Chapter 1

Introducing Application Servers

In This Chapter

▶ Understanding the role of application servers

▶ Meeting the J2EE family of technologies

▶ Outlining the major features of WebLogic

In the most general sense, a *server* is a program that provides information to a client that requests that information. Sometimes a server is a computer used to centralize resources so that they can be shared by a number of different users. For instance, file servers centralize file storage, database servers centralize data storage, and web servers centralize the distribution of web content. In a similar vein, an application server centralizes key programming tasks. Doing so has many advantages, as you will discover.

In this chapter, you find out about application servers, in particular BEA's WebLogic Server. In a recent Gartner study, BEA WebLogic Server had 34 percent of the application server market share — the largest market share of any single vendor. BEA Systems is at the forefront of market developments and support of new standards.

WebLogic is not the only application server on the market. WebLogic's main competitors are IBM's WebSphere and JBoss, an open-source application server released under the LGPL license. In addition to these two Java-based application servers, WebLogic faces non-Java competition, mainly from the growing Microsoft .NET family of products.

Application Server Basics

Enterprise JavaBeans (EJB) is a technology for developing, assembling, deploying, and managing distributed applications in an enterprise environment. This basically means that EJB provides a Java framework for executing objects residing on different machines over a distributed network. This is a powerful capability: It enables a user to harness the power of different machines transparently, in a single application.

A machine hosting and executing an EJB object is called an *EJB application server.* WebLogic, as an EJB application server, also acts as a container for EJBs. A *container* provides a management system for EJB objects. An efficient container removes the need for users and developers (to a certain extent) to be concerned about exactly how an object will be used. Put another way, an EJB application server provides APIs and interfaces, and an EJB is like a plug-in that provides business logic for a specific application. As a developer, you're writing modules (EJBs) that are dropped into the application server, which then loads and runs the EJBs when needed.

Servers work closely with clients. A *client* requests information from a server or requests that a server do something. The server, acting on the request, sends the requested information to the client or does what it is asked to do.

BEA WebLogic Server, as an EJB application server, interacts with clients in a similar manner. The machine that requests WebLogic to run an EJB program is the client. This client program can be a stand-alone Java program or another server. (Often web servers are the clients for the services of EJBs.) EJBs allow a busy web server to focus on what it was designed to do: serve web pages. The web server calls upon EJBs, which reside on an application server, to perform business-specific tasks, such as retrieving data from a database.

This division of labor is the key reason to use an application server. Dividing the task between the client and the application server results in three immediate advantages:

- Reliability
- Scalability
- Modularity

Achieving reliability through redundancy

You can run an application on your desktop machine only as long as your machine is operational. In other words, if your machine "hangs" (becomes locked) or the power goes off, you can't continue to work. Application servers, on the other hand, can offer a more reliable way of running an application through a concept called *redundancy.* This simply means that you add multiple servers, instruct them to act together as if they were a single server, and then allow clients to access them. If one of the servers becomes unavailable, the other servers pick up the slack and respond to the needs of the clients.

You can also work on an ailing server without disturbing the other servers. You are free to reboot the crashed application server without affecting the stability of the remaining application servers. Using multiple application servers in this way can increase the reliability of your application.

Making applications scalable

As more and more clients make requests of an application server, more and more demands are placed on that server. As the overall demands become greater, the capability of the server to quickly fulfill each individual request decreases. One solution to this problem is to add more horsepower to the machine used to run the application server — perhaps more memory, a faster hard drive, or even a faster CPU. A better solution, however, is to add another server, clustering it with the existing server. Now the deluge of client requests can be serviced by two machines acting as one. Need more power? Add a third, fourth, or fifth machine. This is the essence of *scalability.*

As requests for services come in from the clients, the cluster automatically dispatches these requests to the least busy of the application servers. This allows you to increase the capacity of your application by simply adding additional application servers rather than going through the costly process of upgrading a production server. As a bonus, the additional servers also increase the reliability of your system.

