BEA WebLogic Server

and BEA WebLogic Express

Introduction to BEA WebLogic Server

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Introduction to BEA WebLogic Server

Part Number	Document Date	Software Version
860-001002-006	June 4, 2001	WebLogic Server Version 6.0

Contents

2. WebLogic Server Services

Index

About This Document

This document introduces BEA WebLogic Server™ features and describes the architecture of applications that run on the WebLogic Server platform.

What You Need to Know

This document is intended mainly for application developers who want to build e-commerce applications using the Java 2 Platform, Enterprise Edition (J2EE) from Sun Microsystems. It is assumed that the audience for this document know Web technologies, object-oriented programming techniques, and the Java programming language.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com.

Contact Us!

Your feedback on the BEA WebLogic Server documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Server documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Server 6.0 release.

If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number

- Your company name and company address

- Your machine type and authorization codes

- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Identifies variables in code. *Example*: `String CustomerName;`
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

Convention	Item
[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: ``` java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send] ```
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. *Example*: ``` java weblogic.deploy [list|deploy|undeploy|update] password {application} {source} ```
...	Indicates one of the following in a command line: ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed.
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 What Is WebLogic Server?

The following sections provide an overview of the WebLogic Server™ e-commerce platform:

- The WebLogic Server Solution

- What Is WebLogic Express?

- WebLogic Server Application Architecture

- Software Component Tiers

- Application Logic Layers

The WebLogic Server Solution

Today's business environment demands Web and e-commerce applications that accelerate your entry into new markets, help you find new ways to reach and retain customers, and allow you to introduce new products and services quickly. To build and deploy these new solutions, you need a proven, reliable e-commerce platform that can connect and empower all types of users while integrating your corporate data, mainframe applications, and other enterprise applications in a powerful, flexible end-to-end e-commerce solution. Your solution must provide the performance, scalability, and high availability needed to handle your most critical enterprise-scale computing.

As the industry-leading e-commerce transaction platform, WebLogic Server allows you to quickly develop and deploy reliable, secure, scalable and manageable applications. It manages system-level details so you can concentrate on business logic and presentation.

J2EE Platform

WebLogic Server contains Java 2 Platform, Enterprise Edition (J2EE) technologies. J2EE is the standard platform for developing multitier enterprise applications based on the Java programming language. The technologies that make up J2EE were developed collaboratively by Sun Microsystems and other software vendors, including BEA Systems.

J2EE applications are based on standardized, modular components. WebLogic Server provides a complete set of services for those components and handles many details of application behavior automatically, without requiring programming.

Application Deployment Across Distributed, Heterogenous Environments

WebLogic Server provides essential features for developing and deploying mission-critical e-commerce applications across distributed, heterogeneous computing environments. These features include the following:

- Standards leadership—Comprehensive Enterprise Java support to ease implementation and deployment of application components. WebLogic Server is the first independently developed Java application server to achieve J2EE certification.

- Rich client options—WebLogic Server supports Web browsers and other clients that use HTTP; Java clients that use RMI (Remote Method Invocation) or IIOP (Internet Inter-ORB Protocol); and mobile devices that use (WAP) Wireless Access Protocol. Connectors from BEA and other companies enable virtually any client or legacy application to work with a WebLogic Server application.

- Enterprise e-business scalability—Critical resources are used efficiently and high availability is ensured through the use of Enterprise JavaBean business

components and mechanisms such as WebLogic Server clustering for dynamic Web pages, backend resource pooling, and connection sharing.

- Robust administration—WebLogic Server offers a Web-based Administration Console for configuring and monitoring WebLogic Server services. A command-line interface for configuration makes it convenient to administer WebLogic Servers with scripts.

- E-commerce-ready security—WebLogic Server provides Secure Sockets Layer (SSL) support for encrypting data transmitted across WebLogic Server, clients, and other servers. WebLogic security realms feature user authentication and authorization for all WebLogic Server services. External security stores, such as Lightweight Directory Access Protocol (LDAP) servers, can be adapted to WebLogic realms, enabling single sign-on for the enterprise. The Security Service Provider Interface makes it possible to extend WebLogic Security services and to implement WebLogic Security features in applications.

- Maximum development and deployment flexibility—WebLogic Server provides tight integration with and support for leading databases, development tools, and other environments.

What Is WebLogic Express?

BEA WebLogic Express™ is a scalable platform that serves dynamic content and data to Web and wireless applications. WebLogic Express incorporates the presentation and database access services from WebLogic Server, enabling developers to create interactive and transactional e-business applications quickly and to provide presentation services for existing applications.

WebLogic Express offers many of the services and APIs available with WebLogic Server, including WebLogic JDBC features, JavaServer Pages (JSP), servlets, Remote Method Invocation (RMI), and Web server functionality.

WebLogic Express differs from WebLogic Server in that WebLogic Express does not provide Enterprise JavaBeans (EJB), Java Message Services (JMS), or the two-phase commit protocol for transactions.

WebLogic Server Application Architecture

WebLogic Server is an application server: a platform for developing and deploying multitier distributed enterprise applications. WebLogic Server centralizes application services such as Web server functionality, business components, and access to backend enterprise systems. It uses technologies such as caching and connection pooling to improve resource use and application performance. WebLogic Server also provides enterprise-level security and powerful administration facilities.

WebLogic Server operates in the middle tier of a multitier (or *n*-tier) architecture. A multitier architecture determines where the software components that make up a computing system are executed in relation to each other and to the hardware, network, and users. Choosing the best location for each software component lets you develop applications faster; eases deployment and administration; and provides greater control over performance, utilization, security, scalability, and reliability.

WebLogic Server implements J2EE, the Java Enterprise standard. Java is a network-savvy, object-oriented programming language, and J2EE includes component technologies for developing distributed objects. This functionality adds a second dimension to the WebLogic Server application architecture—a layering of application logic, with each layer selectively deployed among WebLogic Server J2EE technologies.

The next two sections describe these two views of WebLogic Server architecture: software tiers and application logic layers.

Software Component Tiers

The software components of a multitier architecture consist of three tiers:

■ The *client tier* contains programs executed by users, including Web browsers and network-capable application programs. These programs can be written in virtually any programming language.

