Chapter 2. J2EE Overview

Table of Contents

Sun ONE Application Server 7 is compatible with the latest Java[tm] 2 Platform, Enterprise Edition (J2EE) 1.3 specification implementations. This module provides a brief overview of the J2EE environment and the advantages of developing J2EE compliant applications. It highlights J2EE concepts that are a foundation for the rest of the course material.

Objectives

- Provide a high-level overview of the J2EE architecture
- Identify the services and components which comprise the J2EE specification
- Describe how J2EE technology applications are packaged
- Define web services

What is J2EE?

The Java 2 Platform, Enterprise Edition (J2EE) is an architecture for implementing enterprise Java[tm] applications. Two primary features of J2EE are that components run in specific containers, or execution environments, and that specific roles have been defined for the development, deployment, and management of enterprise Java applications. The person who develops a J2EE component may not necessarily be the person who assembles or deploys the application.

The J2EE specification defines guidelines for creating enterprise-scale, distributed applications assembled from building block components. J2EE application components rely on component-specific containers for lifecycle support, transaction control, security, and other services.

J2EE branding assures enterprises and developers that APIs and development features are available for J2EE implementation, and that they work in a uniform way across platforms.

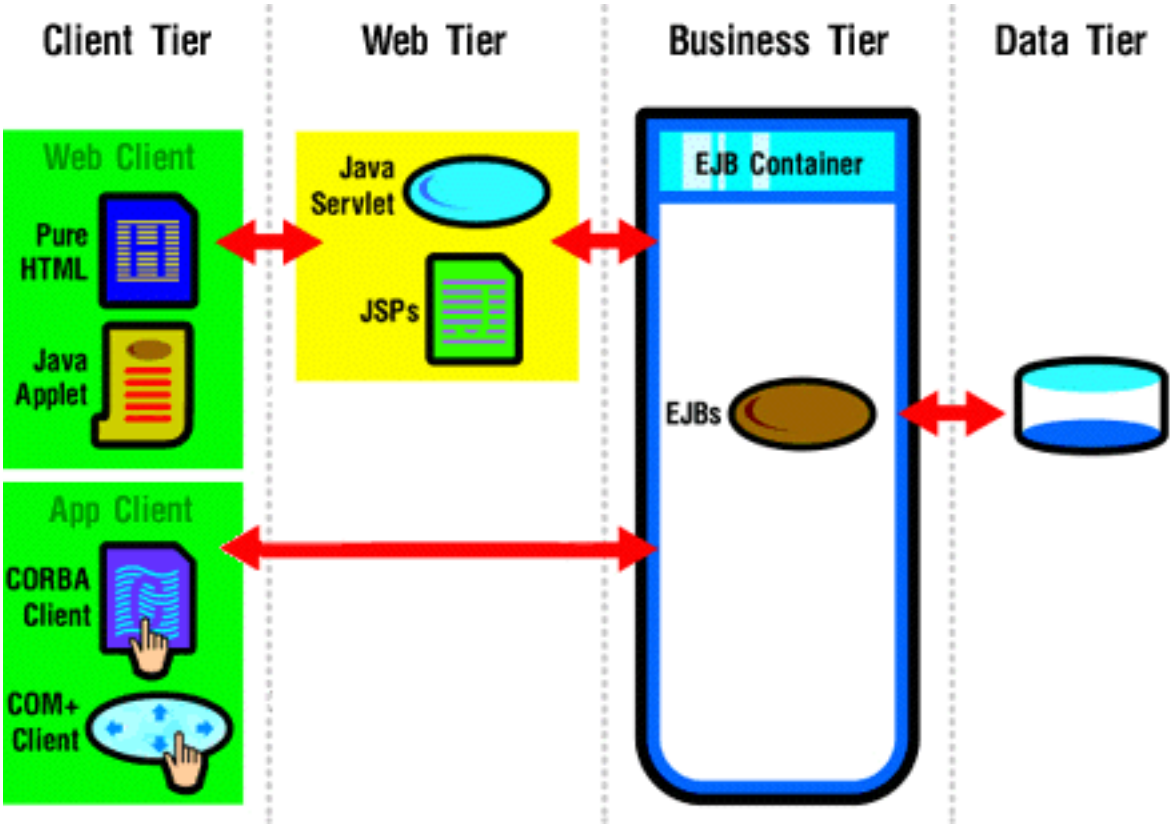
J2EE Architecture

The J2EE architecture can be summarized as follows:

- Multitier application model that gives developers the ability to reuse components for building scalable, distributed, enterprise-class Java applications
- Container-based so that each J2EE component, whether local or remote, runs in a specific execution environment
- Definition of roles that segregate key tasks for application development, deployment, and administration

J2EE Tiers

The J2EE platform uses a multitiered distributed application model. It defines four tiers for components: the client tier, the web tier, the business tier, and the data tier.



Client Tier

The client presentation displays data to a user and also provides a mechanism for users to submit information through the use of form-based input. A J2EE client can be a web client or an application client.

