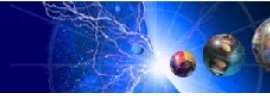


J2EE Overview



Agenda



- J2EE Application Roles
- J2EE Components
- J2EE Services
- J2EE Communications
- References (EJB, Resource, Security Role)
- J2EE Packaging



What is Java2, Enterprise Edition?



- A set of related specifications: A single standard for implementing and deploying enterprise applications
- Current J2EE specification level is 1.2. WebSphere Application Server 4.0 fully supports J2EE 1.2
- Consensus and collaboration from numerous major enterprise software vendors. IBM contributed to over 80% of the J2EE APIs



- ▶ Customers can use J2EE as a reference to identify product capabilities before they invest in a vendor (technology provider).
- ▶ Sun licenses the technology of J2EE to other companies (e.g., Web technology providers).
- ▶ The more companies that license J2EE, the more J2EE is validated by the marketplace. More companies will want to license it!
- ▶ Web Technology Providers (like IBM!)
- ▶ Technology Providers want customers to recognize their offerings (tools, servers, etc.) as capable of meeting the customer's needs.

Benefits of J2EE



- Simplified architecture and development
 - ▶ Variety of standard services, components, and clients
 - ▶ Choices of tools
 - ▶ Portability
 - ▶ Integration with existing information systems
- Separation of Responsibilities
- Scalability
- Flexible security model



- ▶ J2EE defines a simple standard that applies to all aspects of architecting and developing large scale applications. J2EE naturally encourages the adoption of a multiple tier architecture and a strict separation between business logic and the presentation layer.
- ▶ The architectural approach suggested by the J2EE standards lends itself to the implementation of highly scalable infrastructures, such as WebSphere, where the application workload can be transparently distributed across a number of concurrent processes, or even across a number of parallel systems.
- ▶ J2EE provides technologies to facilitate the integration of existing applications with newly developed J2EE applications.
- ▶ The specifications are generic enough to allow an end customer to choose among a wide array of middleware, tools, hardware, operating systems, and application designs. J2EE has sparked a whole new market of application servers, development tools, connectors, and components. IBM is in the leading position with a vast portfolio of best-of-breed products.
- ▶ Security is one of the premier areas of focus in J2EE. The security specifications have been significantly redesigned to better fit the demands of Internet-based environments and to better integrate with legacy security schemes.

J2EE Technologies Summary



J2EE 1.2 Required Standard Extension APIs

API	Applet	Application Client	Web	EJB
JDBC 2.0	N	Y	Y	Y
JTA 1.0	N	N	Y	Y
JNDI 1.2	N	Y	Y	Y
Servlet 2.2	N	N	Y	N
JSP 1.1	N	N	Y	N
EJB 1.1	N	Y ¹	Y ²	Y
RMI-IIOP 1.0	N	Y	Y	Y
JMS 1.0	N	Y	Y	Y
JavaMail 1.1	N	N	Y	Y
JAF 1.0	N	N	Y	Y

1 Application clients can only make use of the enterprise bean client APIs.

2 Servlets and JSP pages can only make use of the enterprise bean client APIs.



- ▶ J2EE Application components execute in runtime environments provided by the containers that are part of the J2EE platform. The J2EE platform supports four separate types of containers, one for each J2EE application component type: application client containers, applet containers, web containers for servlets and JSPs and enterprise java bean containers.
- ▶ The J2EE platform also includes a number of Java Standard extensions. The table indicates which standard extensions are required to be available in each type of container and also indicates the required version of the Standard Extension.

WebSphere 4.0 JDK Level



- WebSphere 4.0 uses JDK level 1.3

- WebSphere 3.5 used JDK 1.2.2

- Major Performance Enhancements for V4.0
 - ▶ IBM JRE 1.3 results are improved by 22% over JRE 1.2.2
 - ▶ Improvements in JVM, JIT Compiler, Garbage Collection and Threads Management





- ▶ The J2EE Specification breaks the life cycle of an application into six distinct roles or responsibilities.
- ▶ Output from one role is often the input to the next.
- ▶ In practice, two roles may be performed by a single person or group, and a role may be split across several people or groups.

J2EE Platform Roles



- J2EE Product Provider
 - ▶ Implements the J2EE Product APIs
 - ▶ IBM is a J2EE Product Provider (WebSphere 4.0)

- Tool Provider
 - ▶ Implements tools which enable other tasks on the J2EE Platform
 - ▶ Development Environments
 - ▶ Application Assembly Tools
 - ▶ Monitoring Tools
 - ▶ IBM is a Tool Provider(Visual Age for Java, Tivoli etc)



▶ J2EE Product Provider

The company that designs and makes available for purchase the J2EE platform, APIs, and other features defined in the J2EE specification. Product providers are operating system, database system, application server, or Web server vendors who implement the J2EE platform.

▶ Tool Provider

The person or company who makes development, assembly, and packaging tools used by component providers, assemblers, and deployers.

J2EE Platform Roles



- **Application Component Provider**
Responsible for developing components of the application and the deployment descriptors.

- **Application Assembler**
Combines components from Component Providers into a single application.

- **Deployer**
Installs and configures the application into the Runtime Environment.

- **System Administrator**
Monitors the Runtime Environment.



- ▶ **Application Component Provider**
The company or person who creates web components, enterprise beans, applets, or application clients for use in J2EE applications.

- ▶ **Application Assembler**
The company or person who obtains application component JAR files from Component Providers and assembles the JAR files into a J2EE application EAR file.
The Assembler or Deployer can edit the deployment descriptor directly or use tools that correctly add XML tags according to interactive selections. A software developer performs the following tasks to deliver an EAR file containing the J2EE application:
 - 1) Assembles EJB JAR and Web components (WAR) files created in the previous phases into a J2EE application EAR file.
 - 2) Specifies the deployment descriptor for the J2EE application.
 - 3) Verifies that the contents of the EAR file are well formed and comply with the J2EE specification.