■ The *middle tier* contains WebLogic Server and other servers that are addressed directly by clients, such as existing Web servers or proxy servers.

■ The *backend tier* contains enterprise resources, such as database systems, mainframe and legacy applications, and packaged enterprise resource planning (ERP) applications.

Client applications access WebLogic Server directly, or through a Web server or proxy server. WebLogic Server connects with backend services on behalf of clients, but clients do not directly access backend services.

Figure 1-1 illustrates the three tiers of the WebLogic Server architecture.

Figure 1-1   Three-Tier Architecture
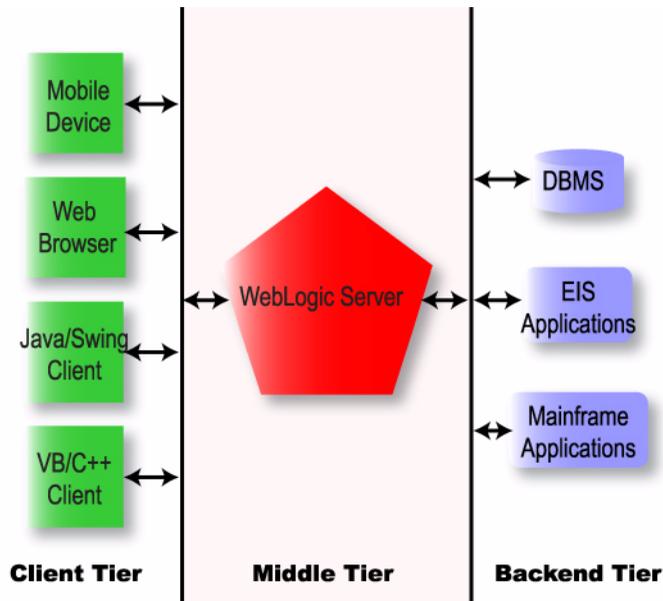
Client-Tier Components

WebLogic Server clients use standard interfaces to access WebLogic Server services. WebLogic Server has complete Web server functionality, so a Web browser can request pages from WebLogic Server using the Web's standard HTTP protocol. WebLogic Server servlets and JavaServer Pages (JSPs) produce the dynamic, personalized Web pages required for advanced e-commerce Web applications.

Client programs written in Java may include highly interactive graphical user interfaces built with Java Swing classes. They can also access WebLogic Server services using standard J2EE APIs.

All these services are also available to Web browser clients by deploying servlets and JSP pages in WebLogic Server.

CORBA-enabled client programs written in Visual Basic, C++, Java, and other programming languages can execute WebLogic Server Enterprise JavaBeans and RMI (Remote Method Invocation) classes using WebLogic RMI-IIOP. Client applications written in any language with support for the HTTP protocol can access any WebLogic Server service through a servlet.

Middle-Tier Components

The middle tier includes WebLogic Server and other Web servers, firewalls, and proxy servers that mediate traffic between clients and WebLogic Server. The Nokia WAP server, part of the BEA mobile commerce solution, is an example of another middle tier server that provides connectivity between wireless devices and WebLogic Server.

Applications based on a multitier architecture require reliability, scalability, and high performance in the middle tier. The application server you select for the middle tier is, therefore, critical to the success of your system.

The WebLogic Server Cluster option allows you to distribute client requests and backend services among multiple cooperating WebLogic Servers. Programs in the client tier access the cluster as if it were a single WebLogic Server. As the workload increases, you can add WebLogic Servers to the cluster to share the work. The cluster uses a selectable load-balancing algorithm to choose a WebLogic Server in the cluster that is capable of handling the request.

When a request fails, another WebLogic Server that provides the requested service can take over. Failover is transparent whenever possible, which minimizes the amount of code that must be written to recover from failures. For example, servlet session state can be replicated on a secondary WebLogic Server so that if the WebLogic Server that is handling a request fails, the client's session can resume uninterrupted on the secondary server. WebLogic EJB, JMS, JDBC, and RMI services are all implemented with clustering capabilities.

Backend-Tier Components

The backend tier contains services that are accessible to clients only through WebLogic Server. Applications in the backend tier tend to be the most valuable and mission-critical enterprise resources. WebLogic Server protects them by restricting direct access by end users. With technologies such as connection pools and caching, WebLogic Server uses backend resources efficiently and improves application response.

Backend services include databases, enterprise resource planning (ERP) systems, mainframe applications, legacy enterprise applications, and transaction monitors. Existing enterprise applications can be integrated into the backend tier using the Java Connector Architecture (JCA) specification from Sun Microsystems. WebLogic Server makes it easy to add a Web interface to an integrated backend application.

A database management system is the most common backend service, required by nearly all WebLogic Server applications. WebLogic EJB and WebLogic JMS typically store persistent data in a database in the backend tier.

A JDBC connection pool, defined in WebLogic Server, opens a predefined number of database connections. Once opened, database connections are shared by all WebLogic Server applications that need database access. The expensive overhead associated with establishing a connection is incurred only once for each connection in the pool, instead of once per client request. WebLogic Server monitors database connections, refreshing them as needed and ensuring reliable database services for applications.

WebLogic Enterprise Connectivity, which provides access to BEA WebLogic Enterprise™ systems, and Jolt® for WebLogic Server, which provides access to BEA Tuxedo® systems, also use connection pools to enhance system performance.

Application Logic Layers

WebLogic Server implements J2EE component technologies and services. J2EE component technologies include servlets, JSP Pages, and Enterprise JavaBeans. J2EE services include access to standard network protocols, database systems, and messaging systems. To build a WebLogic Server application, you must create and assemble components, using the service APIs when necessary.

Components are executed in the WebLogic Server Web container or EJB container. Containers provide the life cycle support and services defined by the J2EE specifications so that the components you build do not have to handle underlying details.

Web components provide the presentation logic for browser-based J2EE applications. EJB components encapsulate business objects and processes. Web applications and EJBs are built on J2EE application services, such as JDBC, JMS (Java Messaging Service), and JTA (Java Transaction API).

Figure 1-2 illustrates WebLogic Server component containers and application services.