Improving modularity

Modularity has long been one of the chief design goals of computer programming. *Modular program design* breaks the program into smaller units, or modules, that are developed separately. Often these modules can be reused across several applications. Object oriented programming (OOP) was created to facilitate the creation of modular programs, among other design goals.

One of the most fundamental ways of making a program modular is to separate *presentation logic* — the part of the program that interacts with the user — from *business logic* — the part of the program that makes decisions and performs calculations. Presentation logic should be housed in the web server, because the web server is responsible for transmitting the HTML that will be presented to the user. Business logic should be housed in the application server so that it can be reused by any web pages that may need it. The same business logic is often needed across many web pages. For example, the business logic to update inventory would be reused on any page that affects inventory.

An application server enables this separation. Business logic is placed in EJBs. The application server executes the EJBs, and the results are sent to the presentation program running on the web server.

J2EE, Java's Approach to Application Servers

Java 2 Platform, Enterprise Edition (J2EE) contains additions to the Java environment that Sun Microsystems created to facilitate such enterprise concepts as application servers. Sun has defined a specific way in which to build application servers for Java. One advantage to this approach is that content you develop for WebLogic Server can be used also with other J2EE application servers. In other words, you can migrate the content to another J2EE application server, if needed.

J2EE is not just one technology, but rather a collection of technologies. Sun defines standards embodied as J2EE, which other vendors implement. For example, WebLogic implements the following J2EE components:

- JavaServer Pages (JSP)
- Enterprise JavaBeans (EJB)
- Java Transaction Service (JTS)
- Java Message Service (JMS)
- Java Naming and Directory Interface (JNDI)

In other chapters, you find out more about these components of J2EE. In this section, I briefly review the function of each of these to give you an overview of how they fit together.

JavaServer Pages

JavaServer Pages (*JSP*) allow you to embed Java code directly into HTML-like documents. JSP has access to nearly all the core features of the Java programming language, except you're returning only streams back to the user's browser. This allows you to construct complex applications using only JSP. However, just because you can construct complex JSP-based applications does not mean that you should. JSP is best restricted to presentation logic, with more complex business logic delegated to EJBs.

Enterprise JavaBeans

Enterprise JavaBeans (EJB) technology allows code to be executed on a remote system. This remote system is the application server. EJB is commonly used to isolate business logic from presentation logic, which usually consists of JSP. EJB coordinates access with the database and shields higher levels, such as JSP, from the need to directly access the database. In this way, if you were to change database servers or the format of your database, all code related to data access would be in one location.

Java Transaction Service

Java Transaction Service (JTS) is a transaction manager that allows requests to be segmented into transactions. These transactions succeed or fail as a whole. This prevents partial transactions from persisting if only a part of the transaction is successful.

Java Message Service

The *Java Message Service (JMS)* API was developed to allow Java applications to be message driven. A message-compatible EJB can receive and generate messages. These messages can contain any data needed by the program. Messaging is asynchronous, so considerable time can elapse before a response message is received, if at all. JMS also allows messages to be saved to a message store, such as a file or a database.

Java Naming and Directory Interface

Java Naming and Directory Interface (JNDI) is a standard extension to the Java platform that provides naming and directory information to Java programs. This allows EJB and other resources to have names that can be looked up by their client programs. JNDI is a high-level standard and can use any number of underlying name and directory services.

Enterprise applications

Enterprise applications tie many of the previously mentioned components together into one application. An enterprise application is most commonly made up of a web application and any EJB that may be used by that web application. The entire enterprise application is packaged as a single archive file, which can be easily deployed to a server such as WebLogic. This allows for easy packaging, distribution, and deployment of your enterprise applications.

Major Features of WebLogic Server

As mentioned, WebLogic is the most popular application server available for Java. WebLogic has gained this popularity due, in part, to a full set of features. In this section, you are introduced to some of these features. In other chapters, they are described in much greater detail.