- ▶ **Deployer**
Deploys (installs) the J2EE application into the J2EE server.

- ▶ **System Administrator**
Administers the computing and networking infrastructure where J2EE applications run and oversees the Runtime Environment.

IBM WebSphere Application Server V4.0

J2EE Components



J2EE Platform Technologies

■ Components

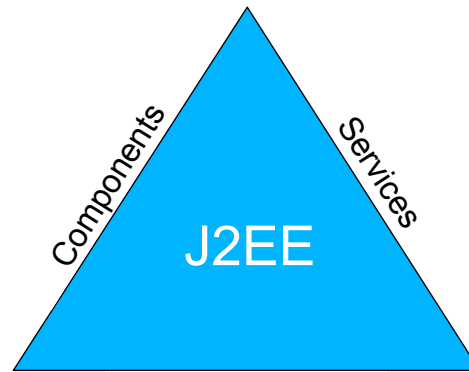
- ▶ Client Side Components
 - Applets
 - Application Clients
- ▶ Server Side Components
 - EJBs
 - Web Components (Servlets, JSP)

■ Services

- ▶ Functions used by J2EE components
- ▶ APIs implemented by J2EE platform provider (WebSphere)

■ Communication

- ▶ Enables communication between collaborating components
- ▶ Provided by containers



- ▶ Components are to be provided by the application developers and include servlets, Java Server Pages, and Enterprise Java Beans.
- ▶ The services are functions that are accessible to the components via a standard set of APIs. Each level of the J2EE specification will call for supporting a specific level of each set of APIs. For example, the current level of J2EE supported by WebSphere V4.0 is 1.2, and that implies support for JDBC level 2.0.
- ▶ The essence of J2EE is the definition of a distributed, object-oriented infrastructure. Components need to be able to communicate with each other in this distributed world and therefore the containers need to provide the appropriate communication mechanisms to make this happen.

J2EE Components Overview



- Implemented by the Application Component Provider
- Assembled by the Application Assembler
- Deployed by the Deployer
- Managed by the System Administrator
- Run inside containers provided by the J2EE Product Provider
- Implemented, Assembled, Deployed, and Managed with tools provided by the Tool Provider



J2EE Components: Application Clients



- Client-side components that execute inside a J2EE-compliant "client container"
 - ▶ The client container provides security, communication, and other required J2EE services.

- Application clients can interact with:
 - ▶ EJB through RMI/IIOP
 - ▶ Web components through HTTP/HTTPS

- Deployment is platform specific
 - ▶ Details are not specified by J2EE



- ▶ Applets are GUI components that usually run in a Web browser, but can run in a variety of other applications or devices that support the applet programming model. Applets can be used to provide a powerful user interface for J2EE applications

- ▶ Application Clients are Java Programming Language programs that are often GUI programs that execute on a desktop computer. Application Clients offer a user similar experience to that of native applications and have access to all of the facilities of the J2EE middle tier.

Server Side Component - JSP



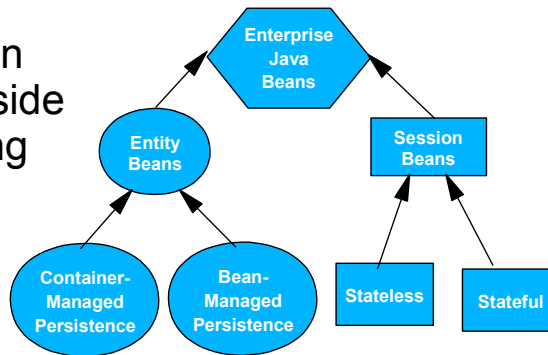
- HTML documents
 - ▶ Embedded JSP-specific tags
 - ▶ Inline Java Code
- On the server, a JSP is parsed and compiled into a Java Servlet
- WebSphere V4.0 fully supports JSP1.1
- JSP 0.91 and 1.0 are removed in WebSphere V4.0



- ▶ JSP enables server-side scripting using Java. Server-side scripting is not an alternative to client-side scripting. Client-side scripting (JavaScript or Java Applet) is important for performing input validity tests and other computation eliminating interaction with the server.
- ▶ A JSP is an HTML document with special embedded tags and inline Java code. At runtime, the JSP is parsed and compiled into a Java servlet.
- ▶ JSPs enable you to effectively separate the HTML coding from the business logic in your Web pages.
- ▶ JSPs can be built using a Web page editor.
- ▶ JSPs provide a simpler mechanism to create dynamic content than coding a servlet.
- ▶ This parsing, code generation, compilation need only be performed once (if .jsp file does not change).
- ▶ The loading and execution is a separate step which follows normal servlet loading and execution rules.
- ▶ If the "JSP servlet" is currently loaded and a request for the JSP page is made, the service method for the JSP servlet service method is invoked.
- ▶ If the JSP servlet exists (.class file) but is not yet loaded it is loaded and then its service method is invoked.

Server-Side Components - EJBs

- The Enterprise JavaBean architecture is a server-side technology for developing and deploying business logic components.



- J2EE 1.2 specification includes EJB 1.1
- WebSphere V3.x supported EJB 1.0 beans with some EJB 1.1 functionality
- WebSphere V4.0 fully supports the EJB 1.1 specification.