Web Client

A web client consists of two parts: dynamic web pages generated by the web components running in the web tier, and a web browser. A web client is sometimes called a thin client.

The technologies that support this tier include Hypertext Markup Language (HTML), eXtensible Markup Language (XML), and applets.

Application Client

A J2EE application client runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided by a markup language. Application clients directly access enterprise beans running in the business tier.

The technologies that support this tier include JavaBeans technology and Common Object Request Broker Architecture (CORBA), or Component Object Model (COM) clients.

Web Tier

The server-side presentation tier accepts user input from HTML, XML, or applets running on a client. It is also responsible for generating responses based on input from the client-side presentation tier. This tier is implemented with servlets and/or JavaServer Pages[tm] (JSPs) technology. Servlets provide presentation logic. JSPs provide presentation layout and are used for output processing. Servlets and JSP are one of the best Object Oriented replacement for an old fashion CGI access.

Business Logic Tier

The business logic tier is the core of the J2EE architecture because it provides the means to scale and execute business logic in a distributed, enterprise environment. This tier is implemented with Enterprise JavaBeans[tm] (EJBs) technology. EJB technology is commonly used to access backend data sources as part of transactions. These transactions are defined by the business logic and process flow inherent in EJB technology.

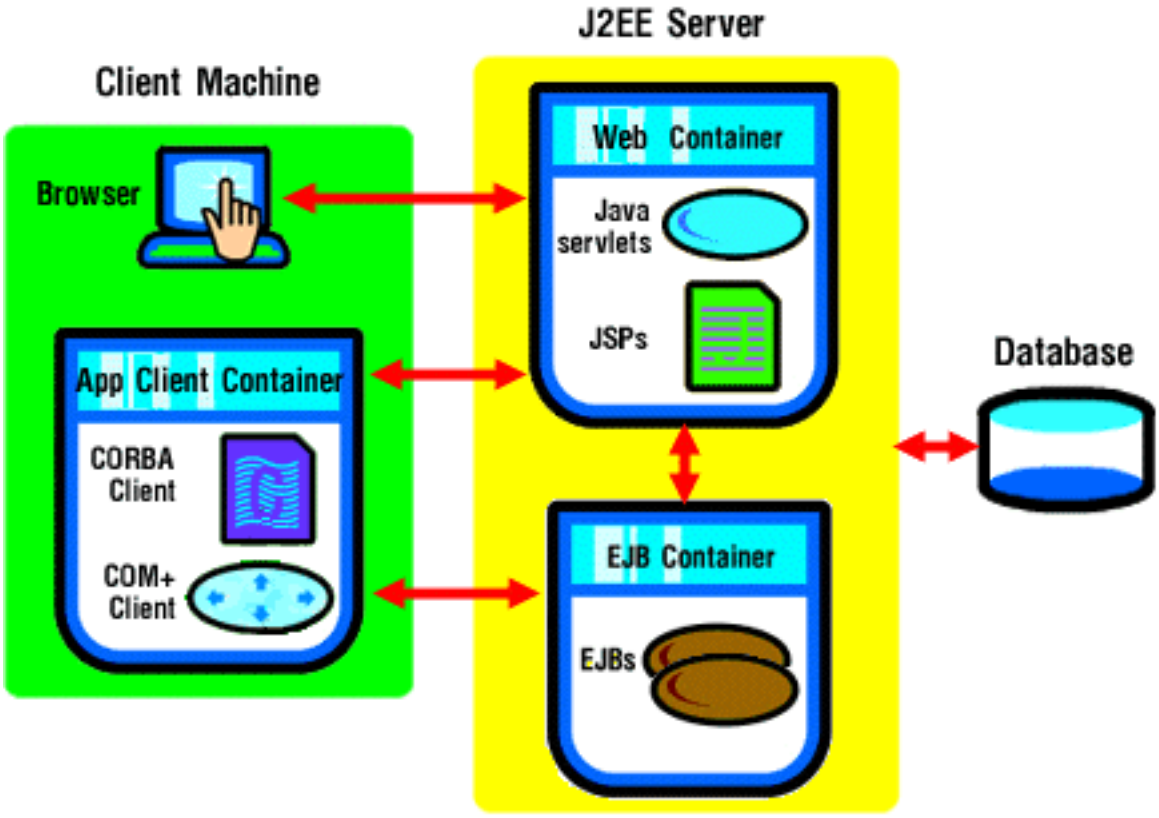
Data Tier (EIS Tier)

The data tier is sometimes referred to as the EIS tier, where EIS stands for Enterprise Information System. This tier provides access to backend data sources. Application components can use the JDBC[tm] API to call relational databases. Enterprise beans can also use EJB technology's transaction management to interact with multiple databases.

J2EE Containers

In addition to being multitiered, the J2EE architecture is also container-based such that specific components execute in known runtime environments defined by the J2EE specification.