Figure 1-2   Application Logic Layers
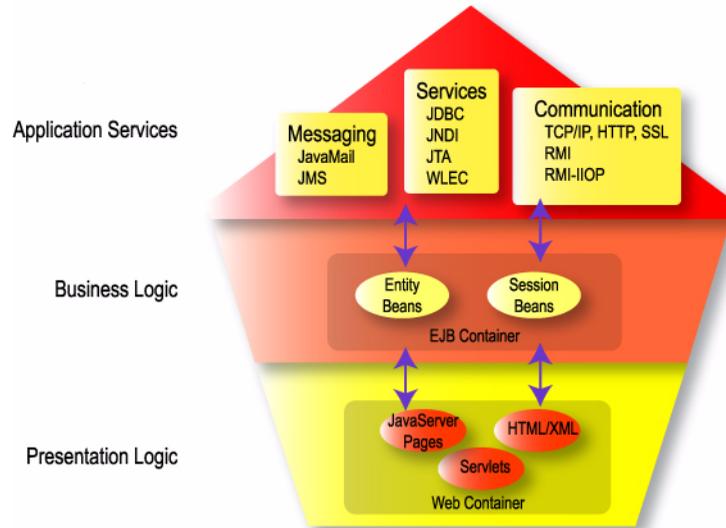
The following sections discuss the presentation layer, business logic, and application services.

Presentation Logic Layer

The presentation layer includes an application's user interface and display logic. Most J2EE applications use a Web browser on the client machine because it is much easier than deploying client programs to every user's computer. In this case, the presentation logic is the WebLogic Server Web container. Client programs written in any programming language, however, must contain either logic to render HTML or their own presentation logic.

Web Browser Clients

Web-based applications built with standard Web technologies are easy to access, maintain, and port. Web browser clients are standard for e-commerce applications.

In Web-based applications, the user interface is represented by HTML documents, JavaServer Pages (JSP), and servlets. The Web browser contains the logic to render the Web page on the user's computer from the HTML description.

JavaServer Pages (JSP) and servlets are closely related. Both produce dynamic Web content by executing Java code on WebLogic Server each time they are invoked. The difference between them is that JSP is written with an extended version of HTML, and servlets are written with the Java programming language.

JSP is convenient for Web designers who know HTML and are accustomed to working with an HTML editor or designer. Servlets, written entirely in Java, are more suited to Java programmers than to Web designers. Writing a servlet requires some knowledge of the HTTP protocol and Java programming. A servlet receives the HTTP request in a request object and writes HTML (usually) on its result object.

JSP pages are converted to servlets before they are executed on WebLogic Server, so ultimately JSP pages and servlets are different representations of the same thing. JSP pages are deployed on WebLogic Server the same way an HTML page is deployed. The `.jsp` file is copied into a directory served by WebLogic Server. When a client requests a `.jsp` file, WebLogic Server checks whether the page has been compiled or has changed since it was last compiled. If needed, it calls the WebLogic JSP compiler, which generates Java servlet code from the `.jsp` file, and then it compiles the Java code to a Java class file.

Non-Browser Clients

A client program that is not a Web browser must supply its own code for rendering the user interface. Non-browser clients usually contain their own presentation and rendering logic, depending on WebLogic Server only for business logic and access to backend services. This makes them more difficult to develop and deploy and less suited for Internet-based e-commerce applications than browser-based clients.

Client programs written in Java can use any WebLogic Server service over Java RMI (Remote Method Invocation). RMI allows a client program to operate on a WebLogic Server object the same way it would operate on a local object in the client. Because RMI hides the details of making calls over a network, J2EE client code and server-side code are very similar.

Java programs can use the Java Swing classes to create powerful and portable user interfaces. Although by using Java you can avoid portability issues, you cannot use WebLogic Server services over RMI unless the WebLogic Server classes are installed

on the client. This means that Java RMI clients are not suited to e-commerce applications. They can be used effectively, however, in enterprise applications in which an internal network makes installation and maintenance viable.

Client programs written in languages other than Java and Java client programs that do not use WebLogic Server objects over RMI can access WebLogic Server using HTTP or RMI-IIOP.

HTTP is the standard protocol for the Web. It allows a client to make different types of requests to a server and to pass parameters to the server. A servlet on WebLogic Server can examine client requests, retrieve parameters from the request, and prepare a response for the client, using any WebLogic Server service. For example, a servlet might respond to a client program with an XML business document. Thus an application can use servlets as gateways to other WebLogic Server services.

WebLogic RMI-IIOP allows CORBA-enabled programs to execute WebLogic Server enterprise beans and RMI classes as CORBA objects. The WebLogic Server RMI and EJB compilers can generate IDL (Interface Definition Language) for RMI classes and enterprise beans. IDL generated this way is compiled to create skeletons for an ORB (Object Request Broker) and stubs for the client program. WebLogic Server parses incoming IIOP requests and dispatches them to the RMI run-time system.

Business Logic Layer

Enterprise JavaBeans are the business logic components for J2EE applications. The WebLogic Server EJB container hosts enterprise beans, providing life cycle management and services such as caching, persistence, and transaction management.

There are three types of enterprise beans: entity beans, session beans, and message-driven beans. The following sections describe each type in detail.

Entity Beans

An entity bean represents an object that contains data, such as a customer, an account, or an inventory item. Entity beans contain data values and methods that can be invoked on those values. The values are saved in a database (using JDBC) or some other data store. Entity beans can participate in transactions involving other enterprise beans and transactional services.

Entity beans are often mapped to objects in databases. An entity bean can represent a row in a table, a single column in a row, or an entire table or query result. Associated with each entity bean is a unique primary key used to find, retrieve, and save the bean.

An entity bean can employ one of the following:

- Bean-managed persistence—The bean contains code to retrieve and save persistent values.

- Container-managed persistence—The EJB container loads and saves values on behalf of the bean.

When container-managed persistence is used, the WebLogic EJB compiler can generate JDBC support classes to map an entity bean to a row in a database. Other container-managed persistence mechanisms are available. For example, TOPLink for BEA WebLogic Server, from The Object People (`http://www.objectpeople.com`), provides persistence for an object relational database.

Entity beans can be shared by many clients and applications. An instance of an entity bean can be created at the request of any client, but it does not disappear when that client disconnects. It continues to live as long as any client is actively using it. When the bean is no longer in use, the EJB container may passivate it: that is, it may remove the live instance from the server.