- ▶ The Enterprise JavaBeans (EJB) standard is a server side Java based component architecture for building multi-tier, distributed object applications.
- ▶ EJBs have the same programming model as the client-side JavaBeans programming model, which makes it easy for Java programmers to use EJBs to build applications.
- ▶ There are two kinds of enterprise beans: session beans and entity beans. A session bean represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone. In contrast, an entity bean represents persistent data stored in (usually) one row of a database table. If the client terminates or if the server shuts down, the underlying services ensure the entity bean data is saved.
- ▶ Session bean characteristics: single client, short-lived, do not survive crashes
 - ▶ Session beans can be:
 - ▶ Stateless - useful for simple transactions where all information to complete the transaction is present at the same time. Stateless session beans are high performance and reusable. Used for single-stage transaction management.
 - ▶ Stateful - multiple stages are required to perform the transaction.
- ▶ Entity bean characteristics: multiple clients, long-lived, can survive a crash.
 - ▶ Entity beans can use:
 - ▶ Bean Managed Persistence (BMP) - the programmer controls data access. The beans contain the code to store and retrieve data.
 - ▶ Container Managed Persistence (CMP) - The EJS container handles all aspects of data access leaving the programmer to concentrate on business logic.

J2EE: EJB 1.1 Changes (over 1.0)



- In most cases, you do not need to modify the EJB 1.0 code to deploy in an EJB 1.1 environment

- Deployment:
 - ▶ In 1.0, Deployment information was not standardized
 - ▶ In 1.1, Deployment Descriptors:
 - Must be specified in XML format
 - WebSphere V4.0 Application Assembly Tool helps create the EJB deployment descriptor, or the deployment descriptor can be written manually

- JNDI lookup:
 - ▶ In EJB 1.0, JNDI lookup call is as follows:
 - `initialContext.lookup("com/ibm/Hello");`
 - ▶ In EJB 1.1, it looks like:
 - `initialContext.lookup("java:comp/env/ejb/com/ibm/Hello");`



Server-Side Components - Servlets



- Servlets are Java classes that allow application logic to be embedded in HTTP request-response process.
- J2EE 1.2 requires:
 - ▶ Servlet 2.2 specification
- Servlet 2.2 already in WebSphere V3.5.2
 - ▶ Exception: of the java:comp/ JNDI space
- Servlet 2.2 is fully supported in WebSphere V4.0



- ▶ Servlet 2.2 extends Servlet 2.1 with support for the packaging and deployment of Web applications, standalone, and as part of a J2EE application.
- ▶ Servlet2.2 also addresses security, both standalone and within the J2EE platform and provides support for the notion of a Web application.

Major changes in Servlet 2.2



- **Session scoping**
 - ▶ In 2.2, scoping is per Web application
 - Servlets in different Web applications running in the same servlet engine CANNOT share session information
 - ▶ In 2.1, scoping is per Servlet Engine
- **HTTPSession deprecation**
 - ▶ `getValue(String)` replaced by `getAttribute(String)`
 - ▶ `getValueNames()` replaced by `getAttributeNames()`
 - ▶ `putValue(String, Object)` replaced by `setAttribute(String, Object)`
 - ▶ `removeValue(String)` replaced by `removeAttribute(String)`
- **Removed package**
 - ▶ `com.ibm.servlet.connmgr`
 - Use JDBC connection pool API instead



- ▶ Servlet 2.1 and 2.2 changes are included in the appendix to this module.

IBM WebSphere Application Server V4.0

J2EE Services, Communication



J2EE Services: JNDI



- JNDI allows components to store and retrieve named Java objects
- J2EE 1.2 requires JNDI level 1.2
 - ▶ WebSphere V4.0 provides an integrated JNDI 1.2 compliant name service
- Containers provide two levels of naming schemes:
 - ▶ "Local," read-only, accessible to components
 - ▶ "Global," the actual JNDI namespace
 - ▶ Local names are bound to their global counterparts at deployment



- ▶ API that provides naming and directory functionality to Java programs.
- ▶ Naming services provide name-to-object mappings.
- ▶ Integral part of J2EE to lookup J2EE objects and resources.
- ▶ Independent of any specific directory access protocol.
- ▶ Allows easy deployment of new directory services.
- ▶ Allows manipulating Java instances by name.

J2EE Services: Using JNDI



- To access its naming context, a component creates a `javax.naming.InitialContext` object
 - ▶ WebSphere V4.0 provides a new naming service
 - `com.ibm.websphere.naming.WsnInitialContextFactory`
 - ▶ WebSphere V3.5.2+ uses
 - `com.ibm.ejs.ns.jndi.CNInitialContextFactory`
 - Deprecated in V4.0
- System-provided objects (such as `UserTransaction` objects) are stored in `java:comp/env` in the JNDI name space
- User-defined objects are stored in subcontexts of `java:comp/env`
 - ▶ `java:comp/env/ejb`
 - ▶ `java:comp/env/jdbc`



J2EE Services: JDBC



- Provides database-independent connectivity to a variety of data stores
- J2EE 1.2 specifies:
 - ▶ JDBC 2.0 Core APIs - Basic database services
 - ▶ JDBC 2.0 Extension APIs - Advanced functionality
 - Connection Pooling
 - Transactional capabilities
- Functionality provided by a combination of WebSphere and a compliant JDBC driver
- The JDBC 2.0 Core and Extension APIs are supported in WebSphere V3.5 and above.



J2EE Services: Security



- J2EE access control involves Authentication and Authorization

- Authentication - verify the user's identity

- Authorization - determine if the user has permission to access the requested resource
 - ▶ Authorization based on Roles
 - ▶ Roles can contain users or groups of users
 - Permissions are mapped by Deployer



- ▶ Basic authentication: the Web server authenticates a principal using the user name and password obtained from the Web client.
- ▶ Digest authentication: user name and a password, but transmitted in an encrypted form.
- ▶ Form-based authentication: the Web container can provide an application-specific form for logging in.
- ▶ Certificate authentication: the client uses a public key certificate to establish its identity and maintains its own security context.