Each container can run inside one Java[tm] virtual machine (JVM) or inside its own separate JVM and provides underlying services such as life cycle management, deployment, and threading to support the components executing in it. Containers are the interface between a component and the low-level platform-specific functionality that supports the component. The advantage of this is you do not have to develop these services yourself; the container takes care of it and you are free to concentrate on solving the business problem at hand.



J2EE Client Container

The Client container provides a runtime environment for applets and application client components. This container is in the client tier.

- Application client container: Is responsible for executing all application client components in a J2EE application.
- Applet container: Is responsible for the web browser and Java plug-in combination that runs on the client system.

J2EE Web Container

The web container is a runtime environment for servlets and JSP pages. The web container also provides the ability to layer security into applications on a component-by-component basis. This container is in the web tier.

J2EE EJB Container

The EJB container is a runtime environment for three types of EJBs: entity beans, session beans and message-driven beans. Although EJB components are not the only components that can access datasources, the EJB container provides additional services that facilitate transactions with a database. This container is in the business logic tier.

J2EE Roles

Because J2EE components can be written to function generically, and are portable to any J2EE server, they are often reusable. Therefore, J2EE lends itself to the creation of distributed roles in application development and deploy-

ment. The J2EE specification defines the following roles for the development, deployment, and management of enterprise Java applications:

- Application Component Provider — This role offers a number of subroles, such as HTML, document designers, documents programmers, and enterprise bean developers. It helps develop J2EE applications and reusable components.
- Application Assembler — This role assembles applications from components developed by application component providers. An application may include a static content, represented by HTML pages, graphics, XML documents, etc., and dynamic content represented by Servlets, JSP pages and EJB components.
- Application Deployer — This role deploys web applications and EJB components into a specific operational environment. The application deployer is generally responsible for mapping components built using an object-oriented model to specific elements when the component is deployed.
- System Administrator — This role configures and administrates J2EE applications, computing and networking infrastructure, and the runtime of deployed J2EE applications.
- Tool Provider — This role provides tools for deploying and packaging of J2EE application components. For example, an IDE vendor is an example of a tool provider.
- J2EE Product Provider — This role makes the J2EE APIs available to the application components through containers. It also provides mapping of application components to network protocols and makes available application deployment and management tools.

J2EE Servers and Services

J2EE server is the runtime portion of a J2EE product. J2EE servers comprise both the web container and the EJB container. The web and EJB containers are two major elements of the architecture in Sun Microsystems, Inc.'s reference implementation provided in the J2EE SDK.

J2EE servers must typically provide the following services:

- Naming and directory services — Allows applications to locate services and components using the Java Naming and Directory Interface (J.N.D.I.) API.
- Authentication services — Enforces security by requiring users to authenticate their sessions with a user login.
- HTTP client services — Enables web browsers to access the output of servlets and JSP pages.
- EJB services — Allows clients to invoke methods on EJB components.
- Internet Inter-Orb Protocol (IIOP) services — Enables communication between distributed objects.
- Messaging services — Enables communication between applications and messaging middleware for asynchronous communication between enterprise applications and application components.

Services of EJB Container

EJB components are at the core of the J2EE architecture. In particular, the EJB container, the runtime environment that controls the execution of EJB components, must provide the following services:

- Transaction management — The application developer defines transactional properties in the enterprise bean's

deployment descriptor file. When a client invokes a method in an enterprise bean, the EJB container intervenes. The container reads the descriptor file and handles the enterprise bean's transactions for you. A transaction management system commits writes of data modifications performed within a transaction to a data store and rolls the data back to its initial state if a transaction fails.

- Security — The container permits only authorized clients to invoke an enterprise bean's methods. Each client belongs to a particular role and each role is permitted to invoke certain methods. The roles and the methods they invoke are defined in the enterprise bean's deployment descriptor.

- Remote client connectivity — The container manages the low-level communications between clients and enterprise beans. After an enterprise bean has been created, a client invokes methods on it as if it were in the same virtual machine.

- Life cycle management — An enterprise bean passes through several states during its lifetime. The container creates the enterprise bean instance, moves it between a pool of available instances and the active state, and finally, destroys it. Although the client calls methods to create and remove an enterprise bean, the container performs these tasks behind the scenes.

- Database connection pooling — A database connection is a costly resource. Obtaining a database connection is time-consuming and the number of connections may be limited. To alleviate these problems, the container manages a pool of database connections. An enterprise bean can quickly obtain a connection from the pool. After the bean releases the connection, it may be re-used by another bean.

- Enitity persistence — refers to the creation and maintenance of an enterprise bean throughout the application's lifetime. There are two types of persistence management: container-managed persistence (CMP) and bean-managed persistence (BMP). These are described later in this module.

J2EE Technology

Sun ONE Application Server 7 implements J2EE 1.3 specification.

What's New in J2EE 1.3?

A significant development is the inclusion of the latest versions of the Servlet and JavaServer Page specifications and Enterprise JavaBean (EJB) 2.0 specification.

