to pass to the vm. i.e
@rem set JAVA_OPTIONS=-Dweblogic.attribute=value -Djava.attribute=value
set JAVA_OPTIONS=
```

```
@rem Call WebLogic Server
call "C:\bea\weblogic700b\server\bin\startWebLogic.cmd"

ENDLOCAL
```

Sample *startWebLogic.cmd* (Under *C:\BEA\WebLogic700b\Server\Bin*)

```
@rem ****************************************************************
@rem This script is used to start WebLogic Server
@rem
@rem To create your own start script for your domain,
@rem all you need to set is
@rem DOMAIN_NAME and SERVER_NAME, then call this script from the domain
@rem directory.
@rem
@rem This script sets the following variables before starting
@rem WebLogic Server:
@rem
@rem WL_HOME     - The root directory of your WebLogic installation
@rem JAVA_HOME   - Location of the version of Java used to start
@rem               WebLogic Server. This variable must point to the root
@rem               directory of a
@rem               JDK installation and will be set for you by the
@rem               installer.
@rem               See the WebLogic platform support page
@rem               (http://e-docs.bea.com/wls/platforms/index.html) for
@rem               an up-to-date list of
@rem               supported JVMs on Windows NT.
@rem PATH        - Adds the JDK and WebLogic directories to the
@rem               system path.
@rem CLASSPATH   - Adds the JDK and WebLogic jars to the classpath.
@rem
@rem Other variables that startWebLogic takes are:
@rem
@rem WLS_USER      - admin username for server startup
@rem WLS_PW        - cleartext password for server startup
@rem ADMIN_URL     - if this variable is set, the server started will be
@rem                 managed server, and will look to the url specified
@rem                 (i.e. http://localhost:7001) as the admin server.
@rem STARTMODE     - set to true for production mode servers, false for
@rem                 development mode
@rem JAVA_OPTIONS - Java command-line options for running the server.
@rem                 (These
@rem                 will be tagged on to the end of the JAVA_VM
@rem                 and MEM_ARGS)
```

```
@rem JAVA_VM      - The java arg specifying the VM to run.
@rem                (i.e. -server, -client, etc.)
@rem MEM_ARGS     - The variable to override the standard
@rem                memory arguments passed to java
@rem
@rem jDriver for Oracle users:This script assumes that native libraries
@rem required for jDriver for Oracle have been installed in the proper
@rem location and that your system PATH variable has been set
@rem appropriately.
@rem
@rem For additional information, refer to Installing and Setting up
@rem WebLogic
@rem Server (http://e-docs.bea.com/wls/docs70/install/index.html).
@rem ******************************************************************

@echo off
SETLOCAL

set WL_HOME=C:\bea\weblogic700b
set JAVA_HOME=C:\bea\jdk131

@rem Check that the WebLogic classes are where we expect them to be
:checkWLS
if exist "%WL_HOME%\server\lib\weblogic.jar" goto checkJava
echo The WebLogic Server wasn't found in directory %WL_HOME%\server.
echo Please edit your script so that the WL_HOME variable points
echo to the WebLogic installation directory.
goto finish

@rem Check that java is where we expect it to be
:checkJava
if exist "%JAVA_HOME%\bin\java.exe" goto runWebLogic
echo The JDK wasn't found in directory %JAVA_HOME%.
echo Please edit your script so that the JAVA_HOME variable
echo points to the location of your JDK.
goto finish

:runWebLogic

if not "%JAVA_VM%" == "" goto noResetJavaVM
set JAVA_VM=-hotspot

:noResetJavaVM
if not "%MEM_ARGS%" == "" goto noResetMemArgs
```

```
set MEM_ARGS=-Xms200m -Xmx200m

:noResetMemArgs

@echo on

set CLASSPATH=%JAVA_HOME%\lib\tools.jar;
%WL_HOME%\server\lib\weblogic_sp.jar;
%WL_HOME%\server\lib\weblogic.jar;%CLASSPATH%

set PATH=.;%WL_HOME%\server\bin;%JAVA_HOME%\bin;%PATH%

@echo ************************************************
@echo *  To start WebLogic Server, use a username and   *
@echo *  password assigned to an admin-level user.  By  *
@echo *  default, this is user: installadministrator    *
@echo *  and password: installadministrator.  These     *
@echo *  should both be changed using the WebLogic      *
@echo *  Server console at                              *
@echo *       http://[hostname]:[port]/console          *
@echo ************************************************