J2EE Services: Transactions



- JTA is the API and JTS is the implementation

- JTA is an implementation-neutral interface to distributed transactions
 - ▶ J2EE requires JTA level 1.0
 - ▶ Implemented by combining WebSphere and a JDBC driver that supports JTA

- Two ways to start a transaction
 - ▶ Programmatic (in the code)
 - using the `javax.transaction.UserTransaction` interface
 - ▶ Declarative
 - using EJB Container-Managed Transaction



- ▶ JTA is oriented at application developers and JTS is oriented at Middleware providers.
- ▶ The Java Transaction API (JTA) allows applications to access transactions in a manner that is independent of specific implementations. JTA specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the transactional application, the J2EE server, and the manager that controls access to the shared resources affected by the transactions.

- ▶ The Java Transaction Service (JTS) specifies the implementation of a transaction manager that supports JTA and implements the Java mapping of the Object Management Group Object Transaction Service 1.1 specification. A JTS transaction manager provides the services and management functions required to support transaction demarcation, transactional resource management, synchronization, and propagation of information that is specific to a particular transaction instance.

J2EE: Container Managed Transaction



- EJB container is responsible for managing transaction boundaries
- Dictated by the transaction attribute modes specified for the EJB method (in EJB DD)
 - ▶ Required
 - ▶ Requires New
 - ▶ Not Supported
 - ▶ Supports
 - ▶ Mandatory
 - ▶ Never (new in EJB 1.1)
- Concurrent transactions isolation is dictated by isolation level - No longer in EJB 1.1 spec
 - ▶ Read, Repeatable-Read, Read-Committed, Serializable



- ▶ These two transactional specifiers, the transaction attribute and the isolation attribute, are set in the deployment descriptor. When the bean is deployed, the container generates the necessary transaction support code. The values chosen for these specifiers give the container the information needed to generate that support code.
- ▶ Transaction Attribute
 - ▶ Required: Use client TransactionContext if present. If not, create a new TransactionContext.
 - ▶ Required New: Always create a new transaction.
 - ▶ Not supported: Don't propagate client transaction, if present.
 - ▶ Supported: Use the client transaction, if present.
 - ▶ Mandatory: Use client transaction, if present. If not present, , throw TransactionRequiredException
 - ▶ Never: If client has transaction, throw Exception.

J2EE Communications - RMI/IIOP



- Remote Method Invocation (RMI) turns local method call into remote method calls
 - ▶ From the application view, it just makes a method call
 - ▶ The infrastructure takes care of calling the remote method and receiving the response

- EJB clients use RMI/IIOP to communicate with other EJBs

- J2EE 1.2 supports RMI/IIOP level 1.0



- ▶ An Application Component Provider defines the interface of a remote object in an IDL, then uses an IDL compiler to generate client and server stubs that connect object implementations to an Object Request Broker (ORB). An ORB is a library that enables CORBA objects to locate and communicate with one another. ORBs communicate with each other using the Internet Inter-ORB Protocol (IIOP).
- ▶ RMI-IIOP contains:
- ▶ The *rmic* compiler, which generates:
 - ▶ Client and server stubs that work with any ORB.
 - ▶ An IDL file compatible with the RMI interface. To create a C++ server object, an Application Component Provider would use an IDL compiler to produce the server stub and skeleton for the server object.
 - ▶ A CORBA API and ORB

J2EE Communications: JMS



- JMS - Java Messaging Service
 - ▶ Reliable interface for asynchronous sending and receiving of messages
 - ▶ Loosely coupled Communication
 - ▶ Point-to-point messaging
 - ▶ Publish-subscribe messaging

- JMS 1.0 requires JMS transactions interacting with a single JMS resource provider only: One-Phase Commit

- J2EE 1.3 requires support for distributed JMS Transaction (JMS/XA): Two-Phase Commit.



- ▶ Messaging communication is:
 - ▶ Peer to peer: Any client can send/receive messages to/from another client
 - ▶ Loosely coupled: sender and receiver do not have to be available at the same time to communicate. RMI is tightly coupled.
 - ▶ Asynchronous: A JMS provider can deliver messages to a client as they arrive; a client does not have to request messages in order to receive them.
 - ▶ Reliable: The JMS API can ensure that a message is delivered once and only once. Lower levels of reliability are available for applications that can afford to miss messages or receive duplicate messages.
- ▶ Point-to-point messaging:
 - ▶ A client sends a message to the message queue of another client
 - ▶ Each message has one receiver
- ▶ Publish-subscribe messaging:
 - ▶ Clients subscribe to a well-known node called a topic
 - ▶ All subscribers receive a message sent to the topic
 - ▶ The topic adjusts as publishers and subscribers activate and deactivate.
- ▶ JMS/XA is not available in WebSphere V4.0 AEs
- ▶ WebSphere AE V4.0 supports JMS/XA. JMS/XA requires MQSeries.

