Enterprise Computing: J2EE Containers, Packaging, Web Services and the J2EE APIs

Stephen Gilmore

The University of Edinburgh

J2EE Containers

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details.

J2EE Containers

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details.

The component-based and platform-independent J2EE architecture makes J2EE applications easy to write because business logic is organized into reusable components.

In addition, the J2EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.

Container Services

Containers are the interface between a component and the low-level platform-specific functionality that supports the component.

Container Services

Containers are the interface between a component and the low-level platform-specific functionality that supports the component.

Before a Web, enterprise bean, or application client component can be executed, it must be assembled into a J2EE application and deployed into its container.

Container Services

Containers are the interface between a component and the low-level platform-specific functionality that supports the component.

Before a Web, enterprise bean, or application client component can be executed, it must be assembled into a J2EE application and deployed into its container.

The assembly process involves specifying container settings for each component in the J2EE application and for the J2EE application itself.

Container settings

Container settings customize the underlying support provided by the J2EE server, which includes services such as security, transaction management, Java Naming and Directory Interface™ (JNDI) lookups, and remote connectivity.

Container settings

Container settings customize the underlying support provided by the J2EE server, which includes services such as security, transaction management, Java Naming and Directory Interface™ (JNDI) lookups, and remote connectivity.

Here are some of the highlights:

- The J2EE security model lets you configure a Web component or enterprise bean so that system resources are accessed only by authorized users.

- The J2EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.

- The J2EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.

- JNDI lookup services provide a unified interface to multiple naming and directory services in the enterprise so that application components can access naming and directory services.

- The J2EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.

Configurability and deployment

The fact that the J2EE architecture provides configurable services means that application components within the same J2EE application can behave differently based on where they are deployed.

Configurability and deployment

The fact that the J2EE architecture provides configurable services means that application components within the same J2EE application can behave differently based on where they are deployed.

For example, an enterprise bean can have security settings that allow it a certain level of access to database data in one production environment and another level of database access in another production environment.

Other container roles

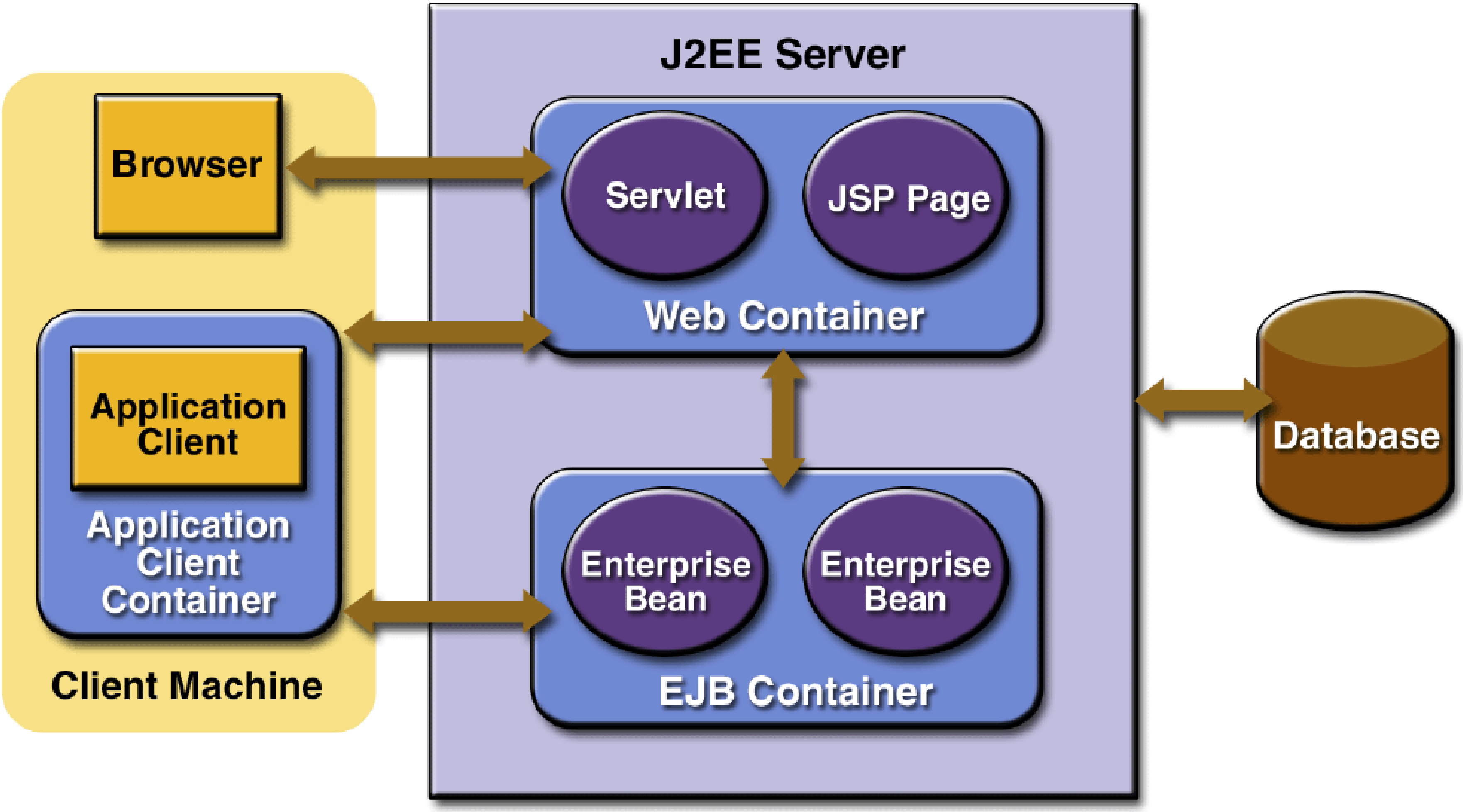
The container also manages non-configurable services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the J2EE platform APIs.