@rem Start Server

@echo off
if "%ADMIN_URL%" == "" goto runAdmin
@echo on
"%JAVA_HOME%\bin\java" %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS% -classpath
 "%CLASSPATH%" -Dweblogic.Domain=%DOMAIN_NAME%
-Dweblogic.Name=%SERVER_NAME%
-Dbea.home="C:\bea"
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.management.server=%ADMIN_URL%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy"
 weblogic.Server
goto finish

:runAdmin
@echo on
"%JAVA_HOME%\bin\java" %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%
-classpath "%CLASSPATH%" -Dweblogic.Domain=%DOMAIN_NAME%
```

```
-Dweblogic.Name=%SERVER_NAME% -Dbea.home="C:\bea"
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy"
 weblogic.Server

:finish

ENDLOCAL
```

Tools

In this section, we will look at the tools involved in working with WebLogic Server.

Deployment

With Java applications, deployment has never been easy. WebLogic provides various options for deploying applications. Use the WebLogic Server Administration Console, the weblogic.Deployer utility, the Marathon utility, or auto-deployment. The weblogic.Deployer utility is further discussed in Chapter 12.

EJBGen

WebLogic 7 has an EJBGen tool that works as an EJB 2.0 code generator. While executing this tool, you are required to provide the name of a bean class file with javadoc comment tags, which will then generate the remote and home classes and the deployment descriptor files for an EJB application. This helps reduce the number of EJB files to edit and maintain. EJBGen allows editing to be limited to one file (the bean class) and annotated with javadoc tags. EJBGen is discussed further in Chapter 12.