J2EE Communications: JavaMail and JAF

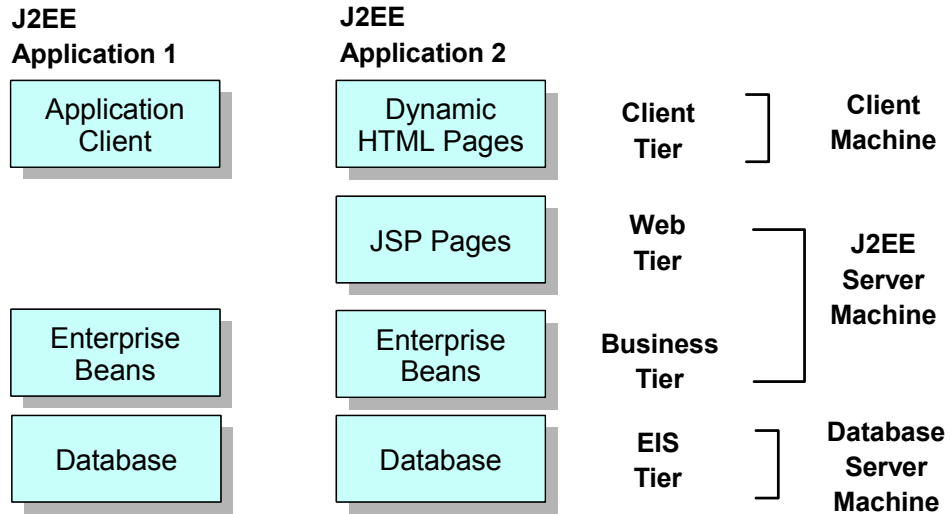


- JavaMail
 - ▶ API that provides abstract classes for e-mail
 - ▶ J2EE 1.2 requires support for JavaMail 1.1
 - ▶ Requires JAF to handle non-plain text mail content (MIME, URL, file attachments)
- JAF - JavaBean Activation Framework
 - ▶ MIME type support
 - ▶ Used by JavaMail to handle data in email messages
 - ▶ Not intended for typical application use
 - Advanced e-mail interactions may require using it
 - ▶ J2EE1.2 requires support for JAF 1.0.



- ▶ JavaMail
 - ▶ Provides APIs for reading, composing, sending emails.
 - ▶ APIs Model a mail system
 - ▶ Provides Platform to allow Web Components to interact with an email system.
 - ▶ APIs require service providers to implement specific protocols
 - ▶ Send Mail - SMTP
 - ▶ Receive/Store Mail - POP3, IMAP

J2EE - Multiple Tiers



► EIS - Enterprise Information System

J2EE Container

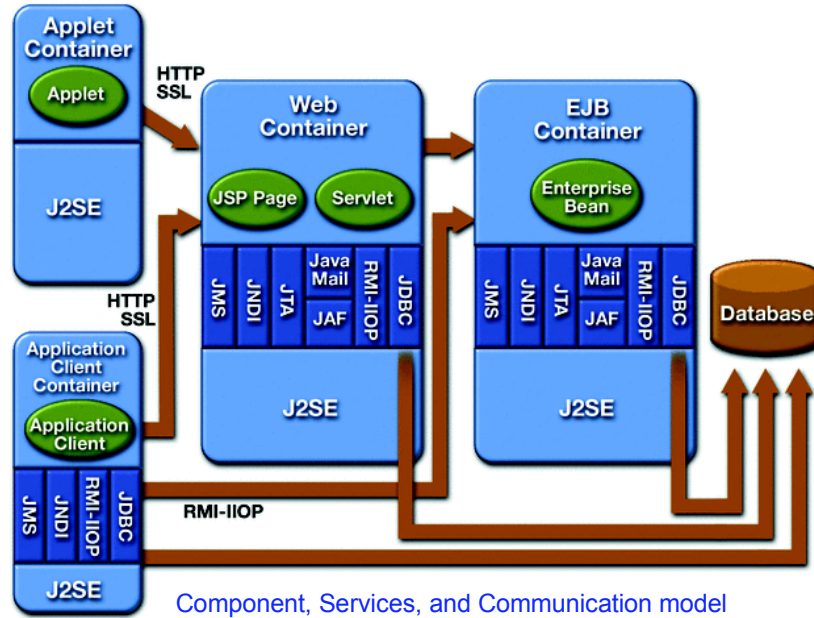


- Each component runs inside a container that is provided by the J2EE platform provider.
- This container provides specific deployment and run-time services to the component (life cycle, security, transactions, etc.).
- What services are offered depends on the type of container.



- ▶ Containers provide the runtime support for the application components. A container provides a federated view of the underlying J2EE APIs to the application components. Interposing a container between the application components and the J2EE services allows the container to inject services defined by the components' deployment descriptors, such as declarative transaction management, security checks, resource pooling, and state management.
- ▶ A typical J2EE product will provide a container for each application component type: application client container, applet container, Web component container, and enterprise bean container.

J2EE Object Model



- ▶ All segments of a J2EE applications are packaged in a separate container.
- ▶ According to J2EE, not only the Enterprise JavaBeans, but also servlets, JSPs, and Java "thick" clients have their own separate containers. This allows for a very clean separation of roles and for a very natural mapping into tiers: the client container implements the GUI, or physical rendition of the information displayed. The Web Container, where JSPs and servlets reside, implements the presentation logic. The EJB container holds the business logic.
- ▶ The containers and the code that is contained therein take advantage of wide array of standard J2EE functions and APIs provided by the J2EE platform.
- ▶ The Applet Container is the Web browser and Java Plug-in combination running on the Client Machine.
- ▶ The J2EE specification requires that these containers provide a Java-compatible Runtime Environment as defined by the Java 2 Platform, Standard Edition v1.2 specification (J2SE). The applet container may use the Java plugin product to provide this environment, or it may provide it natively.
- ▶ The J2EE platform includes a database, accessible through the JDBC API, for the storage of business data. The database is accessible from Web components, enterprise beans, and application client components. The database need not be accessible from applets.

J2EE References



References



- Several objects in the J2EE Model are looked up through up from a JNDI name service
 - ▶ EJBs
 - ▶ JDBC DataSources
 - ▶ JavaMail Providers

- When the object is created by the Deployer, it is registered with the JNDI service under a specific name
 - ▶ `ejb/AccountEJB`
 - ▶ `jdbc/DB2DS`



References



- To use one of these objects, the Application Component Provider programs a look-up from the name service.
 - ▶ Object myEJB = `initialContext.lookup("ejb/AccountEJB");`
 - ▶ Object myDS = `initialContext.lookup("ejb/DB2DS");`

- The problem is that the Application Component Provider needs to know what the Deployer is going to call the object.