Other container roles

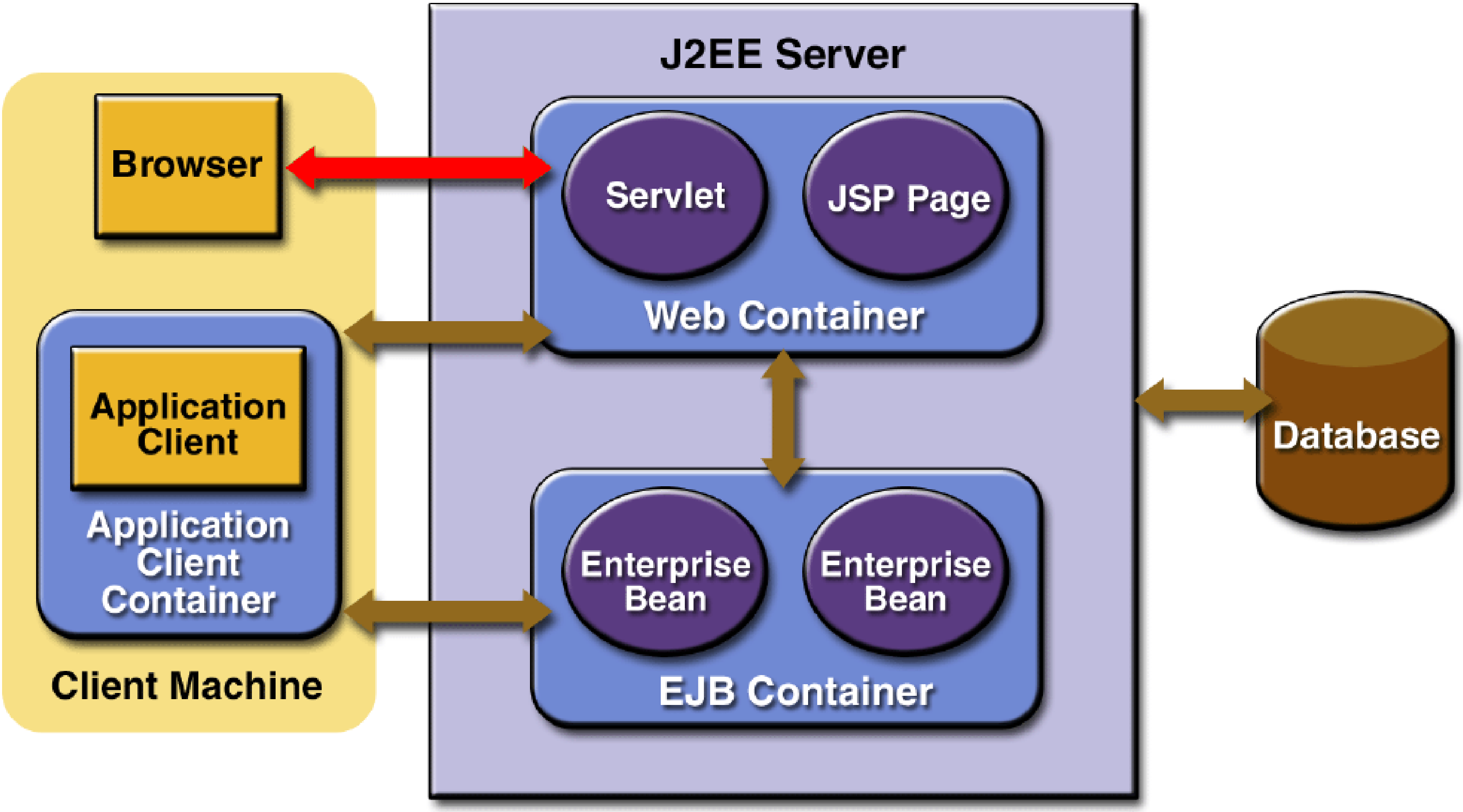
The container also manages non-configurable services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the J2EE platform APIs.

Although data persistence is a non-configurable service, the J2EE architecture lets you override container-managed persistence by including the appropriate code in your enterprise bean implementation when you want more control than the default container-managed persistence provides.