Session Beans

A session bean is a transient EJB instance that serves a single client. Session beans tend to implement procedural logic; they embody actions more than data.

The EJB container creates a session bean at a client's request. It then maintains the bean as long as the client maintains its connection to the bean. Sessions beans are not persistent, although they can save data to a persistent store if needed.

A session bean can be stateless or stateful. Stateless session beans maintain no client-specific state between calls and can be used by any client. They can be used to provide access to services that do not depend on the context of a session, such as sending a document to a printer or retrieving read-only data into an application.

A stateful session bean maintains state on behalf of a specific client. Stateful session beans can be used to manage a process, such as assembling an order or routing a document through a workflow process. Because they can accumulate and maintain state through multiple interactions with a client, session beans are often the controlling

objects in an application. Because they are not persistent, session beans must complete their work in a single session and use JDBC, JMS, or entity beans to record the work permanently.

Message-Driven Beans

Message-driven beans, introduced in the EJB 2.0 specification, are enterprise beans that handle asynchronous messages received from JMS Message Queues. JMS routes messages to a message-driven bean, which selects an instance from a pool to process the message.

Message-driven beans are managed in the WebLogic Server EJB container. Because they are not called directly by user-driven applications, they cannot be accessed from an application using an EJB home. A user-driven application can, however, instantiate a message-driven bean indirectly by sending a message to the bean's JMS Queue.

Application Services Layer

WebLogic Server supplies the fundamental services that allow components to concentrate on business logic without concern for low-level implementation details. It handles networking, authentication, authorization, persistence, and remote object access for EJBs and servlets. Standard Java APIs provide portable access to other services that an application can use, such as database and messaging services.

Network Communications Technologies

Client applications connect with WebLogic Server using standard networking protocols over TCP/IP. WebLogic Server listens for connection requests at a network address that can be specified as part of a Uniform Resource Identifier (URI).

A URI is a standardized string that specifies a resource on a network, including the Internet. It contains a protocol specifier called a *scheme*, the network address of the server, the name of the desired resource, and optional parameters. The URL you enter in a Web browser, for example, `http://www.bea.com/index.html`, is the most familiar URI format.

Web-based clients communicate with WebLogic Server using the HTTP protocol. Java clients connect using Java RMI (Remote Method Invocation), which allows a Java client to execute objects in WebLogic Server. CORBA-enabled clients access WebLogic Server RMI objects using RMI-IIOP, which allows them to execute WebLogic Server objects using standard CORBA protocols.

The scheme in a URI determines the protocol for network exchanges between a client and WebLogic Server. I

Table 1-1 Network Protocols

Scheme	Protocol
HTTP	HyperText Transfer Protocol. Used by Web browsers and HTTP-capable programs.
HTTPS	Hypertext Transfer Protocol over Secure Sockets Layer (SSL). Used by Web browsers and HTTPS-capable client programs.
T3	WebLogic T3 protocol for Java-to-Java connections, which multiplexes JNDI, RMI, EJB, JDBC, and other WebLogic services over a network connection.
T3S	WebLogic T3 protocol over Secure Sockets Layer (SSL).
IIOP	Internet Inter-ORB protocol, used by CORBA-enabled Java clients to execute WebLogic RMI objects over IIOP. Other CORBA clients connect to WebLogic Server with a CORBA naming context instead of a URI for WebLogic Server.

The following sections provide more information about these protocols.

HTTP

HTTP, the standard protocol of the World Wide Web, is a request-response protocol. A client issues a request that includes a URI. The URI begins with `http://` and the WebLogic Server address, and the name of a resource on WebLogic Server, such as an HTML page, servlet, or JSP page. If the resource name is omitted, WebLogic Server returns the default Web page, usually `index.html`. The header of an HTTP request includes a command, usually GET or POST. The request can include data parameters and message content.

WebLogic Server always responds to an HTTP request by executing a servlet, which returns results to the client. An HTTP servlet is a Java class that can access the contents of an HTTP request received over the network and return an HTTP-compliant result to the client.

WebLogic Server directs a request for an HTML page to the built-in `File` servlet. The `File` servlet looks for the HTML file in the document directory of the WebLogic Server file system. A request for a custom-coded servlet executes the corresponding Java class on WebLogic Server. A request for a JSP page causes WebLogic Server to compile the JSP page into a servlet, if it has not already been compiled, and then to execute the servlet, which returns results to the client.

T3

T3 is an optimized protocol used to transport data between WebLogic Server and other Java programs, including clients and other WebLogic Servers. WebLogic Server keeps track of every Java Virtual Machine (JVM) with which it connects, and creates a single T3 connection to carry all traffic for a JVM.

For example, if a Java client accesses an enterprise bean and a JDBC connection pool on WebLogic Server, a single network connection is established between the WebLogic Server JVM and the client JVM. The EJB and JDBC services can be written as if they had sole use of a dedicated network connection because the T3 protocol invisibly multiplexes packets on the single connection.

T3 is an efficient protocol for Java-to-Java applications because it avoids unnecessary network connection events and uses fewer OS resources. The protocol also has internal enhancements that minimize packet sizes

RMI

Remote Method Invocation (RMI) is the standard Java facility for distributed applications. RMI allows one Java program, called the *server*, to publish Java objects that another Java program, called a *client*, can execute. In most applications, WebLogic Server is the RMI server and a Java client application is the client. But the roles can be reversed; RMI allows any Java program to play the role of server.

RMI architecture is similar to the CORBA architecture. To create a remote object, a programmer writes an interface for a Java class that defines the methods that may be executed by a remote client. The WebLogic Server RMI compiler, `rmic`, processes the interface, producing RMI stub and skeleton classes. The remote class, stubs, and skeletons are installed in WebLogic Server.

A Java client looks up a remote object in WebLogic Server using the Java Naming and Directory Interface (JNDI), which is described later in this section. JNDI establishes a connection to WebLogic Server, looks up the remote class, and returns the stubs to the client.

The client executes a stub method as if it were executing the method directly on the remote class. The stub method prepares the call and transmits it over the network to the skeleton class in WebLogic Server.