REMEMBER

Throughout this text, I refer to BEA's WebLogic Server product simply as *WebLogic.* BEA, however, uses the term *WebLogic* to refer to a family of products, including WebLogic Portal, WebLogic Integration, WebLogic Workshop, and WebLogic Express. The popularity of the core WebLogic Server product, however, has led to the shortening of the name to simply WebLogic in many circles.

Platform support

WebLogic can run on many platforms, including Windows and many flavors of UNIX. WebLogic is available also for many large mainframe computer systems, providing WebLogic with greater processing power and scalability. The extensive platform support of WebLogic allows you to mix and match technologies. For example, you might run WebLogic on a mainframe computer system, backing it up with a cluster of less expensive machines that run the same applications. Further, you can test your application on less expensive machines and run your production system on more expensive, higher-bandwidth hardware.

Web applications

Although WebLogic is most commonly thought of as an application server, it can also handle many web server functions. This means WebLogic could be used as an all-in-one solution. JavaServer Pages (JSP) is one of the most common forms of server-side Java programming. WebLogic includes the capability to execute JSP. You can to create web applications in WebLogic that make use of technologies such as JSP and custom tag libraries. Web applications are covered in Chapter 5.

EJB support

Perhaps the most basic feature of a Java-based application server is support for Enterprise JavaBeans (EJB). WebLogic includes extensive support for the five types of EJB:

- Stateless bean
- Stateful bean
- Message bean
- Container-managed persistence (CMP) entity bean
- Bean-managed persistence (BMP) entity bean

Additionally, WebLogic makes other important services available to these beans, such as database connection pooling and naming services. EJB support is discussed in Chapters 6 and 7.

Database connectivity

Databases are often the heart of any serious application. Because of this, WebLogic includes extensive support for relational databases. One of the most important features is *database connection pooling.* This allows WebLogic — instead of individual EJBs — to manage connections to the database.

Database connections are an expensive resource. Processor cycles and extensive network communication are required to open and close these connections, and this can slow down other operations. By using a database connection pool, WebLogic can reuse its pool of open database connections, freeing the application from the overhead of constantly creating and destroying database connections. Database connectivity is discussed more fully in Chapter 12.

Web services

Web services are a new technology that provides a more uniform way of accessing the components of an application. Web services allow your application to receive XML messages from other applications and respond to those requests using XML. This means other applications can make use of your application using only the HTTP protocol.

XML messages are sent and received using the Simple Object Access Protocol (SOAP), a W3C standard that specifies how web services should be accessed by their client programs. By supporting a standard protocol such as SOAP, many different systems can access the web services that you make available through WebLogic Server. Web services are discussed in Chapter 9. Accessing web services is discussed in Chapter 10.

One of the new features of WebLogic (as of Version 7) is WebLogic Workshop, which enables someone who is not familiar with J2EE to construct web services. WebLogic Workshop provides a number of tools and frameworks to make designing web services easier. WebLogic Workshop is discussed in Chapter 11.

Clustering

Clustering is the capability to chain together many individual application servers. These application servers are clones of each other, performing the same task. The clustering capabilities of WebLogic enable these servers to handle requests even though some of the application servers may fail. This greatly increases the reliability of your application.

Clustering also allows your web application to become very scalable. Because you now have many application servers handling requests from clients, you can handle a greater number of incoming requests. Clustering is discussed in more detail in Chapter 16.

Security

Security is a major concern in any application — and when your application is accessible through the Internet, the need for security increases. WebLogic can help you with three specific areas of security:

- **Securing your data transmissions.** Data transmissions are secured using SSL/HTTPS. This prevents a hacker from accessing data packets as they are transmitted between the browser and the web server.
- **Controlling access by users.** You may want to restrict some users from accessing the overall system and restrict other users from accessing only certain parts of the system. WebLogic provides features that allow you to define users and control exactly what they have access to.
- **Verifying administrators.** WebLogic's Administration Console enables you to easily configure your server remotely. Unfortunately, this also means that a hacker can configure your system remotely. WebLogic provides security to all configuration programs to limit access by unauthorized users. Security is discussed in Chapter 18.