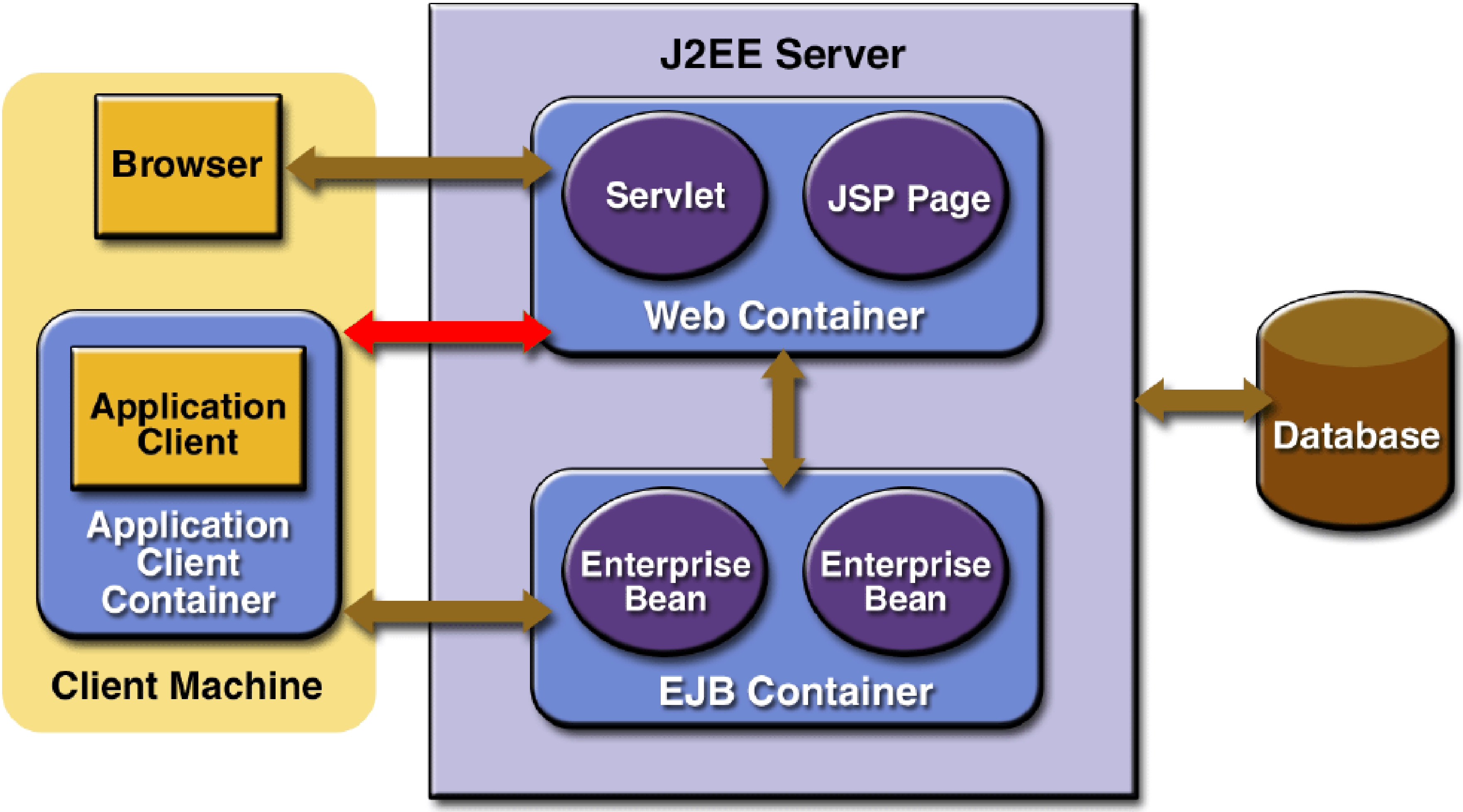
J2EE Server and Containers



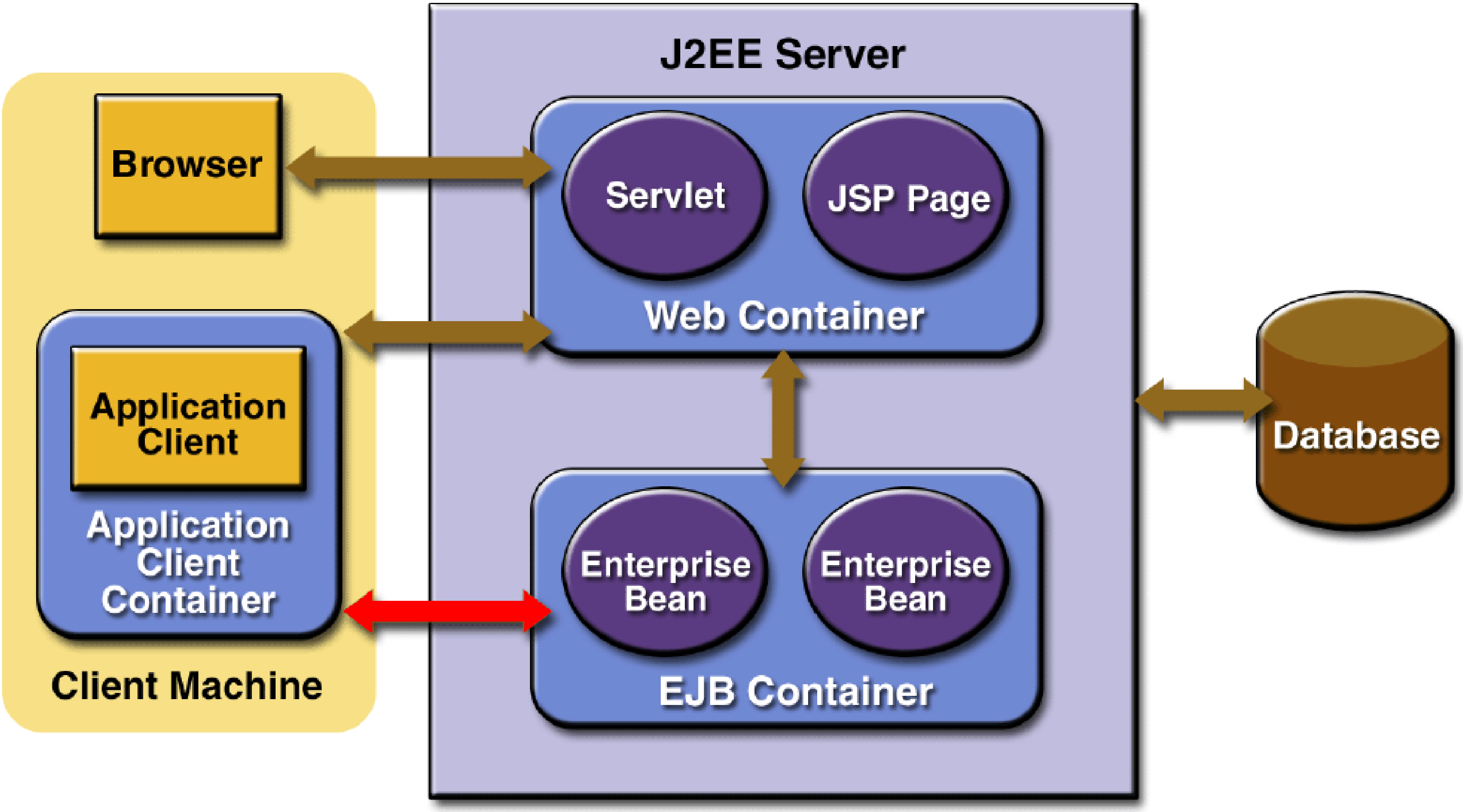
J2EE Server and Containers



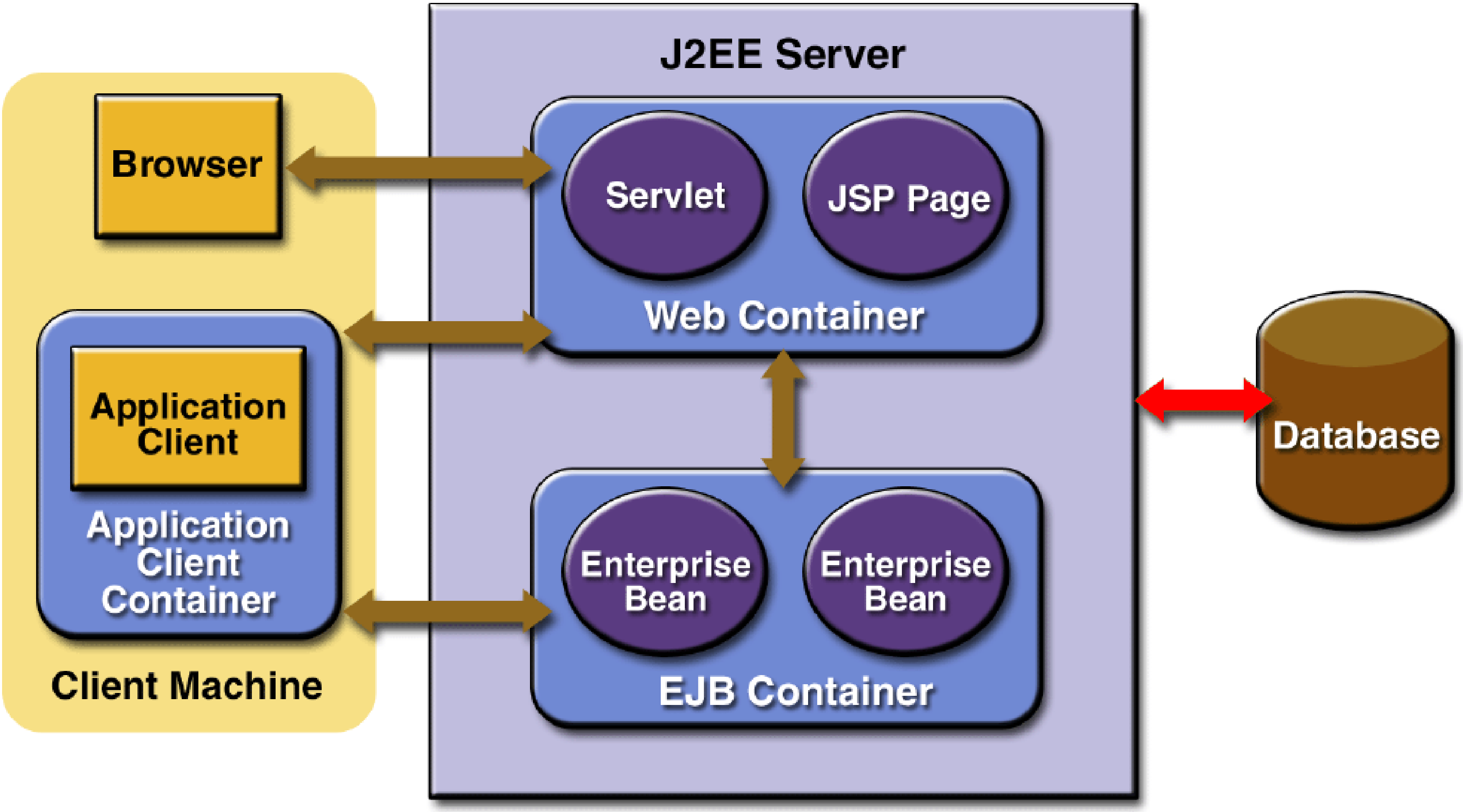
J2EE Server and Containers



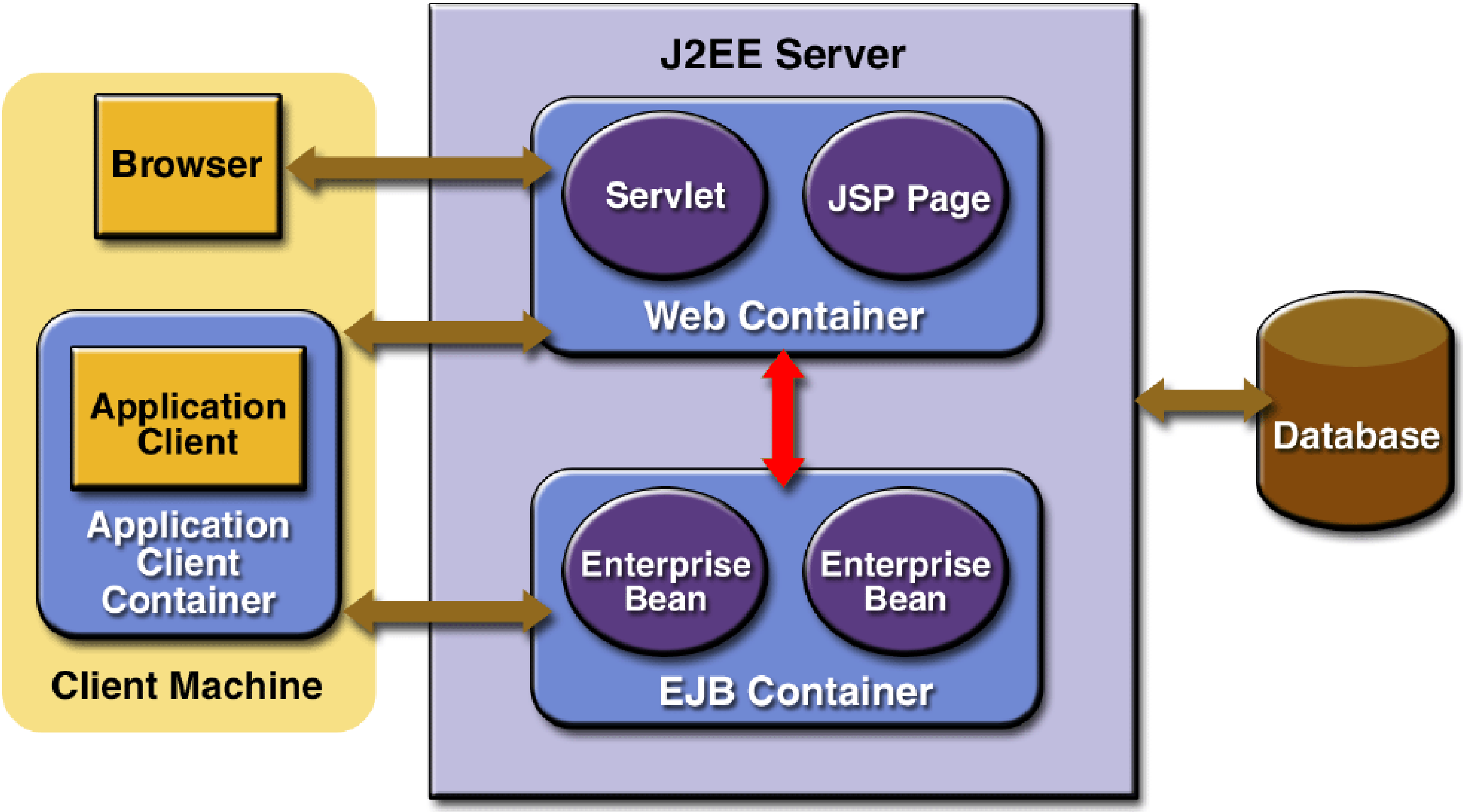
J2EE Server and Containers



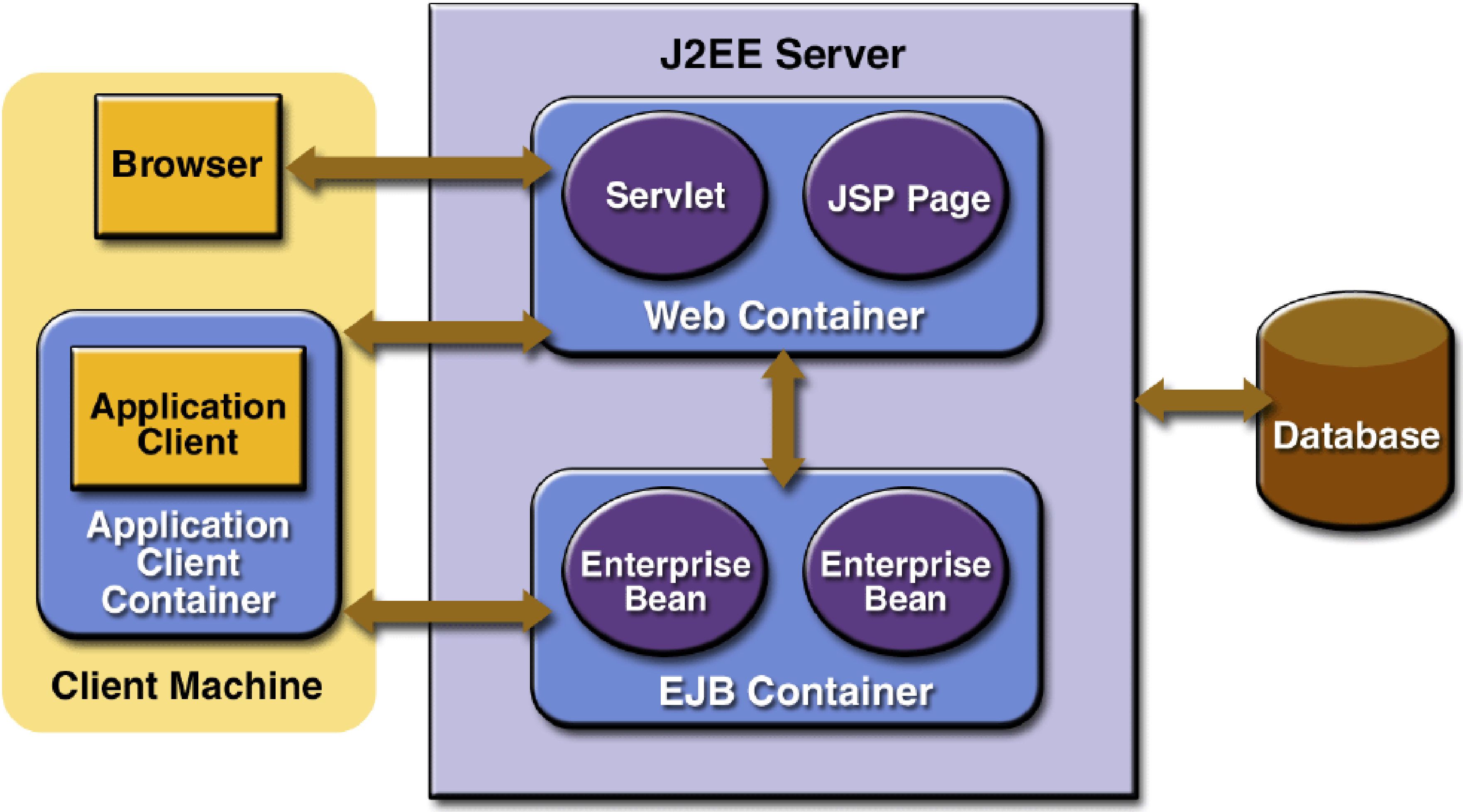
J2EE Server and Containers



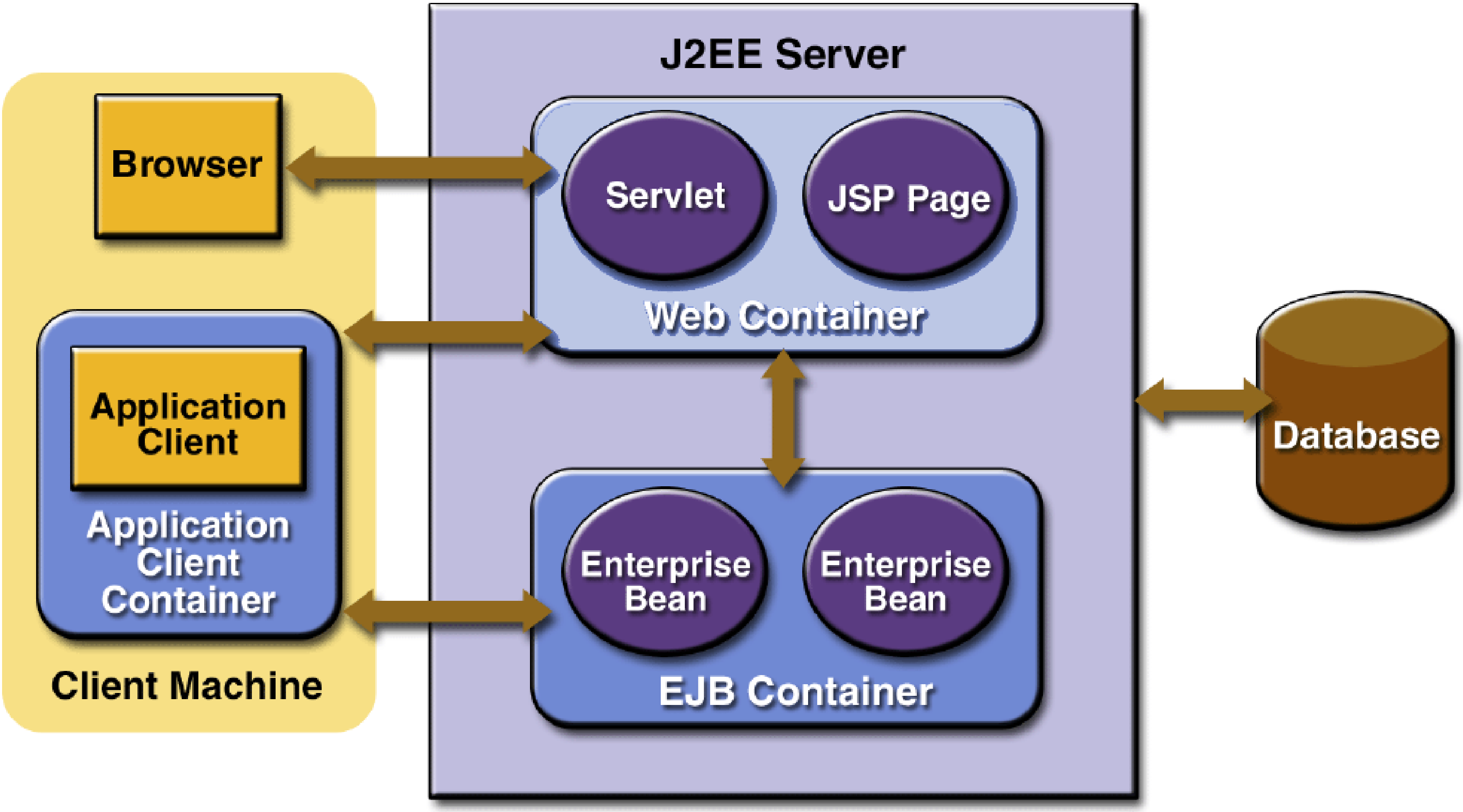
J2EE Server and Containers



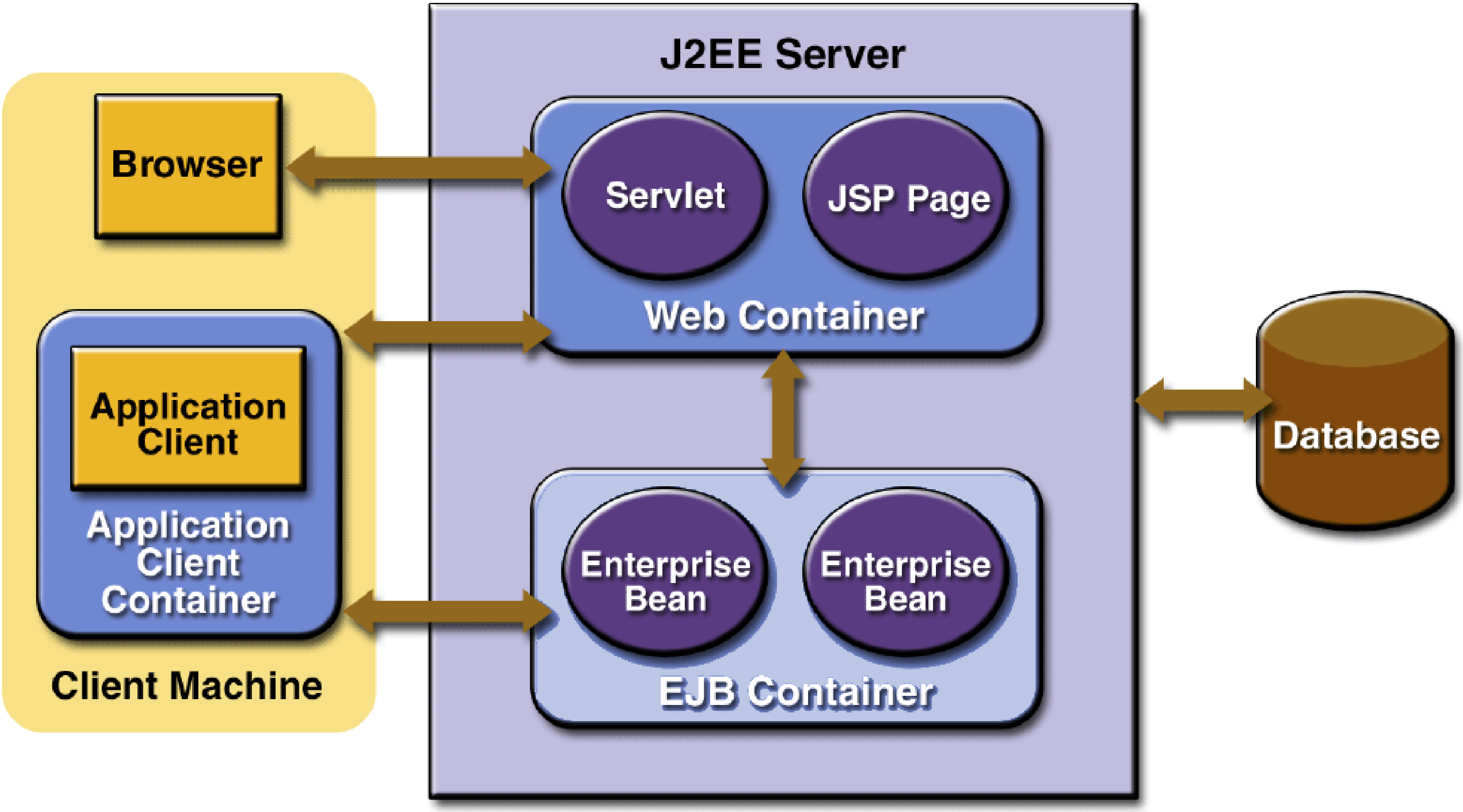
J2EE Server and Containers



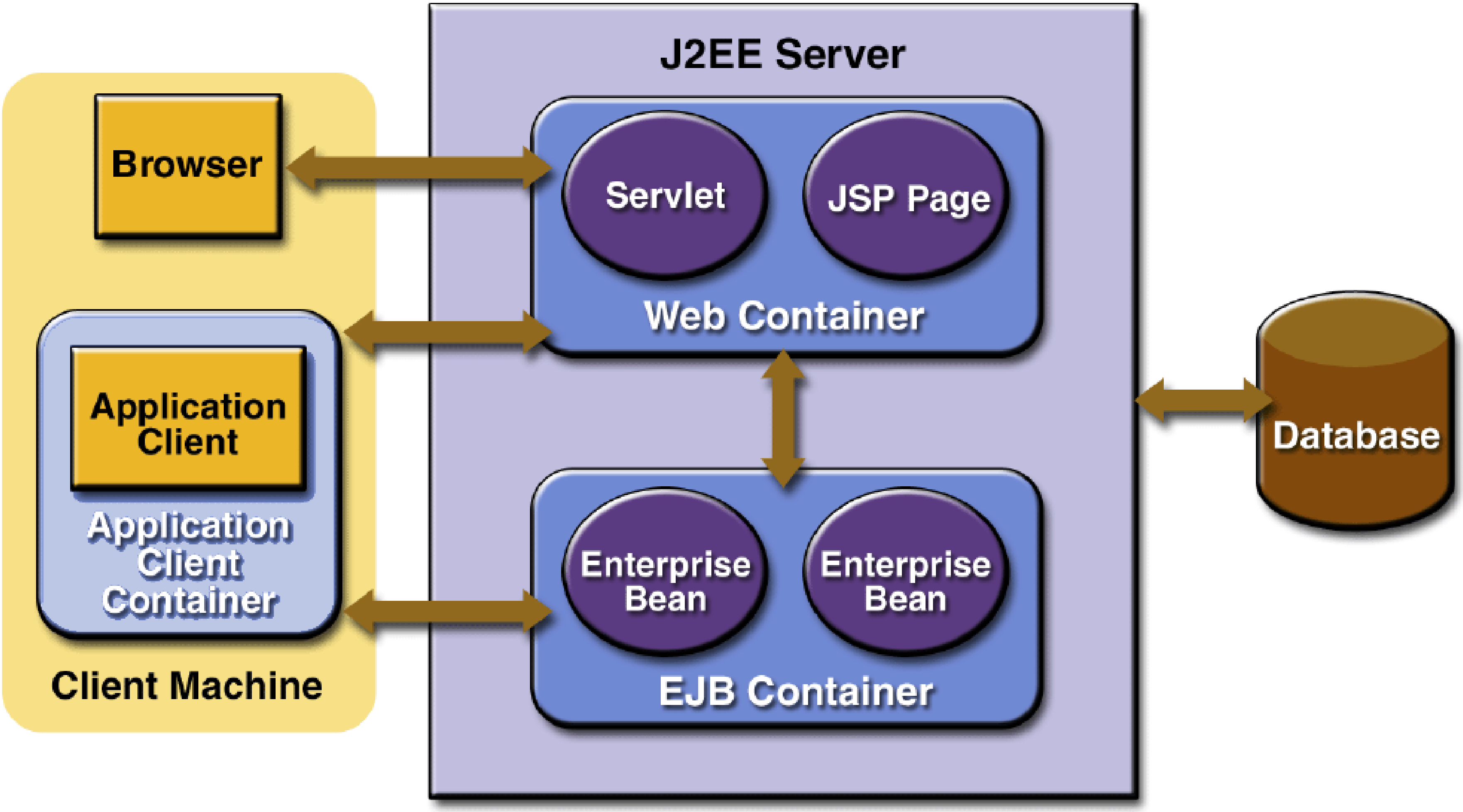
J2EE Server and Containers



J2EE Server and Containers



J2EE Server and Containers



Container Types

The deployment process installs J2EE application components in the J2EE containers.

Container Types

The deployment process installs J2EE application components in the J2EE containers.

J2EE server The runtime portion of a J2EE product. A J2EE server provides EJB and Web containers.

Container Types

The deployment process installs J2EE application components in the J2EE containers.

J2EE server The runtime portion of a J2EE product. A J2EE server provides EJB and Web containers.

Enterprise JavaBeans (EJB) container Manages the execution of enterprise beans for J2EE applications. Enterprise beans and their container run on the J2EE server.

Web container Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

Web container Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

Application client container Manages the execution of application client components. Application clients and their container run on the client.