WebLogic Builder

At times, assembling a J2EE application module, creating and editing its deployment descriptors, and later deploying it to WebLogic Server can prove to be a challenging and time-consuming task. WebLogic Builder, as shown in Figure 1-11, facilitates those challenges as a graphical tool used for assembling a J2EE application module, creating and editing its deployment descriptors, and deploying it to a WebLogic server.

WebLogic Builder provides a visual editing environment for editing an application's deployment descriptor XML files. You can view these XML files as you visually edit them in WebLogic Builder, but you won't need to make textual edits to the XML files.

Figure 1-11 shows how we have opened and accessed the descriptor files from within the EJB application module (EAR). The open descriptor files are *ejb-jar.xml* and *weblogic-ejb-jar.xml*. The changes you make to the descriptors using this tool are saved in the related archive file (JAR or EAR).

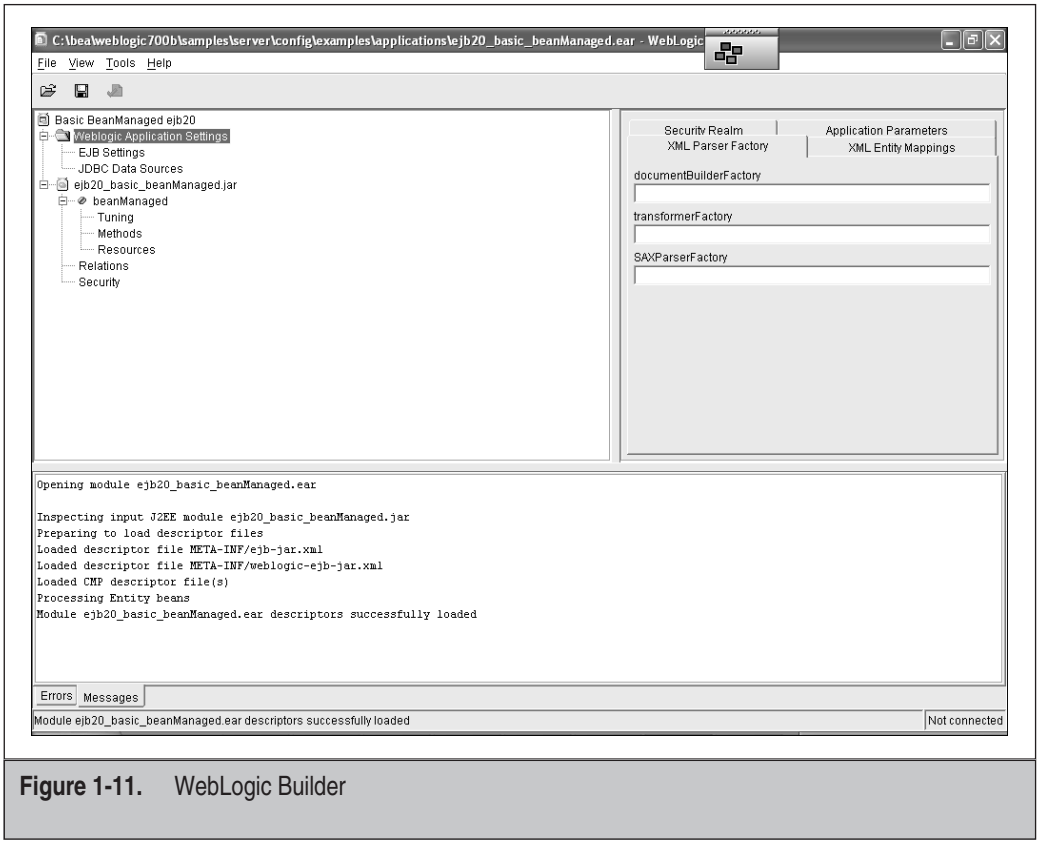


Figure 1-11. WebLogic Builder

WebLogic Workshop