J2EE 1.3 also saw the inclusion of the new J2EE Connector 1.0. The Java Connector Architecture enables vendors of existing enterprise resource planning (ERP), customer relationship management (CRM) and other legacy enterprise information systems (EIS) to provide a resource adapter for J2EE vendors.

The Java API for XML Processing (JAXP) 1.1 -- previously excluded from J2EE -- has also been incorporated into version 1.3.

Also new to J2EE is the Java Authentication and Authorisation Service (JAAS). The JAAS is an enhancement of the Java 2 security model. It provides a standard way for limiting access to resources based on an authenticated user identity.

The JAAS includes standard APIs for login and logout regardless of whether you use smart cards or any other way to authenticate a user.

Finally J2EE 1.3 mandates the support for Java Messaging Service (JMS); support for JMS was previously optional.

Inside the J2EE 1.3 SDK

- Java Servlets 2.3
- JavaServer Pages (JSP) 1.2
- Enterprise JavaBeans (EJB) 2.0
- JDBC 2.0
- Java Transaction API (JTA) 1.0
- Java Message Service (JMS) 1.0
- JavaMail 1.2
- JavaBeans Activation Framework 1.0
- Java API for XML Processing (JAXP) 1.1
- J2EE Connector Architecture 1.0
- Java Authentication and Authorisation Service (JAAS) 1.0

Servlets 2.3

Java Servlets provide a component-based, platform-independent method for building web-based applications. They are the Java platform technology of choice for extending and enhancing web servers. Servlets are a popular alternative to CGI scripts.

In J2EE, servlets make up the presentation logic of an application by acting as a central dispatcher for applications. Servlets process form input, initiate business logic components by accessing Enterprise JavaBeans, and format page output using JSP pages. Servlets control the application's flow from one user interaction to the next by generating content in response to user requests.

What's New in Servlet API 2.3

The Servlet API 2.3 is slated to become a core part of Java 2 Platform, Enterprise Edition 1.3. This version actually leaves the core of previous version, Servlet API 2.2, relatively untouched. The new features added are mostly outside the core. They are:

- Filter mechanism — Filters are a mature version of the old "servlet chaining" concept. They are the objects that can transform a request or modify a response.
- Application lifecycle events — Lifecycle events let "listener" objects be notified when servlet contexts and sessions are initialized and destroyed, as well as when attributes are added or removed from a context or session.
- Internationalization support — Servlet API 2.3 provides much-needed support for handling foreign language form submittals
- Class Loaders — In API 2.3, a servlet container will ensure that classes in a Web application not be allowed to see the server's implementation classes. In other words, the class loaders should be kept separate. This eliminates the possibility of a collision between Web application classes and server classes.

Note

For more information about servlet, see Java Servlet Technology [http://java.sun.com/products/servlet/index.html] from java.sun.com.

JSPs 1.2

JSPs handle most application display tasks, and they work in conjunction with servlets to define the application's presentation screens and page navigation. JSPs are likely to call EJBs and JDBC objects. The EJBs typically encapsulate business logic functionality. JDBC objects are used to connect to databases, make queries, and return query results.

JSP is an extension of the servlet technology created to support authoring of HTML and XML pages. It makes it easier to combine fixed or static template data with dynamic content. Think of servlets and JSP pages as opposite sides of the same coin: each can perform the tasks of the other.

What's New in JSP 1.2

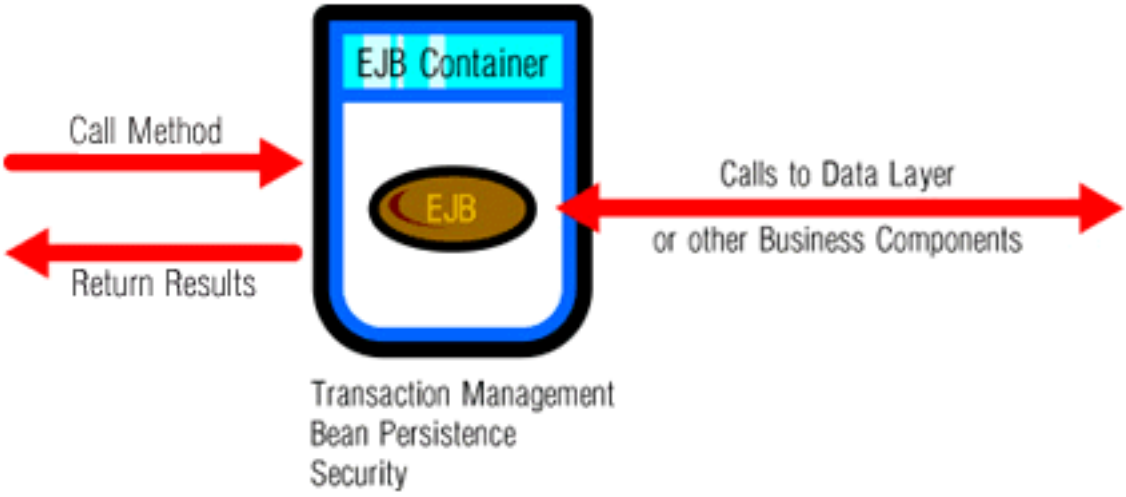
- Using Servlet 2.3 as the foundation for its semantics.
- Defining the XML syntax for JSP pages.
- Providing for translation-time validation of JSP pages.
- Specifying refinements of tag library runtime support.
- Improving character encoding and localization support.