References



- To solve this problem, the J2EE specifications defines the concept of references
- With references, the Application Component Providers call the component whatever they want to, and write the code accordingly.
 - ▶ Object myEJB =
 initialContext.lookup("java:comp/env/ejb/myEJB");
- The Application Component Provider also adds an entry to the Deployment Descriptor stating that the code references an EJB called `ejb/myEJB`.



References



- The Deployer must substitute the real JNDI names for the internal names used by the Application Component Provider when the Application is installed
 - ▶ In the example, the Deployer would link `ejb/myEJB` with `ejb/AccountEJB`.
 - ▶ How this linkage is handled depends on the J2EE product in use.



Types of References



- EJB References

- Resource References
 - ▶ JDBC DataSources, JavaMail Resources etc.

- Security Role References



IBM WebSphere Application Server V4.0

J2EE Packaging and Deployment



Packaging

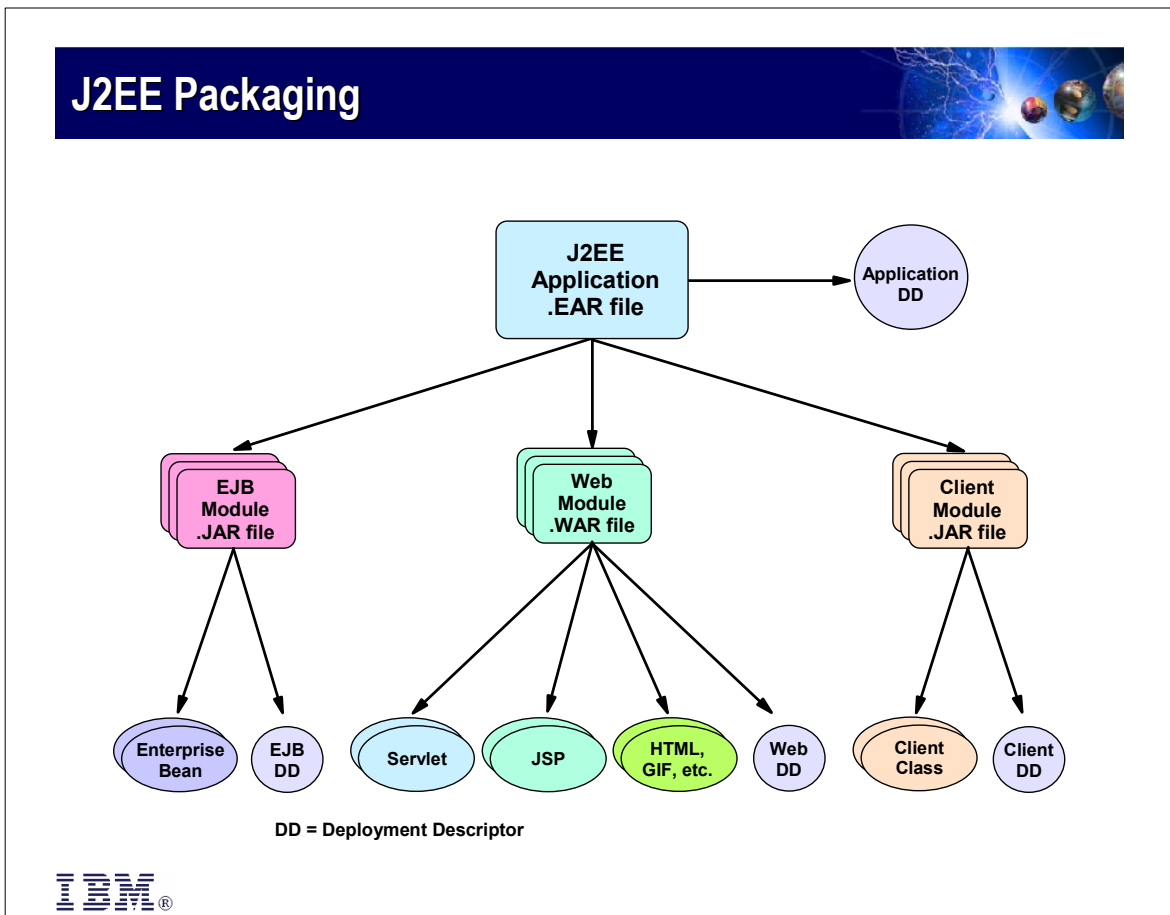


- Components go in archives
 - ▶ JAR - Java Archive
 - EJBs, Application Clients
 - ▶ WAR - Web Archive
 - JSPs, HTML, GIF, JPG, Servlet class
 - ▶ EAR - Enterprise Archive
 - JARs and WARs
- Deployment Descriptors
 - ▶ At each level -
 - EAR file - application.xml
 - WAR file - web.xml
 - EJB JAR file - ejb-jar.xml
 - Client JAR file - application-client.xml



- ▶ EJBs are packaged in a JAR file. The EJB JAR file will contain all the EJBs and a deployment descriptor.
- ▶ Application clients are also packaged in a JAR file. The Application Client JAR will contain all the Java classes of the Application Client and a deployment descriptor.
- ▶ Web Components are packaged in a WAR file. The WAR file will contain the Web components and a deployment descriptor
- ▶ The WAR files and the JAR files together constitute the EAR file. The EAR file will also contain the application deployment descriptor.