WebLogic Workshop is an integrated development environment (IDE) that offers a GUI-based approach to developing distributed, interconnected, and loosely coupled enterprise-class Web services. With WebLogic Workshop, you can design Web services as you might draw them on paper and then add code to support the services' functionality. You can focus on developing your service's application logic rather than on writing code to support platform infrastructure. Figure 1-12 demonstrates the concept.

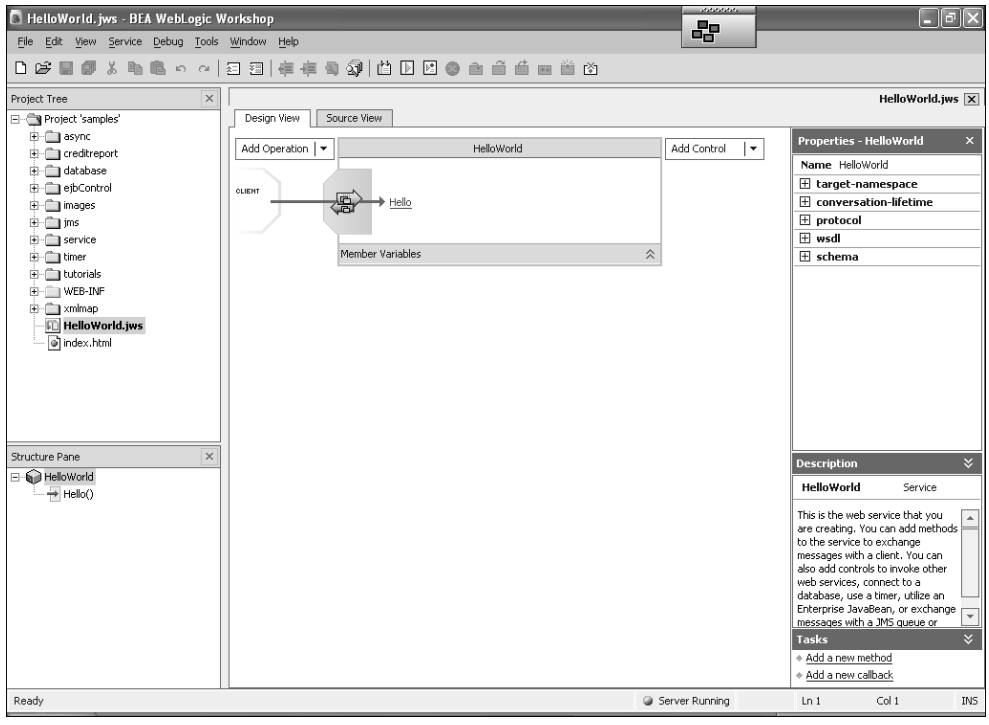


Figure 1-12. WebLogic Workshop

CONFIGURATION BASICS

At the heart of WebLogic Server is the configuration file *config.xml*. This file contains configuration information about the entire WebLogic Server domain. Figure 1-13 demonstrates the information recorded within *config.xml*.

The *config.xml* file is made up of numerous XML elements, each describing various aspects of the WebLogic Server domain. Configuration information related to various J2EE components, such as JSPs, servlets, EJBs, JDBC connections, JMS, JTA, JNDI, and so forth, is stored in this file.

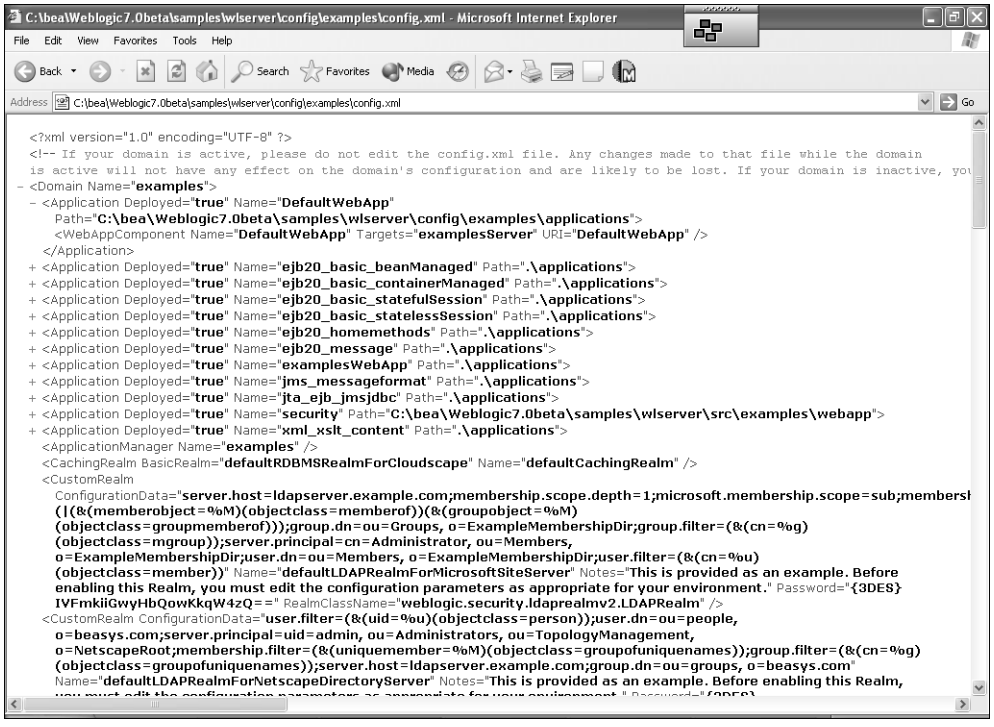


Figure 1-13. The *config.xml* file

You can manipulate *config.xml* manually by using your favorite editor and check the reflection in the behavior of the configured components when you run the server. You can also use the WebLogic Administration Console to manipulate various parameters related to the WebLogic applications or server, and later check *config.xml* for their updated values. Exercising this is handy and helps further understanding of the role of *config.xml*.

Following is a sample *config.xml* file:

Sample *config.xml*