Note

For more information about JSPs, see JavaServer Pages [http://java.sun.com/products/jsp/index.html] from java.sun.com.

EJBs 2.0

Enterprise JavaBeans (EJB) components are the workhorses of applications. If servlets act as central dispatchers for applications and handle presentation logic, EJBs do the bulk of the application's actual data and rules processing. EJB components do not provide presentation or visible user interface services. Each EJB component encapsulates one or more application tasks or application objects, including data structures and the methods that operate on them.



EJB components always work within the context of a container that serves as a link between the EJB components

and the server that hosts them. The API defines three types of beans: entity beans, session beans and message driven beans.

Entity Beans

Entity beans are used to represent business entities, primarily persistent data, in a relational database. When programmed with application logic, the logic should apply only to the entity bean itself, not to other entities, such as other beans.

Session Beans

Session beans are used to provide multiple services for a specific Web client. Unlike entity beans, session beans typically are programmed with complex application logic for server-side transactions. For example, a session bean can access the database on behalf of a client request, but can also coordinate transactions between multiple entity beans.

Session beans can be coded as stateful or stateless. A stateful session bean is identified with a specific client. When a stateful session bean is instantiated, the EJB container maintains a unique EJB object and instance for that bean. In contrast, a stateless session bean does not belong to a specific client. Stateless session beans can be placed in a pool of available instances to be made available to a number of clients.

Message Driven Beans

Message Driven Beans (MDB) are similar to session and entity Beans in that they support the framework provided by an EJB component. However, message-driven beans are also Java[tm] Messaging Service (JMS[tm]) listeners and perform tasks based upon the request it receives from a client in the form of JMS Messages.

Unlike session and entity beans, MDBs process message queues asynchronously, thereby making better use of server resources. The MDB can handle many client requests simultaneously and therefore, does not create message queue bottlenecks.

The most visible difference between MDBs and session and entity beans is that clients do not access MDBs through interfaces. A MDB interfaces with JMS. In other words, a Message Driven Bean is a JMS Listener. MDBs accept two JMS messaging models, publish-and-subscribe and point-to-point. MDB as JMS consumers can either subscribe to JMS topics (for publish-and-subscribe) or listen to JMS queues (for point-to-point).

What's new in EJB 2.0?

EJB 2.0 technology represents a major new release offering many significant benefits, further simplifying and expediting development and deployment of more feature-rich enterprise applications. It is an important step in streamlining the development and deployment of J2EE applications. The key new features of the EJB 2.0 specification are:

- Local interfaces — For session and entity beans, EJB relationships are now based on the local interface.
- Container-managed Persistence (CMP) —Provides a new way of handling container-managed persistence.
- Container-managed Relationships — Allows you to define the relationship between the entity beans in the implementation classes and the deployment descriptors.
- Integration with JavaTM Message Service (JMS) — This new type of EJB is a Java Message Service consumer.
- Additional methods on the home interface — Allow you to implement business logic that is independent of a specific entity bean instance.
- New Query Language (EJB-QL) — The new EJB Query Language (EJB-QA) provides for navigation across a

network of entity beans defined by container-managed relationships.

Note

For more information about EJB 2.0, see Enterprise JavaBeans Technology [http://java.sun.com/products/ejb/index.html] from java.sun.com.

JDBC 2.0

JDBCTM technology is an API that lets you access virtually any tabular data source from the JavaTM programming language. It provides cross-DBMS connectivity to a wide range of SQL databases, and now, with the new JDBC API, it also provides access to other tabular data sources, such as spreadsheets or flat files.

What's New in JDBC 2.0 API?

With the JDBC 2.0 API, you will be able to do the following:

- Scroll forward and backward in a result set or move to a specific row
- Make updates to database tables using methods in the Java programming language instead of using SQL commands
- Send multiple SQL statements to the database as a unit, or batch
- Use the new SQL3 datatypes as column values