On WebLogic Server, the skeleton class unpacks the request and executes the method on the server-side object. Then it packages the results and returns them to the stub on the client side.

WebLogic EJB and several other services available to Java clients are built on RMI. Most applications should use EJB instead of using RMI directly, because EJB provides a better abstraction for business objects. In addition, the WebLogic Server EJB container provides enhancements such as caching, persistence, and life cycle management that are not automatically available to remote classes.

RMI-IIOP

Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) is a protocol that allows CORBA client programs to execute WebLogic RMI objects, including enterprise beans. RMI-IIOP is based on two specifications from the Object Management Group (`http://www.omg.com`):

- Java-to-IDL mapping

- Objects-by-value

The Java-to-IDL specification defines how an IDL is derived from a Java interface. The WebLogic Server compilers for RMI and EJB give you the option of producing IDL when compiling RMI and EJB objects. This IDL can be compiled with an IDL compiler to produce the stubs required by a CORBA client.

The objects-by-value specification defines how complex data types are mapped between Java and CORBA. To use objects-by-value, a CORBA client must use an Object Request Broker (ORB) with CORBA 2.3 support. Without a CORBA 2.3 ORB, CORBA clients can use only Java primitive data types.

SSL

Data exchanged with the HTTP and T3 protocols can be encrypted with the Secure Sockets Layer (SSL) protocol. Using SSL assures the client that it has connected with an authenticated server and that data transmitted over the network is private.

SSL uses public key encryption, which requires that you purchase a Server ID, which is a certificate for your WebLogic Server, from a Certificate Authority such as VeriSign. When a client connects to the WebLogic Server SSL port, the server and client execute a protocol that includes authenticating the server's Server ID and negotiating encryption algorithms and parameters for the session. WebLogic Server can also be configured to require the client to present a certificate, an arrangement that is called *mutual authentication*.

Data and Access Services

WebLogic Server implements standard J2EE technologies to provide data and access services to applications and components. These services include the following APIs:

- Java Naming and Directory Interface (JNDI)

- Java Database Connectivity (JDBC)

- Java Transaction API (JTA)

The following sections discuss these services in detail.

JNDI

The Java Naming and Directory Interface (JNDI) is a standard Java API that enables applications to look up an object by name. WebLogic Server binds the Java objects it serves to a name in a naming tree. An application can look up objects, such as RMI objects, Enterprise JavaBeans, JMS Queues and Topics, and JDBC DataSources, by getting a JNDI context from WebLogic Server and then calling the JNDI lookup method with the name of the object. The lookup returns a reference to the WebLogic Server object.

WebLogic JNDI supports WebLogic Cluster load balancing and failover. Each WebLogic Server in a cluster publishes the objects it serves in a replicated cluster-wide naming tree. An application can get an initial JNDI context from any WebLogic Server

in the cluster, perform a lookup, and receive an object reference from any WebLogic Server in the cluster that serves the object. A configurable load-balancing algorithm is used to spread the workload among the servers in the cluster.

JDBC

Java Database Connectivity (JDBC) provides access to backend database resources. Java applications access JDBC using a JDBC driver, which is a database vendor-specific interface for a database server. Although any Java application can load a vendor's JDBC driver, connect to the database, and perform database operations, WebLogic Server provides a significant performance advantage by offering JDBC connection pools.

A JDBC connection pool is a named group of JDBC connections managed through WebLogic Server. At startup time WebLogic Server opens JDBC connections and adds them to the pool. When an application requires a JDBC connection, it gets a connection from the pool, uses it, and then returns it to the pool for use by for other applications. Establishing a database connection is often a time-consuming, resource-intensive operation, so a connection pool, which limits the number of connection operations, improves performance.

To register a connection pool in the JNDI naming tree, define a DataSource object for it. Java client applications can then get a connection from the pool by performing a JNDI lookup on the DataSource name.

Server-side Java classes use the WebLogic JDBC pool driver, which is a generic JDBC driver that calls through to the vendor-specific JDBC driver. This mechanism makes application code more portable, even if you change the brand of database used in the backend tier.

The client-side JDBC driver is the WebLogic JDBC/RMI driver, which is an RMI interface to the pool driver. Use this driver the same way you use any standard JDBC driver. When the JDBC/RMI driver is used, Java programs can access JDBC in a manner consistent with other WebLogic Server distributed objects, and they can keep database data structures in the middle tier.

WebLogic EJB and WebLogic JMS rely on connections from a JDBC connection pool to load and save persistent objects. By using EJB and JMS, you can often get a more useful abstraction than you can get by using JDBC directly in an application. For example, using an enterprise bean to represent a dataful object allows you to change

the underlying store later without modifying JDBC code. If you use persistent JMS messages instead of coding database operations with JDBC, it will be easier to adapt your application to a third-party messaging system later.

JTA

The Java Transaction API (JTA) is the standard interface for managing transactions in Java applications. By using transactions, you can protect the integrity of the data in your databases and manage access to that data by concurrent applications or application instances. Once a transaction begins, all transactional operations must commit successfully or all of them must be rolled back.

WebLogic Server supports transactions that include EJB, JMS, and JDBC operations. Distributed transactions, coordinated with two-phase commit, can span multiple databases that are accessed with XA-compliant JDBC drivers, such as BEA WebLogic jDriver for Oracle/XA.

The EJB specification defines bean-managed and container-managed transactions.When an enterprise bean is deployed with container-managed transactions, WebLogic Server coordinates the transaction automatically. If an enterprise bean is deployed with bean-managed transactions, the EJB programmer must provide transaction code.

Application code based on the JMS or JDBC API can initiate a transaction, or participate in a transaction started earlier. A single transaction context is associated with the WebLogic Server thread executing an application; all transactional operations performed on the thread participate in the current transaction.

Messaging Technologies

The J2EE messaging technologies provide standard APIs that WebLogic Server applications can use to communicate with one another as well as with non-WebLogic Server applications. The messaging services include the following APIs:

- Java Message Service (JMS)
- JavaMail

The following sections describe these APIs in detail.

JMS

Java Messaging Service (JMS) enables applications to communicate with one another by exchanging messages. A message is a request, report, and/or event that contains the information needed to coordinate communication between different applications. A message provides a level of abstraction, allowing you to separate details about the destination system from the application code.