Web container Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

Application client container Manages the execution of application client components. Application clients and their container run on the client.

Applet container Manages the execution of applets. Consists of a Web browser and Java Plug-in running on the client together.

Packaging

A J2EE application is delivered in an Enterprise Archive (EAR) file. An EAR file is a standard Java Archive (JAR) file with an `.ear` extension.

Packaging

A J2EE application is delivered in an Enterprise Archive (EAR) file. An EAR file is a standard Java Archive (JAR) file with an `.ear` extension.

The EAR file contains J2EE modules. Using EAR files and modules makes it possible to assemble a number of different J2EE applications using some of the same components.

Packaging

A J2EE application is delivered in an Enterprise Archive (EAR) file. An EAR file is a standard Java Archive (JAR) file with an `.ear` extension.

The EAR file contains J2EE modules. Using EAR files and modules makes it possible to assemble a number of different J2EE applications using some of the same components.

No extra coding is needed; it is just a matter of assembling various J2EE modules into J2EE EAR files.

J2EE modules

A J2EE module consists of one or more J2EE components for the same container type and one component deployment descriptor of that type.

J2EE modules

A J2EE module consists of one or more J2EE components for the same container type and one component deployment descriptor of that type.

A deployment descriptor is an XML document with an `.xml` extension that describes a component's deployment settings.

J2EE modules

A J2EE module consists of one or more J2EE components for the same container type and one component deployment descriptor of that type.

A deployment descriptor is an XML document with an `.xml` extension that describes a component's deployment settings.

An enterprise bean module deployment descriptor, for example, declares transaction attributes and security authorizations for an enterprise bean.

The role of deployment descriptors

Because deployment descriptor information is declarative, it can be changed without modifying the bean source code.

The role of deployment descriptors

Because deployment descriptor information is declarative, it can be changed without modifying the bean source code.

At run time, the J2EE server reads the deployment descriptor and acts upon the component accordingly.

The role of deployment descriptors

Because deployment descriptor information is declarative, it can be changed without modifying the bean source code.

At run time, the J2EE server reads the deployment descriptor and acts upon the component accordingly.

A J2EE module without an application deployment descriptor can be deployed as a stand-alone module.

Types of J2EE modules

The four types of J2EE modules are:

- Enterprise JavaBeans modules contain class files for enterprise beans and an EJB deployment descriptor. EJB modules are packaged as JAR files with a `.jar` extension.

Types of J2EE modules

The four types of J2EE modules are:

- Enterprise JavaBeans modules contain class files for enterprise beans and an EJB deployment descriptor. EJB modules are packaged as JAR files with a `.jar` extension.

- Web modules contain JSP files, class files for servlets, GIF and HTML files, and a Web deployment descriptor. Web modules are packaged as JAR files with a `.war` (Web ARchive) extension.

- Resource adapter modules contain all Java interfaces, classes, native libraries, and other documentation, along with the resource adapter deployment descriptor. Together, these implement the Connector architecture for a particular EIS. Resource adapter modules are packages as JAR files with a `.rar` (Resource adapter ARchive) extension.

- Resource adapter modules contain all Java interfaces, classes, native libraries, and other documentation, along with the resource adapter deployment descriptor. Together, these implement the Connector architecture for a particular EIS. Resource adapter modules are packages as JAR files with a `.rar` (Resource adapter ARchive) extension.

- Application client modules contain class files and an application client deployment descriptor. Application client modules are packaged as JAR files with a `.jar` extension.

Web Services Support

Web services are Web-based enterprise applications that use open, Extensible Markup Language (XML)-based standards and transport protocols to exchange data with calling clients.

Web Services Support

Web services are Web-based enterprise applications that use open, Extensible Markup Language (XML)-based standards and transport protocols to exchange data with calling clients.