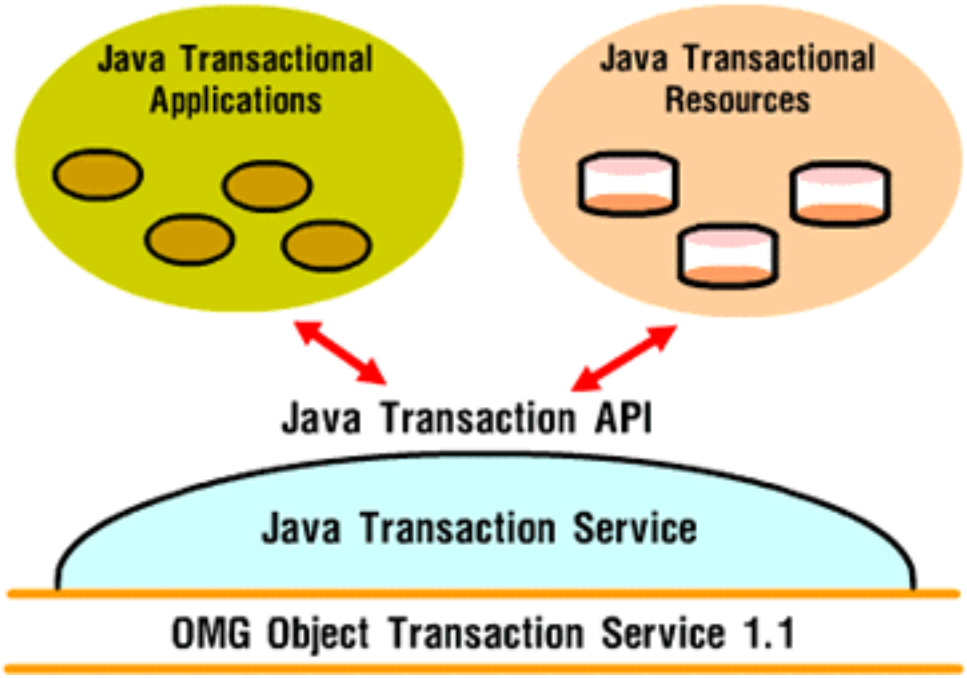
Note

For more information about JDBC API, see JDBC Technology [http://java.sun.com/products/jdbc/index.html] from java.sun.com.

JTA 1.0

J2EE simplifies application programming for distributed transaction management. J2EE includes support for distributed transactions through two specifications: Java Transaction API (JTA) and Java Transaction Service (JTS).

JTA and JTS allow application servers built on J2EE to take the burden of transaction management off of the component developer. Developers can define the transactional properties of EJB technology based components during design or deployment using declarative statements in the deployment descriptor. The application server takes over the transaction management responsibilities.



JTS

Java Transaction Service (JTS) provides specification for building a Transaction Manager that supports the Java Transaction API (JTA). Internally the JTS implements the Java mapping of the Object Management Group (OMG) Object Transaction Service (OTS) 1.1 specification.

JTS uses the standard CORBA ORB/TS interfaces and Internet Inter-ORB Protocol (IIOP) for transaction context propagation between JTS Transaction Managers.

Note

For more information about JTS, see Java Transaction Services [http://java.sun.com/products/jts/] from java.sun.com.

JTA

Java Transaction API (JTA) provides a way for J2EE components and clients to manage their own transactions and for multiple components to participate in a single transaction. The JTA 1.0.1 specifies an architecture for building transactional application servers and defines a set of interfaces for various components of this architecture. The components are: the application, resource managers, and the application server, as shown in the figure above.

Although the JTS is a Java implementation of the OMG OTS 1.1 specification, the JTA retains the simplicity of the XA and TX functional interfaces of the X/Open Distributed Transaction Processing (DTP) model. The Open Group's XA specification defines a model for distributed transaction processing. The JTA is a Java mapping of the XA protocol.

Note

For more information about JTA, see Java Transaction API [http://java.sun.com/products/jta/] from java.sun.com.

JMS 1.0

The Java Message Service (JMS) is an API for accessing enterprise messaging systems. JMS makes it easy to write business applications that asynchronously send and receive critical business data and events.

JMS provides a set of standard Java language interfaces to enterprise messaging systems, often called message-oriented middleware. These interfaces are implemented by products called JMS providers. The JMS API and provider framework enables the development of portable, message-based applications in the Java programming language.

Messaging Models

JMS defines two messaging model: Point-to-Point and Publish/Subscribe.

Point-to-Point

In the point-to-point messaging model, messages are sent to the message broker's queue. Because of the inherent nature of queues, messages follow a first-in-first-out paradigm. This requires that the broker retain subsequent messages until previous messages have been consumed. Messages are read only once through the queue.

Publish/Subscribe

In the publish and subscribe messaging model, each session object publishes and subscribes to topics. To publish messages, a session must create a publisher object for the topic. Only authorized publishers and subscribers can create or receive messages for a topic. To subscribe to a topic, a session must create the necessary subscriber object.

Benefits of JMS

JMS supports connection pooling and user mapping.

- Connection pooling enhances the performance and reliability of applications using JMS through the creation and management of pools of connections to a JMS-enabled messaging product.
- User mapping speeds up application development and eases administration by supporting the easy mapping of users authenticated at the web application level to users, groups, and roles authorized by a JMS-enabled messaging provider.

Note

For more information about JMS API, see Java Message Service [http://java.sun.com/products/jms/] from java.sun.com.