WebLogic JMS implements two messaging models: point-to-point (PTP) and publish/subscribe (pub/sub). The PTP model allows any number of senders to send messages to a Queue. Each message in the Queue is delivered to a single reader. The pub/sub model allows any number of senders to send messages to a Topic. Each message on the Topic is sent to every reader with a subscription to the Topic. Messages can be delivered to readers synchronously or asynchronously.

JMS messages can be persistent or non-persistent. Persistent messages are stored in a database and are not lost if WebLogic Server restarts. Non-persistent messages are lost if WebLogic Server is restarted. Persistent messages sent to a Topic can be retained until all interested subscribers have received them.

JMS supports several message types that are useful for different types of applications. The message body can contain arbitrary text, byte streams, Java primitive data types, name/value pairs, serializable Java objects, or XML content.

JavaMail

WebLogic Server includes the Sun JavaMail reference implementation. JavaMail allows an application to create e-mail messages and send them through an SMTP server on the network.

2 WebLogic Server Services

The following sections describe WebLogic Server services:

- WebLogic Server as a Web Server

- Security Services

- WebLogic Clusters

- Server Management and Monitoring

WebLogic Server as a Web Server

WebLogic Server can be used as the primary Web server for advanced Web-enabled applications. A J2EE Web Application is a collection of HTML or XML pages, JavaServer Pages, servlets, Java classes, applets, images, multimedia files, and other types of files.

How WebLogic Server Functions as a Web Server

A Web Application runs in the Web container of a Web server. In a WebLogic Server environment, a Web server is a logical entity, deployed on one or more WebLogic Servers in a cluster.

The files in a Web Application are stored in a directory structure that, optionally, can be packed into a single .war (Web ARchive) file using the Java jar utility. A set of XML deployment descriptors define the components and run-time parameters of an application, such as security settings. Deployment descriptors make it possible to change run-time behaviors without changing the contents of Web Application components, and they make it easy to deploy the same application on multiple Web servers.

Web Server Features

When used as a Web server, WebLogic Server supports the following functionality:

- Virtual hosting

- Support for proxy server configurations

- Load balancing

- Failover

This section describes how each of these functions is supported by WebLogic Server.

Virtual Hosting

WebLogic Server supports virtual hosting, an arrangement that allows a single WebLogic Server or WebLogic Cluster to host multiple Web sites. Each logical Web server has its own host name, but all Web servers are mapped in DNS to the same cluster IP address. When a client sends an HTTP request to the cluster address, a WebLogic Server is selected to serve the request. The Web server name is extracted from the HTTP request headers and is maintained on subsequent exchanges with the client so that the virtual hostname remains constant from the client's perspective.

Multiple Web applications can be deployed on a WebLogic Server, and each Web application can be mapped to a virtual host.

Using Proxy Server Configurations

WebLogic Server can be integrated with existing Web servers. Requests can be proxied from a WebLogic Server to another Web server or, using a native plug-in supplied with WebLogic Server, from another Web server to WebLogic Server. BEA supplies plug-ins for Apache Web Server, Netscape Enterprise Server, and Microsoft Internet Information Server.

The use of proxy Web servers between clients and a set of independent WebLogic Servers or a WebLogic Cluster makes it possible to perform load balancing and failover for Web requests. To the client, there appears to be only one Web server.

Load Balancing

You can set up multiple WebLogic Servers behind a proxy server to accommodate large volumes of requests. The proxy server performs load-balancing, distributing requests across the multiple servers in the tier behind it.

The proxy server can be a WebLogic Server, or it can be an Apache, Netscape, or Microsoft Web server. WebLogic Server includes native code plug-ins for some platforms that allow these third-party Web servers to proxy requests to WebLogic Server.

The proxy server is set up to redirect certain types of requests to the servers behind it. For example, a common arrangement is to configure the proxy server to handle requests for static HTML pages and redirect requests for servlets and JavaServer Pages to a WebLogic Cluster behind the proxy.

Failover

When a Web client starts a servlet session, the proxy server may send subsequent requests that are part of the same session to a different WebLogic Server. WebLogic Server provides session replication to ensure that a client's session state remains available.

There are two types of session replication:

- *JDBC session replication* can be used with a WebLogic Cluster or with a set of independent WebLogic Servers. It does not require the WebLogic Server Clustering option.

- *In-memory session replication* requires the WebLogic Server Clustering option.

JDBC session replication writes session data to a database. Once a session has been started, any WebLogic Server the proxy server selects can continue the session by retrieving the session data from the database.

When a WebLogic Cluster is deployed behind a proxy server, servlet sessions can be replicated over the network to a secondary WebLogic Server selected by the cluster, avoiding the need to access a database. In-memory replication uses fewer resources and is much faster than JDBC session replication, so it is the best way to provide failover for servlets when you have a WebLogic Cluster.

Security Services

WebLogic Server provides security for applications through a service called a security realm. A security realm provides access to two services:

- An *authentication* service, which allows WebLogic Server to verify the identity of users

- An *authorization* service, which controls users' access to applications

Authentication

A realm has access to a store of users and groups and can authenticate a user by testing a user-supplied credential (usually a password) against the username and credential in the security store. Web browsers support authentication by requesting a username and password when a Web client tries to access a protected WebLogic Server service. Other WebLogic Server clients supply usernames and credentials programmatically when they establish WebLogic Server connections.

Authorization

WebLogic Server services are protected with Access Control Lists (ACLs). An ACL is a list of users and groups who are authorized to access a service. Once it has authenticated a user, WebLogic Server checks the ACL for a service before allowing the user to access the service.

Alternative and Custom Realms

The security realm is pluggable—you can use the default File realm, an alternative realm supplied with WebLogic Server, or you can create your own realm. The default File realm stores users, passwords, groups, and ACLs in an encrypted file. The file is managed through the Administration Console.