J2EE Packaging



- ▶ A J2EE application is packaged in an Enterprise Archive, a file with a .EAR extension.
 - ▶ The application has a deployment descriptor, shown in the diagram as DD, allowing configuration to a specific container's environment when deployed.
 - ▶ The application can include one or more modules.
- ▶ J2EE components are grouped in modules, and each module has its own deployment descriptor.
 - ▶ EJB modules group related EJBs in a single module, and are packaged in Java Archive (JAR) files.
 - ▶ Note that there is only one deployment descriptor for all of the EJBs in the module. Previously, in WebSphere V3.5, each Enterprise bean had its own deployment descriptor.
 - ▶ Web modules group servlet class files, JSPs, HTML files, and images. They are packaged in Web Application Archive (WAR) files.
 - ▶ Application client modules are packaged in Java Archive (JAR) files.

J2EE: Deployment Descriptor (DD)



- XML-based files describing the environment for modules and their components
 - ▶ Each module and EAR file has a deployment descriptor
- Can be automatically created and verified by WebSphere's deployment tool
 - ▶ Application Assembly Tool (AAT)
- Can be manually created or edited



- ▶ Dictates component's interaction with container

J2EE: EJB Deployment Descriptor



- **ejb-jar.xml**
 - ▶ One DD for all the EJBs in the jar
 - ▶ For each EJB:
 - home, remote and bean name
 - For Session Beans, specify session type
 - For Entity Beans, specify persistence type
 - ◆ CMP: primary key and container managed fields
 - Transaction type
 - Environment Entries
 - EJB References
 - ▶ **Common Assembly Information**
 - Specify security roles used in the EJB
 - ◆ Method-mapping (maps roles to methods)
 - For Container-managed transactions
 - ◆ maps EJB method to transaction attribute



J2EE: WAR Deployment Descriptor



■ web.xml

- ▶ Defines servlets, JSP, static resources
 - Servlet URL pattern, class/file, attributes
- ▶ Security Constraints
 - For each URL pattern assign Security Roles for methods
 - (GET, POST, etc.)
- ▶ Login configuration
- ▶ EJB and Resource references
- ▶ Security Roles for this web module



J2EE: EAR Deployment Descriptor



- application.xml
 - ▶ Defines all modules packages in the EAR file
 - EJB JARs
 - WARs
 - Application Client jars
 - ▶ Specifies defined Security Roles



J2EE Resources



- <http://java.sun.com/j2ee>
 - ▶ J2EE Specification
 - ▶ J2EE Blueprints
 - ▶ J2EE Reference Implementation
 - ▶ Java PetStore application
 - ▶ Articles and discussions on J2EE
 -
- Developing Java Enterprise Applications by Stephen Asbury and Scott R. Weiner



**Appendix:
EJB 1.1 / Servlet 2.2/
JSP 1.1**



J2EE: EJB 1.1 changes ...



■ Primitive primary keys

- ▶ In 1.0, had to encapsulate them in a primary key class.
- ▶ In 1.1, client can look up a primitive key of the type `java.lang.Integer` as follows:
 - **`accountHome.findByPrimaryKey(new Integer(5));`**
 - Primary key classes use for primitive data types is deprecated



J2EE: EJB 1.1 Code changes



- In most cases, you do not need to modify the EJB 1.0 code to deploy in an EJB 1.1 environment
- CREATION
 - ▶ For CMP Entity Bean
 - `ejbCreate` method MUST return the same type as the primary key
 - Actual value returned must be null
 - ▶ **Each `ejbCreate` must have a matching `ejbPostCreate`** (was optional in 1.0)
- SECURITY
 - ▶ `javax.security.Identity`, `getCallerIdentity()` and `isCallerInRole(Identity)` are deprecated
 - ▶ **In 1.1, use `getCallerPrincipal()` and `isCallerInRole(String)` from interface `javax.ejb.EJBContext`**





■ TRANSACTIONS

- ▶ `javax.jts.UserTransaction` interface must be renamed to `javax.transaction.UserTransaction`
- ▶ `TX_BEAN_MANAGED` for Entity Beans not allowed
 - Entity beans must let the container manage the transactions
 - Only session beans may use Bean-Managed Transaction
- ▶ Cannot have `UserTransaction` **and** `SessionSynchronization` interface at the same time for Bean-Managed Transaction
 - `SessionSynchronization` allows beans to participate in a transaction
 - Session beans that implement `SessionSynchronization` must have the transactions managed by the container
- ▶ Catch `IllegalStateException` for `javax.transaction.Transaction.commit()` method



J2EE: EJB 1.1 Code changes...



■ EXCEPTION HANDLING

- ▶ `javax.ejb.RemoteException` exception is deprecated in 1.1
- ▶ Use **`javax.ejb.EJBException`** or a more specific exception such as `javax.ejb.CreateException`.
- ▶ Rollback on System exception:
 - In EJB 1.1, **system exceptions result in transaction rollback**
 - In EJB 1.0, system exceptions do NOT result in transaction rollback
- ▶ Rollback on Application exception:
 - In EJB 1.1, **Container DOES NOT automatically rollback a transaction**
 - ◆ Must use `EJBContext.setRollbackOnly()` to indicate container to rollback
 - In EJB 1.0, Container automatically rollbacks the transaction

■ FINDER METHODS

- ▶ Finder methods must throw `FinderException`
 - VAJ currently does this, but non-VAJ users must ensure this is done



Deprecated Servlet APIs in WAS 4.0



Newly Deprecated Methods	Replacement Methods
<code>UnavailableException(Servlet servlet, String message)</code>	<code>UnavailableException(String message)</code>
<code>UnavailableException(int sec, Servlet servlet, String message)</code>	<code>UnavailableException(String message, int sec)</code>
<code>getValue(String name)</code>	<code>Object HttpSession.getAttribute(String name)</code>
<code>getValueNames()</code>	<code>Enumeration HttpSession.getAttributeNames()</code>
<code>putValue(String message, Object value)</code>	<code>void HttpSession.setAttribute(String name)</code>
<code>removeValue(String message)</code>	<code>void HttpSession.removeAttribute(String name)</code>