The J2EE platform provides the XML APIs and tools you need to quickly design, develop, test, and deploy Web services and clients that fully interoperate with other Web services and clients running on Java-based or non-Java-based platforms.

Web Services and J2EE XML APIs

It is easy to write Web services and clients with the J2EE XML APIs. All you do is pass parameter data to the method calls and process the data returned, or for document-oriented web services, send documents containing the service data back and forth.

Web Services and J2EE XML APIs

It is easy to write Web services and clients with the J2EE XML APIs. All you do is pass parameter data to the method calls and process the data returned, or for document-oriented web services, send documents containing the service data back and forth.

No low-level programming is needed because the XML API implementations do the work of translating the application data to and from an XML-based data stream that is sent over the standardized XML-based transport protocols.

XML and data transport

The translation of data to a standardized XML-based data stream is what makes Web services and clients written with the J2EE XML APIs fully interoperable.

XML and data transport

The translation of data to a standardized XML-based data stream is what makes Web services and clients written with the J2EE XML APIs fully interoperable.

This does not necessarily mean the data being transported includes XML tags because the transported data can itself be plain text, XML data, or any kind of binary data such as audio, video, maps, program files, CAD documents or the like.

Extensible Markup Language

Extensible Markup Language is a cross-platform, extensible, and text-based standard for representing data.

Extensible Markup Language

Extensible Markup Language is a cross-platform, extensible, and text-based standard for representing data.

When XML data is exchanged between parties, the parties are free to create their own tags to describe the data, set up schemas to specify which tags can be used in a particular kind of XML document, and use XML style sheets to manage the display and handling of the data.

A first XML example

For example, a Web service can use XML and a schema to produce price lists, and companies that receive the price lists and schema can have their own style sheets to handle the data in a way that best suits their needs.

A first XML example

For example, a Web service can use XML and a schema to produce price lists, and companies that receive the price lists and schema can have their own style sheets to handle the data in a way that best suits their needs.

- One company might put the XML pricing information through a program to translate the XML to HTML so it can post the price lists to its Intranet.

- A partner company might put the XML pricing information through a tool to create a marketing presentation.

- A partner company might put the XML pricing information through a tool to create a marketing presentation.
- Another company might read the XML pricing information into an application for processing.

HTTP-SOAP Transport Protocol

Client requests and Web service responses are transmitted as Simple Object Access Protocol (SOAP) messages over HTTP to enable a completely interoperable exchange between clients and Web services all running on different platforms and at various locations on the Internet.

HTTP-SOAP Transport Protocol

Client requests and Web service responses are transmitted as Simple Object Access Protocol (SOAP) messages over HTTP to enable a completely interoperable exchange between clients and Web services all running on different platforms and at various locations on the Internet.

HTTP is a familiar request and response standard for sending messages over the Internet, and SOAP is an XML-based protocol that follows the HTTP request and response model.

The SOAP portion of a transported message handles the following:

- Defines an XML-based envelope to describe what is in the message and how to process the message.

- Includes XML-based encoding rules to express instances of application-defined data types within the message.

- Defines an XML-based convention for representing the request to the remote service and the resulting response.

WSDL Standard Format

The Web Services Description Language (WSDL) is a standardized XML format for describing network services.

WSDL Standard Format

The Web Services Description Language (WSDL) is a standardized XML format for describing network services.

The description includes the name of the service, the location of the service, and how to communicate with the service. WSDLs can be stored in UDDI registries and/or published on the Web.

WSDL Standard Format

The Web Services Description Language (WSDL) is a standardized XML format for describing network services.

The description includes the name of the service, the location of the service, and how to communicate with the service. WSDLs can be stored in UDDI registries and/or published on the Web.

The J2EE platform provides a tool for generating the WSDL for a Web service that uses remote procedure calls to communicate with clients.

UDDI and ebXML Standard Formats

Other XML-based standards such as Universal Description, Discovery, and Integration (UDDI) and ebXML (Electronic Business XML) make it possible for businesses to publish information on the Internet about their products and Web services where the information can be readily and globally accessed by clients who want to do business.

J2EE APIs—Enterprise JavaBeans

An Enterprise JavaBeans™ (EJB™) component or enterprise bean is a body of code with fields and methods to implement modules of business logic.

J2EE APIs—Enterprise JavaBeans

An Enterprise JavaBeans™ (EJB™) component or enterprise bean is a body of code with fields and methods to implement modules of business logic.

You can think of an enterprise bean as a building block that can be used alone or with other enterprise beans to execute business logic on the J2EE server.

J2EE APIs—Enterprise JavaBeans

An Enterprise JavaBeans™ (EJB™) component or enterprise bean is a body of code with fields and methods to implement modules of business logic.

You can think of an enterprise bean as a building block that can be used alone or with other enterprise beans to execute business logic on the J2EE server.

There are three kinds of enterprise beans: session beans, entity beans, and message-driven beans.

Enterprise beans and databases

Enterprise beans often interact with databases.

Enterprise beans and databases

Enterprise beans often interact with databases.

One of the benefits of entity beans is that you do not have to write any SQL code or use the JDBC™ API directly to perform database access operations; the EJB container handles this for you.

Enterprise beans and databases

Enterprise beans often interact with databases.

One of the benefits of entity beans is that you do not have to write any SQL code or use the JDBC™ API directly to perform database access operations; the EJB container handles this for you.

However, if you override the default container-managed persistence for any reason, you will need to use the JDBC API.

Enterprise beans and databases

Enterprise beans often interact with databases.

One of the benefits of entity beans is that you do not have to write any SQL code or use the JDBC™ API directly to perform database access operations; the EJB container handles this for you.

However, if you override the default container-managed persistence for any reason, you will need to use the JDBC API.

Also, if you choose to have a session bean access the database, you have to use the JDBC API.

J2EE APIs—Java Servlet Technology

Java Servlet technology lets you define HTTP-specific servlet classes.

J2EE APIs—Java Servlet Technology

Java Servlet technology lets you define HTTP-specific servlet classes.

A servlet class extends the capabilities of servers that host applications accessed by way of a request-response programming model.

J2EE APIs—Java Servlet Technology

Java Servlet technology lets you define HTTP-specific servlet classes.

A servlet class extends the capabilities of servers that host applications accessed by way of a request-response programming model.

Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers.

J2EE APIs—JavaServer Pages Technology

JavaServer Pages™ (JSP™) technology lets you put snippets of servlet code directly into a text-based document.

J2EE APIs—JavaServer Pages Technology

JavaServer Pages™ (JSP™) technology lets you put snippets of servlet code directly into a text-based document.

A JSP page is a text-based document that contains two types of text: static template data, which can be expressed in any text-based format such as HTML, WML, and XML, and JSP elements, which determine how the page constructs dynamic content.

J2EE APIs—Java Message Service

The Java Message Service (JMS) is a messaging standard that allows J2EE application components to create, send, receive, and read messages.

J2EE APIs—Java Message Service

The Java Message Service (JMS) is a messaging standard that allows J2EE application components to create, send, receive, and read messages.

It enables distributed communication that is loosely coupled, reliable, and asynchronous.

J2EE APIs—Java Naming and Directory Interface

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality.

J2EE APIs—Java Naming and Directory Interface

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality.

It provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

J2EE APIs—Java Naming and Directory Interface

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality.

It provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

Using JNDI, a J2EE application can store and retrieve any type of named Java object.

JNDI—naming environments

J2EE naming services provide application clients, enterprise beans, and Web components with access to a JNDI naming environment.