WebLogic Server includes alternative realms that access an external store of users and groups and sometimes ACLs. Alternative realms are provided for these external security systems:

- Lightweight Directory Access Protocol (LDAP) service, such as Netscape Directory Server, Microsoft Site Server, and Novell NDS

- UNIX login service

- Windows NT domain

- RDBMS, a realm that uses a database system to store users, groups, and ACLs

A custom realm can be developed for use in WebLogic Server by implementing the WebLogic Realm interfaces. WebLogic Server supports realms with a built-in caching system, to minimize calls into the external store. Any features that are not implemented in a custom realm fall back on the default File realm. For example, it is common to develop a custom realm that uses an external store for users and groups, but to maintain ACLs in the File realm.

Encryption

WebLogic Server supports the Secure Sockets Layer (SSL) protocol, which protects data transferred over a wire. SSL is the standard protocol for secure Web connections. WebLogic Server supports SSL on Web (HTTPS) connections and in stand-alone Java clients that use RMI-based services.

To provide SSL, you must first purchase a Server ID for your WebLogic Server from a certificate authority, such as VeriSign or GTE Cybertrust. When a client establishes a secure connection, WebLogic Server sends its certificate to the client, allowing the client to verify that the connection is proper. Clients examine the certificate to make sure that it has not expired, matches the server that sent it, and is signed by a recognized certificate authority. The server and client then exchange encryption keys and establish a secure connection.

WebLogic Server can also be configured to require mutual authentication, which requires a client to submit a client certificate that the server must validate before accepting a connection.

WebLogic Clusters

A WebLogic Cluster is a group of WebLogic Servers that work together to provide a powerful and reliable Web Application platform. A cluster appears to its clients as a single server but it is, in fact, a group of servers acting as one. It provides two key benefits that are not provided by a single server: scalability and availability.

Benefits of Using Clusters

WebLogic Clusters bring scalability and high-availability to J2EE applications in a way that is transparent to application developers. The benefit of scalability is that it expands the capacity of the middle tier beyond that of a single WebLogic Server or a single computer. The only limitation on cluster membership is that all WebLogic Servers must be able to communicate via IP multicast. New WebLogic Servers can be added to a cluster dynamically to increase capacity.

A WebLogic Cluster also guarantees high availability by using the redundancy of multiple servers to insulate clients from failures. The same service can be provided on multiple servers in a cluster. If one server fails, another can take over. The ability to have a functioning server take over from a failed server increases the availability of the application to clients.

Cluster Architecture

A WebLogic Cluster consists of a number of WebLogic Servers deployed on a network, coordinated with a combination of Domain Name Service (DNS), JNDI naming tree replication, session data replication, and WebLogic RMI.

Web proxy servers between Web clients and the WebLogic Cluster coordinate clustering services for servlets and JavaServer Pages. Web proxy servers can be WebLogic Servers, or third-party Web servers from Netscape, Microsoft, or Apache, used with a plug-in supplied with WebLogic Server.

Web clients connect with a WebLogic Cluster by directing requests to a proxy server. Java RMI-based clients connect with a WebLogic Cluster using a cluster address defined on the network.

Server-side code also benefits from the load-balancing and failover services provided by a WebLogic Cluster. In J2EE applications, most application code runs in the middle tier and can use services distributed among several WebLogic Servers. For example, a servlet running on WebLogic Server *A* could use an enterprise bean on WebLogic Server *B* and read messages from a JMS Queue on WebLogic Server *C*.

How a WebLogic Cluster Is Defined in a Network

WebLogic Server services are accessed through DNS, the standard naming service for resources on a network, including the Internet. DNS maps IP addresses, such as *170.0.20.1*, to names, such as *mycomputer.mydomain.com* or *www.bea.com*. Each WebLogic Server runs on the network at a unique IP address. A client connects to a WebLogic Server by encoding in a URL its name and the number of the port where it is listening for connections.

For example, a WebLogic Server running on a computer named *onyx*, configured to listen on port 7701, can be accessed with a Web browser using the following URL: *http://onyx:7701*. For this connection to succeed, the name server on the network must be able to resolve the name *onyx* in the local domain. If the destination server is in another domain on the Internet, the full domain name, for example *http://onyx.bea.com:7701*, must be supplied.

An additional DNS entry maps the names of all the WebLogic Servers participating in a cluster to a single cluster name. Clients connect to the cluster using the cluster name or through a Web proxy server that directs requests into the cluster. When DNS performs a lookup on a cluster name, it returns a list of all the servers that belong to the cluster. A client usually selects the first server in the list and, if it gets no response, tries the second server, working its way through the list until it gets a response.

DNS provides the initial load-balancing service that distributes requests across the servers in the cluster. Each DNS responds to a lookup on the cluster name, by rotating the list of servers by one, so that eventually each server gets a turn.

An intelligent router, proxy server, firewall, or other software operating on the network may override DNS and select the initial server based on machine load, network traffic, or other dynamic load-balancing criteria.

The initial WebLogic Server connection provides the naming service for the client. It looks up the service requested by the client and chooses a server from the cluster to handle the request, using a load-balancing algorithm configured in WebLogic Server.

How WebLogic Servers in a Cluster Communicate

WebLogic Servers in a cluster communicate with each other using IP multicast to replicate certain classes of information to all servers in the cluster. A common multicast address is configured for each server in the cluster. When one server sends a message to the cluster's multicast address, all servers receive the message. This process is much more efficient than having servers send point-to-point messages. However, it does require all the servers in a cluster to be on a network with multicast support. Multicast does not work on the Internet, so a cluster cannot traverse the Internet.

For some services, the cluster selects primary and secondary WebLogic Servers. If the primary WebLogic Server starts processing a request and then becomes unavailable, the secondary server can take over processing of the request without interruption. The primary server replicates state to the secondary server using a server-to-server connection.

Most services can be deployed on any number of WebLogic Servers in a cluster. As each service is deployed, the WebLogic Server uses IP multicast to add the service to a cluster-wide naming tree. Any server in the cluster can find a WebLogic Server to provide a given service by looking up the service in the cluster-wide naming tree. When more than one server can provide a service, the cluster uses a configurable load-balancing algorithm to choose a server.

Clustered Services

Most WebLogic Server services can be clustered; that is, they can be deployed on an unlimited number of servers in the cluster. The cluster selects the WebLogic Server that will provide a service. Once that server has been selected and stateful objects have been instantiated on the server, the client is *pinned* to that WebLogic Server until it has finished with the service. If a WebLogic Server hosting a pinned object fails, the client must detect the failure and create another instance on another server in the cluster.