JavaMail 1.2

JavaMail is a set of API for modeling a mail system. It has pre-built implementations of some of the most popular protocols for mail transfer and provides an easy way to use them. The JavaMail API is designed to make adding electronic mail capability to simple applications easy, while supporting the creation of sophisticated common mail functions and protocols. The API defines classes like Message, Store, and Transport.

The current JavaMail implementations actually include providers for some of the most essential protocols and specifications on the Internet today, including the Simple Mail Transfer Protocol (SMTP), the Internet Message Access Protocol (IMAP), the Post Office Protocol (POP3), and the Multipurpose Internet Mail Extension (MIME).

Note

For more information about JavaMail, see JavaMail [http://java.sun.com/products/javamail/] from java.sun.com.

JAF 1.0

The JavaBeans Activation Framework (JAF) allows Java developers to take advantage of standard services to determine the type of an arbitrary piece of data, encapsulate access to it, and discover the operations available on it, then instantiate the appropriate JavaBeans component to perform said operation(s). JAF is implemented as a standard extension.

Using JAF within your application enables application developers to easily extend the application's capabilities when new data types need to be rendered or additional commands need to be implemented on the data. Sun's JavaMail classes utilize JAF for the display of various MIME mail attachments.

Note

For more information about JFA 1.0, see JavaBeans Activation Framework [http://java.sun.com/ products/ javabeans/ glasgow/jaf.html]from java.sun.com.

JAXP 1.1

The JavaTM API for XML Processing (JAXP) supports processing of XML documents using DOM, SAX, and XSLT. JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation. Depending on the needs of the application, developers have the flexibility to swap between XML processors (such as high performance vs. memory conservative parsers) without making application code changes. Thus, application and tools developers can rapidly and easily XML-enable their Java applications for e-commerce, application integration, and dynamic web publishing.

Note

For more information about JAXP, see Java API for XML Processing [http://java.sun.com/xml/jaxp/] from java.sun.com.

JCA 1.0

The J2EE Connector Architecture enables vendors of existing enterprise resource planning (ERP), customer relationship management (CRM) and other legacy enterprise information systems (EIS) to provide a resource adapter for J2EE vendors. With the Connector Architecture, J2EE vendors need not write unique connectivity drivers for each EIS.

JCA defines the functionality that an application vendor must provide to enable other back-end system vendors to plug into J2EE. The JCA consists of Common Client Interface (CCI) that provides an interface for application developers to connect and access back-end systems. CCI is similar to JDBC with a difference that CCI can also work with nonrelational systems. The JCA also consists of a set of system-specific services that define the services to be present in a connector.

Note

For more information about JCA, see J2EE Connector Architecture [http://java.sun.com/j2ee/connector/] from java.sun.com.

JAAS 1.0

The Java[tm] Authentication and Authorization Service (JAAS) is a set of packages that enable services to authenticate and enforce access controls upon users. JAAS Support has been integrated into the J2EE 1.3 platform. The key features are as follows:

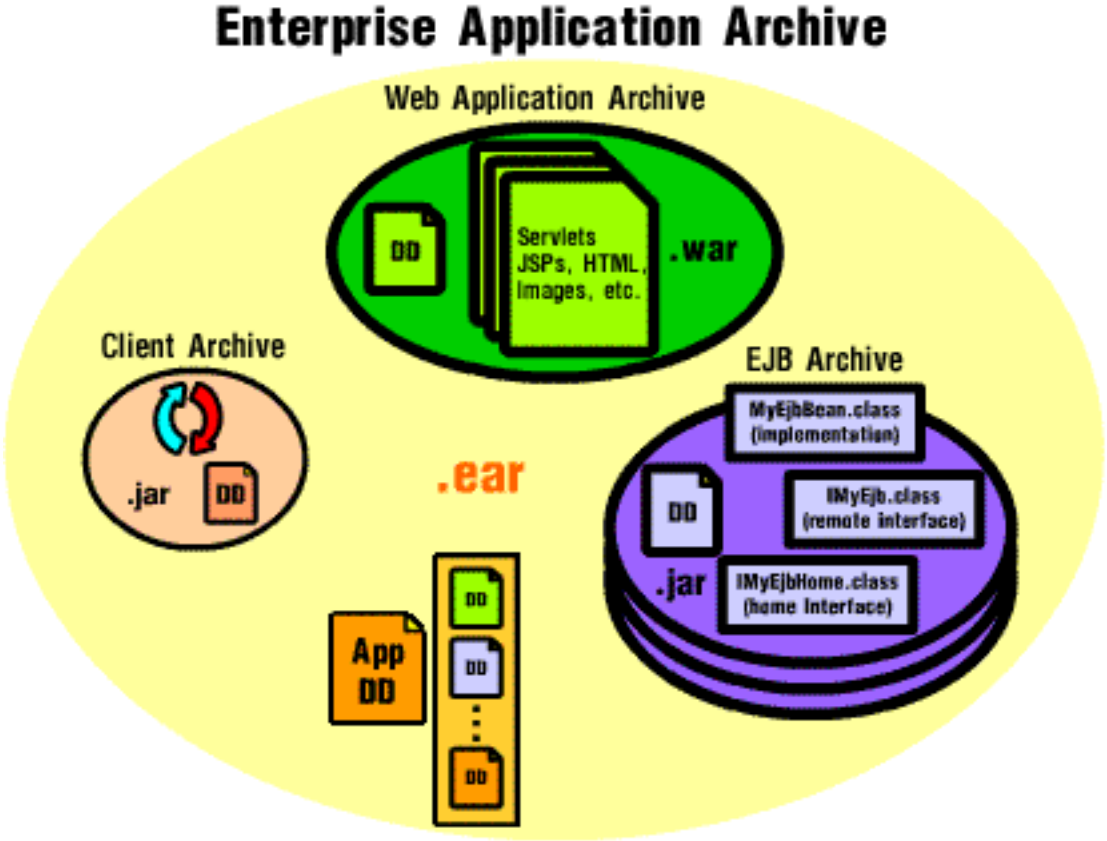
- Plugable Authentication Module (PAM) framework implementation for authenticating users
- Flexible access control policy for user-based, group-based, and role-based authorization.