JNDI—naming environments

J2EE naming services provide application clients, enterprise beans, and Web components with access to a JNDI naming environment.

A naming environment allows a component to be customized without the need to access or change the component's source code.

JNDI—naming environments

J2EE naming services provide application clients, enterprise beans, and Web components with access to a JNDI naming environment.

A naming environment allows a component to be customized without the need to access or change the component's source code.

A container implements the component's environment and provides it to the component as a JNDI naming context.

JNDI and legacy applications

Because JNDI is independent of any specific implementations, applications can use JNDI to access multiple naming and directory services, including existing naming and directory services such as LDAP, NDS, DNS, and NIS.

JNDI and legacy applications

Because JNDI is independent of any specific implementations, applications can use JNDI to access multiple naming and directory services, including existing naming and directory services such as LDAP, NDS, DNS, and NIS.

This allows J2EE applications to coexist with legacy applications and systems.

J2EE APIs—Java Transaction API

The Java Transaction API (JTA) provides a standard interface for demarcating transactions.

J2EE APIs—Java Transaction API

The Java Transaction API (JTA) provides a standard interface for demarcating transactions.

The J2EE architecture provides a default auto commit to handle transaction commits and rollbacks.

J2EE APIs—Java Transaction API

The Java Transaction API (JTA) provides a standard interface for demarcating transactions.

The J2EE architecture provides a default auto commit to handle transaction commits and rollbacks.

An auto commit means that any other applications viewing data will see the updated data after each database read or write operation.

However, if your application performs two separate database access operations that depend on each other, you will want to use the JTA API to demarcate where the entire transaction, including both operations, begins, rolls back, and commits.

J2EE APIs—JavaMail API

J2EE applications can use the JavaMail$^{\text{TM}}$ API to send e-mail notifications.

J2EE APIs—JavaMail API

J2EE applications can use the JavaMail™ API to send e-mail notifications.

The JavaMail API has two parts: an application-level interface used by the application components to send mail, and a service provider interface.

J2EE APIs—JavaMail API

J2EE applications can use the JavaMail™ API to send e-mail notifications.

The JavaMail API has two parts: an application-level interface used by the application components to send mail, and a service provider interface.

The J2EE platform includes JavaMail with a service provider that allows application components to send Internet mail.

J2EE APIs—JavaBeans Activation Framework

The JavaBeans Activation Framework (JAF) is included because JavaMail uses it.

J2EE APIs—JavaBeans Activation Framework

The JavaBeans Activation Framework (JAF) is included because JavaMail uses it.

It provides standard services to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and create the appropriate JavaBeans component to perform those operations.

J2EE APIs—Java API for XML Processing

The Java API for XML Processing (JAXP) supports the processing of XML documents using Document Object Model (DOM), Simple API for XML Parsing (SAX), and XML Stylesheet Language Transformation (XSLT).

J2EE APIs—Java API for XML Processing

The Java API for XML Processing (JAXP) supports the processing of XML documents using Document Object Model (DOM), Simple API for XML Parsing (SAX), and XML Stylesheet Language Transformation (XSLT).

JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation.

J2EE APIs—Java API for XML Processing

The Java API for XML Processing (JAXP) supports the processing of XML documents using Document Object Model (DOM), Simple API for XML Parsing (SAX), and XML Stylesheet Language Transformation (XSLT).

JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation.

JAXP also provides namespace support, which lets you work with schemas that might otherwise have naming conflicts.

J2EE APIs—Java API for XML Registries

The Java API for XML Registries (JAXR) lets you access business and general-purpose registries over the Web.

J2EE APIs—Java API for XML Registries

The Java API for XML Registries (JAXR) lets you access business and general-purpose registries over the Web.

JAXR supports the ebXML Registry/Repository standards and the emerging UDDI specifications.

J2EE APIs—Java API for XML Registries

The Java API for XML Registries (JAXR) lets you access business and general-purpose registries over the Web.

JAXR supports the ebXML Registry/Repository standards and the emerging UDDI specifications.

By using JAXR, developers can learn a single API and get access to both of these important registry technologies.

J2EE APIs—Java API for XML-Based RPC

The Java API for XML-based RPC (JAX-RPC) uses the SOAP standard and HTTP so client programs can make XML-based remote procedure calls (RPCs) over the Internet.

J2EE APIs—Java API for XML-Based RPC

The Java API for XML-based RPC (JAX-RPC) uses the SOAP standard and HTTP so client programs can make XML-based remote procedure calls (RPCs) over the Internet.

JAX-RPC also supports WSDL so you can import and export WSDL documents.

J2EE APIs—Java API for XML-Based RPC

The Java API for XML-based RPC (JAX-RPC) uses the SOAP standard and HTTP so client programs can make XML-based remote procedure calls (RPCs) over the Internet.

JAX-RPC also supports WSDL so you can import and export WSDL documents.

With JAX-RPC and a WSDL, you can easily interoperate with clients and services running on Java-based or non-Java-based systems such as Microsoft's .NET platform.

JAX-RPC and authentication

JAX-RPC relies on the HTTP transport protocol. Taking that a step further, JAX-RPC lets you create service applications that combine HTTP with a Java technology version of the Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols to establish basic or mutual authentication.

JAX-RPC and authentication

JAX-RPC relies on the HTTP transport protocol. Taking that a step further, JAX-RPC lets you create service applications that combine HTTP with a Java technology version of the Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols to establish basic or mutual authentication.

SSL and TLS ensure message integrity by providing data encryption with client and server authentication capabilities.

J2EE Connector Architecture

The J2EE Connector architecture is used by J2EE tools vendors and system integrators to create resource adapters that support access to enterprise information systems that can be plugged into any J2EE product.

J2EE Connector Architecture

The J2EE Connector architecture is used by J2EE tools vendors and system integrators to create resource adapters that support access to enterprise information systems that can be plugged into any J2EE product.

A resource adapter is a software component that allows J2EE application components to access and interact with the underlying resource manager.

Because a resource adapter is specific to its resource manager, there is typically a different resource adapter for each type of database or enterprise information system.

Because a resource adapter is specific to its resource manager, there is typically a different resource adapter for each type of database or enterprise information system.

JAX-RPC and the J2EE Connector Architecture are complementary technologies for enterprise application integration (EAI) and end-to-end business integration.

The J2EE Connector Architecture also provides a performance-oriented, secure, scalable, and message-based transactional integration of J2EE-based Web services with existing EISs that can be either synchronous or asynchronous.

The J2EE Connector Architecture also provides a performance-oriented, secure, scalable, and message-based transactional integration of J2EE-based Web services with existing EISs that can be either synchronous or asynchronous.

Existing applications and EISs integrated through the J2EE Connector Architecture into the J2EE platform can be exposed as XML-based Web services using JAX-RPC and J2EE component models.

Simplified Systems Integration

The J2EE platform is a platform-independent, full systems integration solution that creates an open marketplace in which every vendor can sell to every customer.

Simplified Systems Integration

The J2EE platform is a platform-independent, full systems integration solution that creates an open marketplace in which every vendor can sell to every customer.

Such a marketplace encourages vendors to compete, not by trying to lock customers into their technologies but by trying to outdo each other by providing products and services that benefit customers, such as better performance, better tools, or better customer support.

Systems integration with the J2EE APIs

- Unified application model across tiers with enterprise beans
- Simplified response and request mechanism with JSP pages and servlets
- XML-based data interchange integration with JAXP
- Simplified interoperability with the J2EE Connector Architecture
- Easy database connectivity with JDBC
- Enterprise application integration with message-driven beans, JMS, JTA, and JNDI