To provide more resilient failover, a WebLogic Cluster avoids pinning an object to a server unless absolutely necessary. In some cases the cluster replicates the stateful object to a backup server to enable failover for the service.

Web applications can be clustered, as described in the section "WebLogic Server as a Web Server" on page 2-1. Servlet sessions are replicated to a secondary server, allowing the cluster to recover from a failure transparently.

All Enterprise JavaBeans can be clustered. They can be deployed on an unlimited number of servers in a WebLogic Cluster. However, not all EJB instances can be clustered. An application can get the home interface for an EJB from any server where the bean has been deployed, and it can use that home interface to create bean instances. If the server that provides the home interface fails, a home interface can be retrieved from another server without interrupting the application.

Some types of EJB instances, including stateless session beans and read-only entity beans, can always be clustered. Stateful session beans can be clustered using in-memory replication to provide failover. Read-write entity beans are always pinned to the server where they are instantiated. If the server hosting a read-write entity bean fails, the application must create a new instance.

A JDBC metapool provides clustering for JDBC connection pools deployed on multiple servers in a WebLogic Cluster. When a client requests a connection from the metapool, the cluster selects the server that will provide the connection, allowing load-balancing and protection against server failure. Once a client has a connection, the state maintained by the JDBC driver makes it necessary to pin the client to the host WebLogic Server.

JMS objects can be distributed among the servers in a cluster. Each destination (message Queue or Topic) is managed by a single WebLogic Server in the cluster. But connection factories, which clients use to establish a connection to a destination, can be deployed on multiple servers in a cluster. By distributing destinations and connection factories throughout a cluster, administrators can manually balance the load for JMS services.

Server Management and Monitoring

WebLogic Server administration is accomplished by setting attributes for the servers in a domain, using either the Administration Console or the command-line interface. The Administration Console is a Web browser application that allows you to configure WebLogic Server services, manage security, deploy applications, and monitor services dynamically.

Both the Administration Console and the command-line interface connect to the Administration Server.

Administration Server

The Administration Server is the WebLogic Server used to configure and manage all the WebLogic Servers in its domain. A domain may include multiple WebLogic Clusters and independent WebLogic Servers. If a domain contains only one WebLogic

Server, then that server is the Administration Server. In a domain with multiple instances of WebLogic Server, the first instance to start must be the Administration Server.

Administration Console

The WebLogic Server Administration Console runs in a Web browser. It displays the components of the domain it administers, including clusters and independent WebLogic Servers, in a graphical tree in the left pane. The right pane displays details about the object selected in the left pane. Figure 2-1 is a sample snapshot from an Administration Console session.

Figure 2-1   Administration Console
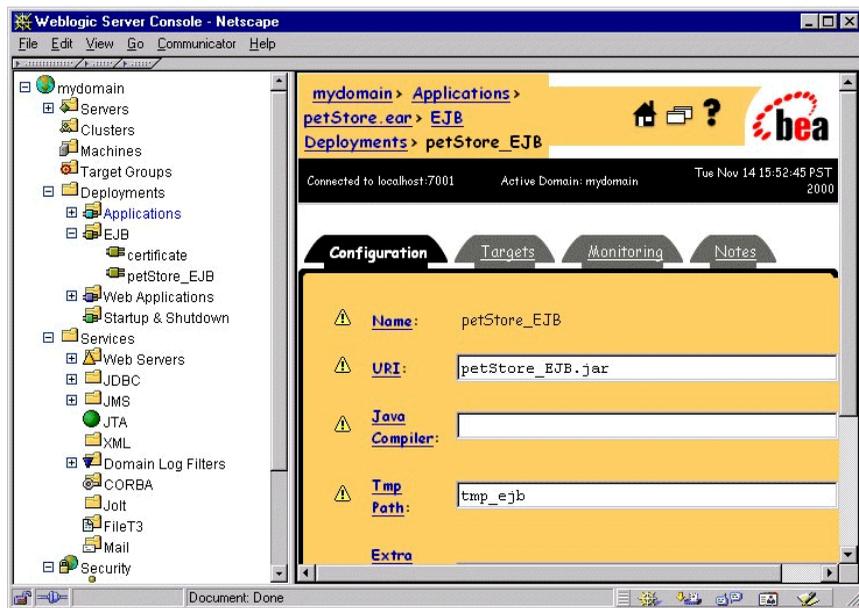
To use the Administration Console to configure a service, select an item in the left pane, and then choose the Configuration tab in the right pane. The Administration Console displays the configurable attributes in the right pane. You can use the online help to find detailed information about the displayed attributes.

The usual process for configuring a service in the Administration Console is to configure the service and then select the *targets* (WebLogic Servers) to which you want to deploy the service.

Each deployed service keeps run-time statistics, which you can view in the Monitoring tab in the right pane of the Administration Console.

Index

publish/subscribe (pub/sub) messaging 1-20

R

RDBMS security realm 2-5
realm 2-5
remote class, RMI 1-16
Remote Method Invocation (RMI) 1-10
    overview 1-15
RMI-IIOP protocol 1-16
router 2-8

S

scalability 2-6
Secure Sockets Layer (SSL) 1-17, 2-6
security realm 2-5
security services 2-4
Server ID 1-17, 2-6
servlets 1-10
session replication 2-3
skeleton class, RMI 1-16
software components 1-4
stub class 1-15, 1-16
Sun Microsystems 1-2
support
    technical vi
Swing 1-10

T

transactions, JTA 1-19
    with EJB 1-19

U

Uniform Resource Identifier (URI) 1-13
UNIX security realm 2-5
user interface
    Swing 1-10
    Web browser 1-10

V

VeriSign 1-17, 2-6
virtual hosting 2-2

W

Web
    application 2-2
    container 1-8
    URIs and URLs 1-13
Web ARchive file 2-2
Web browser clients 1-9
Web container 2-1
Web server 1-6, 2-1
    features 2-2
WebLogic EJB
    relationship to RMI 1-16
WebLogic Enterprise Connectivity 1-7
WebLogic JDBC/RMI driver 1-18
Windows security realm 2-5
Wireless Access Protocol (WAP) 1-6