Chapter 1

An Introduction to J2EE

The Java2 Enterprise Edition (J2EE) is a powerful collection of technologies that sit on top of the Java2 Standard Edition (J2SE) environment. This base of Java2 provides a stable and reliable application environment that runs on many different operating system (OS) platforms. The cross-platform compatibility lifts both application and server environments above the dependencies of specific OS and hardware platforms.

The enterprise technologies that extend the Java2 environment are focused upon providing standard interfaces, which J2EE application server vendors can implement while providing a robust environment for server-based solutions; not only are J2EE enterprise solutions free from OS and hardware dependencies, but also achieve a high degree of portability between application servers. It is precisely this standards-based approach that continues to attract enterprise systems development. Before a picture of J2EE is laid out, let's take a look at the demands of enterprise applications.

Written for *Enterprise Java™ 2, J2EE™ 1.3 Complete*
by Vince E. Marco

Enterprise Applications

Functional requirements of enterprise applications are typically more expansive than many traditional applications. Their functional requirements often include extensive access to a varied array of data sources, and accessibility from a wide array of user interfaces. Enterprise applications also usually have more demanding non-functional requirements including availability, scalability, security, and maintainability. Availability largely refers to an application's capability to process client requests. Enterprises collect and process information using a distributed environment consisting of clients and servers. Enterprise applications include server-based components that desire to provide constant availability to client requests. Application servers restrict the free use of resources (such as threads) to prevent an application from starving a server and preventing clients from accessing needed services.

Scalability refers to the capability of an application to support a wide range of numbers of users. This includes effectively supporting just a handful of users to millions of users by merely changing the deployment environment. In the past, enterprise development was performed on large and expensive mainframes. While this did support large-scale applications, it wasn't effective at deploying small or medium scale applications, and it certainly was costly to maintain a mainframe just for development of the application.

Maintainability involves the management of software complexity. Because Java2 is an object-oriented language, objects are the method for managing this complexity. Objects provide a means for building small software units that interact to provide an entire application. This object technology is extended with components. Components add features such as distributed access, transactions, security, and lifecycle management to objects, and are ideal for server-side objects supporting many clients over a network. As an example, both JavaBeans and Enterprise JavaBeans (EJBs) provide component technology by defining component properties and definition.

J2EE addresses these enterprise application requirements by providing a standard for Java-based application servers. These J2EE application servers are then provided by vendors, each providing the standard J2EE application server functionality, but competing on specific implementation and extended behavior. These servers utilize clustering to provide availability and scalability, enabling enterprise applications to be deployed into a cluster of machines of various sizes.

NOTE

`http://java.sun.com/j2ee` **is the homepage for J2EE and contains links for a plethora of J2EE learning and development resources.**

Java2 Enterprise Edition (J2EE)

J2EE is a collection of several different technologies, each of which helps developers meet the requirements of enterprise applications. Figure 1.1 is a diagram of these technologies.

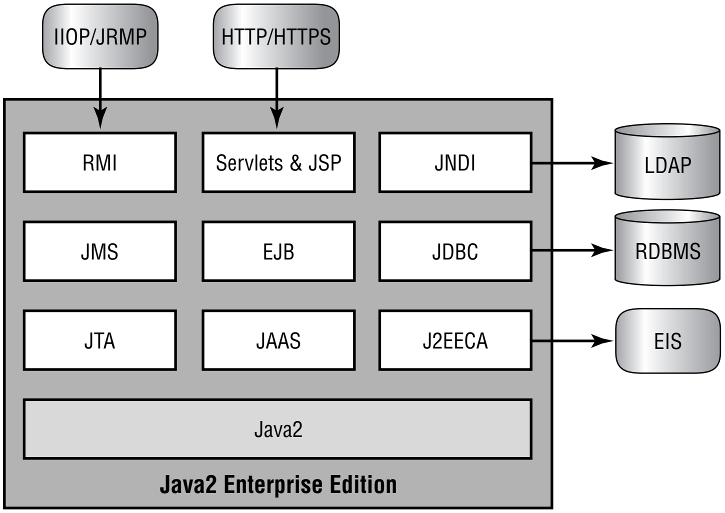

FIGURE 1.1: J2EE technologies

The foundation starts with the standard Java2 environment. This provides a base environment that is cross-platform, and fully supports and leverages the Java2 language and features. A set of technologies is added to this with each technology addressing an enterprise application need. Each technology is composed of an application programming interface (API), and a contract for behavior. The API is composed of a set of Java2 interfaces and classes that define constant values and methods (member functions) for objects not yet implemented. Vendor implementations of these interfaces provide the actual classes that provide J2EE behavior. The behavioral contract is in the form of a specification document, which describes exactly what each technology implementation must do to be compliant with the J2EE specification.

J2EE Technologies

Each J2EE technology focuses on a set of functionality needed by server-based applications. These technologies address the enterprise application demands of availability and scalability through the use of *clustering*.

Clustering is the configuration of multiple servers, each identical in the service provided and enabling client access to be effectively distributed across many servers rather than one large server (see Figure 1.2). This provides effective availability and scalability.