```
<?xml version="1.0" encoding="UTF-8" ?> <!--If your domain is active,
please do not edit the config.xml file. Any changes made to that file
while the domain is active will not have any effect on the domain's
configuration and are likely to be lost. If your domain is inactive,
you may edit this file with an XML editor. If you do so, please refer
to the BEA WebLogic Server Configuration Reference  documentation
available from http://edocs.bea.com/wls/docs61/reference.html.  In
general, we recommend that changes to your configuration file be made
```

```
through the Administration Console.-->
<Domain Name="mydomain">
      <Application Deployed="true" Name="DefaultWebApp"
Path=".\applications" TwoPhase="false">
            <WebAppComponent Name="DefaultWebApp" Targets="myserver"
 URI="DefaultWebApp" />
      </Application>
      <Application Deployed="true" Name="certificate"
Path=".\applications" TwoPhase="false">
            <WebAppComponent Name="certificate"
Targets="myserver" URI="certificate.war" />
      </Application>
      <ApplicationManager Name="mydomain" />
      <FileRealm Name="wl_default_file_realm" />
      <JTA Name="mydomain" />
      <Log FileName=".\wl-domain.log" Name="mydomain" />
      <PasswordPolicy Name="wl_default_password_policy" />
      <Realm FileRealm="wl_default_file_realm"
Name="wl_default_realm" />
      <SNMPAgent Name="mydomain" />
      <Security GuestDisabled="false" Name="mydomain"
PasswordPolicy="wl_default_password_policy" Realm="wl_default_realm"
RealmSetup="true" />
      <SecurityConfiguration Credential=
"{3DES}3WOUyKIzNNk3WJRlIpgHtfjs
 57CUC3t2cXoeQqSzH1w4G4V/jhBQAn
 v8jQNAwV2sOwz2IXE7d5B34d05T08j
 0f5SwTzS9xBH" Name="mydomain" />
      <Server ListenPort="7001" Name="myserver"
NativeIOEnabled="true" ServerVersion="7.0.0.0">
            <COM Name="myserver" />
            <ExecuteQueue Name="default" ThreadCount="15" />
            <JTAMigratableTarget Cluster=""
Name="myserver" UserPreferredServer="myserver" />
            <KernelDebug Name="myserver" />
            <Log FileName=".\myserver\myserver.log" Name="myserver" />
            <SSL Enabled="true" ListenPort="7002"
Name="myserver" ServerCertificateChainFileName="ca.pem"
ServerCertificateFileName="democert.pem"
ServerKeyFileName="demokey.pem" />
            <ServerDebug Name="myserver" />
            <ServerStart Name="myserver" />
            <SystemDataStore Credential=
"{3DES}+cGRcNRxlWhCRWY+Gd1dL3rO
ZqEUWe+bfKIIIfJStns=" Name="myserver" />
           <WebServer DefaultWebApp="DefaultWebApp" LogFileName=".\myserver\access.log"
LoggingEnabled="true" Name="myserver" />
      </Server>
</Domain>
```

CHECKLIST

In this chapter, you acquired necessary information about WebLogic Server and useful skills required for a quick start to administering the WebLogic application server. Here are the key aspects related to WebLogic Server:

- ☐ HTTP service is built into WebLogic Server.
- ☐ WebLogic Server is the platform of choice for deploying and running J2EE enterprise Java applications.
- ☐ WebLogic Server has two types of containers: the Web container and the EJB container.
- ☐ Java servlets, JSPs, and static pages can be deployed and run in Web containers.
- ☐ EJBs can be deployed and run in EJB containers.
- ☐ With J2EE 1.3, resource adapters can now be used to connect to EIS.
- ☐ Development and production environments are different animals.
- ☐ Setting the `CLASSPATH` variable enables applications to locate the relevant application-specific Java classes.
- ☐ The *config.xml* file contains all configurations settings for the WebLogic domain.