Deprecated Packages



Deprecated Package Names	Replacement Package Names
<code>com.ibm.servlet.personalization.sam</code>	No equivalent
<code>com.ibm.servlet.servlet.personalization.util</code>	No equivalent
<code>com.ibm.servlet.connmgr</code>	Use JDBC connection pool API
<code>com.ibm.servlet.personalization.sessiontracking</code>	<code>com.ibm.websphere.servlet.session.IBMSession</code>



Deprecated Interfaces



Deprecated Interface Names	Replacement Interface Names
<code>javax.servlet.http.HttpSessionContext</code>	No replacement. Deprecated as of Servlet API 2.1 for security reasons. This interface will be removed in a future version of this API.
<code>com.ibm.servlet.personalization.sessiontracking.IBMHttpSessionBindingListener</code>	This class is deprecated based on the design point with WebSphere Application Server 3.0 that all instances running in a cluster are "equals", hence there should be no distinguishing factor as to which instance the <code>HttpSessionBindingListener</code> processing takes place in. For release 3.0, this interface will be processed just like its base interface defined in the Servlet API <code>HttpSessionBindingListener</code>



Deprecated Classes



Deprecated Classes	Replacement Classes
<code>com.ibm.servlet.personalization.sessiontracking.IBMHttpSessionBindingEvent</code>	No replacement. This class is deprecated based on the design point with WebSphere Application Server 3.0 that all instances running in a cluster are "equals", hence there should be no distinguishing factor as to which instance the <code>HttpSessionBindingListen</code> takes place in.
<code>com.ibm.servlet.personalization.sessiontracking.IBMSessionContextImpl</code>	No replacement. This class has been deprecated due to the deprecation of <code>javax.servlet.http.HttpSessionContext</code> . All public methods in this class will either return null or false (for "get'ers") or have no effect.
<code>com.ibm.servlet.personalization.sessiontracking.IBMSessionData</code>	Use <code>com.ibm.websphere.servlet.session.IBMSession</code> . This class is deprecated as of WAS 3.0. All <code>HttpSession</code> objects returned to servlets will still be an instance of this class. All public methods in this class will either return null (in the case of a "get'er") or have no effect (in the case of a "set'er").
<code>com.ibm.servlet.connmgr.IBMConnMgrUtil</code>	JDBC connection pool API
<code>com.ibm.servlet.connmgr.IBMConnMgr</code>	JDBC connection pool API
<code>com.ibm.servlet.connmgr.IBMConnection</code>	JDBC connection pool API
<code>com.ibm.servlet.connmgr.IBMJdbcConn</code>	JDBC connection pool API
<code>com.ibm.servlet.connmgr.IBMConnPoolSpec</code>	JDBC connection pool API
<code>com.ibm.servlet.connmgr.IBMJdbcConnSpec</code>	JDBC connection pool API



Deprecated Constructors



Deprecated Constructor Names	Replacement Constructor Names
<code>javax.servlet.UnavailableException(int, Servlet, String)</code>	As of Java Servlet API 2.2, use <code>UnavailableException(String, int)</code> instead.
<code>avax.servlet.UnavailableException(Servlet, String)</code>	As of Java Servlet API 2.2, use <code>UnavailableException(String)</code> instead.
<code>com.ibm.servlet.personalization.sessiontracking.IBMHttpSessionBindingEvent(HttpSession, String)</code>	No replacement. See its deprecated class <code>com.ibm.servlet.personalization.sessiontracking.IBMHttpSessionBindingEvent</code> in the deprecated classes table.



Deprecated JSP Tags



Deprecated JSP .91 Tags	Replacement JSP 1.1 Tags
<BEAN></BEAN>	<jsp:useBean />
<SERVLET></SERVLET>	<jsp:include />
<PARAM>	<jsp:setProperty /> when following <BEAN> tag <jsp:param /> when following <SERVLET> tag
<REPEAT>	<tsx:repeat>
</REPEAT>	</tsx:repeat>
<REPEATGROUP>	<tsx:repeat>
</REPEATGROUP>	</tsx:repeat>
<INSERT></INSERT>	<tsx:getProperty /> when following <REPEAT>, <USER>, or <PASSWORD> tags
\$()	<tsx:getProperty />
<SCRIPT ></SCRIPT>	<%! %>
<%@ language=	<%@ page language=
<%@ import=	<%@ page import=
<%@ content_type=	<%@ page contentType=
<%@ extends=	<%@ page extends=
<%@ method=	No equivalent JSP tag
<%@ implements=	No equivalent JSP tag



Deprecated JSP Tags (continued)



Deprecated JSP .91 Tags	Replacement JSP 1.1 Tags
<DBCONNECT>	<tsx:dbconnect>
</DBCONNECT>	</tsx:dbconnect>
<USERID>	<tsx:userid>
</USERID>	</tsx:userid>
<PASSWORD>	<tsx:password>
</PASSWORD>	</tsx:password>
<DBQUERY>	<tsx:dbquery>
</DBQUERY>	</tsx:dbquery>
<DBMODIFY>	<tsx:dbmodify>
</DBMODIFY>	</tsx:dbmodify>
<!--#include -->	<%@ include %>
<!--#config -->	No equivalent JSP tag
<!--#echo -->	No equivalent JSP tag
<!--#exec -->	No equivalent JSP tag
<!--#fsize -->	No equivalent JSP tag
<!--#lastmod -->	No equivalent JSP tag