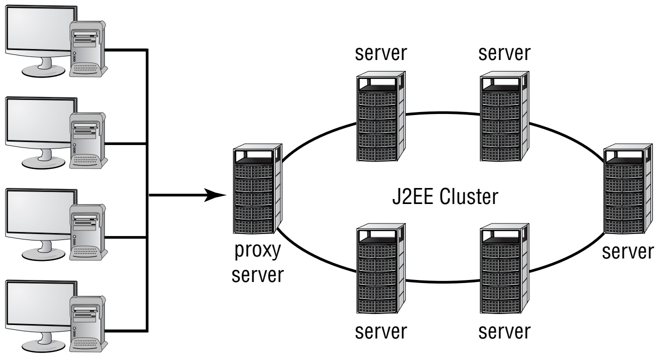

FIGURE 1.2: Clustering of J2EE servers

Applications can be built on one machine as small as a laptop and then deployed to as many servers as needed. Some of these can even be main-frames. Application maintenance is enhanced by using object-oriented and component-based framework development. Also important to J2EE development is multi-tiered development and the separation of tech-nologies. The following describes each technology.

Remote Method Invocation (RMI) RMI provides a way to access distributed Java objects running on a remote server on the network. It can use the Java Remote Method Protocol (JRMP), which supports RMI, or the Internet Inter-Orb Protocol (IIOP), which supports both RMI and CORBA distributed method calls (see Chapter 18, "Persistence and Remote Method Invocation").

Java Database Connectivity (JDBC) JDBC provides a Java-based, standardized API for executing SQL-92 queries and statements on database servers from different vendors (see Chapter 9, “Database Connectivity [JDBC]”).

Java Naming and Directory Interface (JNDI) JNDI provides a uniform and standardized Java-based naming and directory services interface for accessing various types of naming providers such as LDAP and DNS (see Chapter 8, “Java Naming and Directory Interface [JNDI]”).

Java Authorization & Authentication Service (JAAS) JAAS provides a uniform interface for managing enterprise-wide security (see Chapter 22, “EJB Transactions and Security”).

Java Transaction API (JTA) JTA is a standard Java interface for supporting local and distributed transactions (see Chapter 22).

Java Management Extensions (JMX) Java Management Extensions is a framework for managing Java-based services supporting web and external tool integration.

Java Messaging Service (JMS) JMS is an interface that allows standardized access to Message Oriented Middleware servers that can provide asynchronous messaging, through reliable point-to-point and publish-and-subscribe messaging models (see Chapter 17, “Java Messaging Service [JMS]”).

Enterprise JavaBeans (EJBs) A container-based component model that allows for distributed and container managed Java components. A developer creates Java components, called beans, which are placed in an EJB server that provides life-cycle management and services to the bean, such as security, transactions, and object pooling. EJBs can be an excellent technology to create business components that need to be very scalable or support other typical enterprise characteristics (see Chapter 20, “EJB Architecture and Clients”; Chapter 21, “Session, Entity, and Message-Driven EJBs”; Chapter 22; and Chapter 23, “EJB Environment, Client, and Design Issues”).

Servlets and JSPs A model for container managed components that process client requests to a server. They are most commonly used for handling HTTP or HTTPS web requests. Java Server Pages (JSPs) provide an abstraction of servlets that makes it easier to create servlets which predominantly produce dynamic web content such as HTML or XML content. Together servlets and JSPs provide the web interface behavior for J2EE (see Chapter 2, "The Basic Servlet API"; Chapter 3, "The Basic JSP API"; Chapter 4, "Servlet Web Applications"; Chapter 5, "Introducing JavaBeans"; Chapter 6, "Session Management"; and Chapter 7, "Using Custom Tags").

JavaMail and JavaBeans Activation Framework (JAF) An API that enables the sending of email from Java applications. The JavaBeans Activation Framework is used by JavaMail and enables the support of MIME content within email messages.

JavaIDL An API that enables J2EE application components to invoke external CORBA objects via the IIOP protocol. These CORBA objects may be written in a wide variety of languages and run on any CORBA platform (see Chapter 19, "Java IDL and CORBA Connectivity").

J2EE Connector Architecture (J2EECA) This is a standard Java-based architecture for connecting transactional J2EE applications to existing Enterprise Information Systems (see Chapter 24, "J2EE Connector Architecture").

Multi-Tiered Applications

Enterprise applications are characterized by their distributed and scalable nature. These applications encompass both client and server components, and include web applications and services. The explosion of the Internet has led to the need to access and manage large amounts of information within these enterprise applications. Meeting the requirements of modern enterprise application requires not only robust technologies but effective architectural patterns and recommended guidelines for most effectively using these technologies. This has identified the need to separate business logic from the technologies used within these applications. Sun has provided a guideline for building J2EE applications called the J2EE Blueprint. This blueprint organizes enterprise applications into multiple tiers to manage the complexity.

These tiers provide a means of managing the dependencies between the J2EE technologies and reducing the accidental complexity added by the technologies themselves. The base tier includes enterprise information systems and databases. The business model containing the business logic and functionality required by the application sits above the base tier. The user interface sits above the business model, and consists of Java clients such as Java-based applications or applets as well as web clients supported through servlets and JSPs. These tiers separate the aspects of an enterprise application as well as the business logic so that each tier can change at its own rate while minimizing the change to the entire application.

NOTE

The J2EE Blueprints are available online at `http://java.sun.com/j2ee/blueprints`**.**

What's Next

The Java2 Enterprise Edition defines a new standard for enterprise applications and the servers used to deploy them. The J2EE specification provides several technologies for the building and deploying of enterprise applications. Not every enterprise application will use every technology, but these applications now have a standard framework of tools as well as a market of vendors supporting the environment.

This market extends from application servers to frameworks of EJBs to integrated development environments (IDEs) and tools. The J2EE frameworks provide the J2EE developer with many tools for building cross-platform applications for today's informational enterprise.

The following chapters will detail each technology, and present a complete understanding of how they operate and how to use them. As you cover each chapter, the full picture of the Java2 Enterprise Edition will become complete.