Note

For more information about JAAS, see Java Authentication and Authorization Service [http://java.sun.com/products/jaas/index.html] from java.sun.com.

Packaging

The way in which J2EE applications are packaged is conducive to multiple developers contributing components for an application. Specialized archive files contain Java classes and XML deployment descriptors specific to the application. As long as the J2EE components are packaged correctly, the origin of the components is irrelevant.



Client Archive File

The client archive file is also a standard java archive file (.JAR). It contains the classes that comprise the client application as well as the application's deployment descriptor (DD) file in XML format.

Web Archive File

The web archive file (.WAR) stores the web-related (web container) components of the J2EE application, such as JSP pages, servlets, HTML pages, and image files. The .WAR file also includes all the Java classes used to define these components, and the web component deployment descriptor (DD) file in XML format, which is used to identify information about the web components.

EJB Archive File

The EJB archive file is a standard java archive file (.JAR). It contains the classes, helper classes, and interfaces, such as the home and remote interfaces, that are required by the EJB component. It also contains the EJB deployment descriptor (DD) file in XML format which is used to identify information for the EJB's deployment.

Enterprise Archive File

The Enterprise archive file (.EAR) stores a J2EE application assembled from multiple EJB, client, or web archive files. It also contains the J2EE application deployment descriptor (DD) file in XML format that provides information about the application for deployment.

A J2EE application may contain zero or more client archive JAR files, EJB JAR files, or WAR files. It must contain at least one module, though.

Web Services

What Is Web Services?

In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response. Web Services are in essence a collection of standards and protocols that allow us to make processing requests to remote systems by speaking a common, non-proprietary language and using common transport protocols (HTTP, SMTP).

Current technologies open up the Web only to humans through HTML links and URLs. Web services, on the other hand, provide a well-defined discovery and communication framework based on XML and HTTP that facilitates interaction between machines; they offer true peer-to-peer distributed computing over the Web. The advent of Web services has made the World Wide Web truly complete; they provide the missing link -- a machine-friendly user interface (UI) to the Web.

Core Enabling Technologies

Several new technologies support Web services, but because they build on pre-existing technologies like XML and HTTP, they're not as new as you might think. Lets briefly look at them:

- SOAP (Simple Object Access Protocol) – Is a protocol specification by which users on different platforms can interact with each other over the internet.
- WSDL (Web Services Description Language) – Is an XML-based language used to describe the services a web

service offers. It helps to provide a way for web services access the services electronically.

- UDDI (Universal Description Discovery and Integration) – Provides a specification that permits the publication and location of services in a universal service registry.
- ebXML – Is designed to enable a global electronic marketplace in which enterprises of any size, and in any location, can safely and securely transact business through the exchange of XML-based messages. ebXML aims to encompass and replace Electronic Data Interchange (EDI).

Characteristics of Web Services

Web services help standardize machine-to-machine interaction to create new service breeds that integrate the Web as one large computer. The characteristics of Web services are:

- A web service is accessed programmatically by applications or other web services
- They provide interfaces that can be called from another program.
- They can be registered and can be located through a web service registry. WSDL, UDDI, and ebXML Reg/Rep are popular standards.
- They communicate messages using standard Internet protocols.
- They support loosely coupled connections between applications or systems.

Advantages of J2EE Applications

All J2EE applications offer the benefits such as scalability, portability, and ease in programming. Some specific advantages include the following:

- Well defined, standardized container specific runtime environment — The application model encapsulates layers of functionality in specific types of components. These components are well defined in standardized containers, each having a specific runtime environment.
- Reusable components — Reusable J2EE components enables enterprise developers to assemble applications from standard, commercially available and customized components.
- Diversification of roles for development and deployment of applications — J2EE offers faster development and deployment of applications as specific functionality can be added by individual functional experts.