**EJB 3.0**

Enterprise JavaBeans (EJB) technology is the server-side component architecture for Java Platform, Enterprise Edition (Java EE). EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.

EJB is a subset of J2EE or JEE and a J2EE/JEE server must include an EJB container.

J2EE/JEE is a Specification. It specifies the rules of engagement that people must agree on when writing enterprise software.

The EJB container is a piece of software that implements the EJB Specification. EJB container provides the bean with various services such as life cycle management, security, transaction management and much more. EJB container provides system-level services to enterprise beans and allows the bean developer can concentrate on solving business problems. System-level services 🡪 Transaction management, security authorization, etc.

An *enterprise bean* is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application.

An *enterprise bean* must run under the control of an EJB container.

Components are reusable chunks of functionality yon can modify for different application without touching the java source code.

EJB lets you focus on the business logic for your business and leave the underlying services (transaction, networking, security, etc) to the EJB vendor.

EJB vendors - BEA Systems (BEA WebLogic Server), IBM (WebSphere Application Server), Oracle (Oracle Application Server), SIEMENS (Siemens Enterprise Application Server), Sybase (Sybase Enterprise Application Server)

**Advantages of Using EJB’s**

EJB servers give you a bunch of services, so that you don’t have to write them yourself:

* Transaction Management
* Security
* Concurrency
* Networking
* Resource Management
* Persistence
* Messaging
* Deploy time Customization

Enterprise beans are portable – not just to different JVM’s, but to different EJB servers.

One of the crucial benefits of EJB is WODA (write-once-deploy-anywhere). WODA means you have to learn only one standard API rather than proprietary vendor-specific APIs.

In EJB3.0, the container provides the services to the EJB components in a new way: metadata annotations/deployment annotations.

***Annotations*** essentially allow us to “attach” additional information (officially called attributes) to a Java class, interface, method, or variable.

The additional information conveyed by annotations can be used by a development environment like Eclipse, the Java compiler, a deployment tool, a persistence provider like Hibernate, or a runtime environment like the Java EE container.

Annotations can be used to specify bean types, different attributes such as transaction or security settings, O-R mapping and injection of environment or resource references.

Annotations begin with an @ sign followed by the annotation name which in turn is followed by annotation data (if any).

In EJB, metadata annotations dramatically simplify development and testing of applications, without having to depend on an external XML configuration file.



A *deployment descriptor* is simply an XML file that contains application configuration information.

Every deployment unit in Java EE can have a deployment descriptor that describes its contents and environment.

There are three types of EJB components:

1. Session beans
2. Message-driven beans
3. Entity beans.

*An entity bean IS something.*

*A Session bean DOES something.*

**Session Beans**

A session bean typically represents a *process*.

There are two subtypes of session beans—*stateful session beans* and *stateless session beans*.

A stateful bean can remember conversational state between method calls, while a stateless bean won’t remember anything about the client between method invocations.

The phrase “Conversational State” really means “client-specific state”.

**The Bean lifecycle**

**Stateful session bean**

* Bean **creation** (when client wants a bean)
* Bean **use** (when the client calls a business method)
* Bean **passivation** (the bean is put to sleep to conserve resources)
* Bean **activation** (the bean wakes up to service a business method from the client)
* Bean **removal** (when the client is finished with the bean or the bean times out)

**Stateless session bean**

* Bean **creation** (when the container wants to make a bean)
* Bean **use** (when the client calls a business method)
* Bean **removal** (when the container decides there are too many beans in the pool)

**Stateless session bean callbacks methods**

PostConstruct - is invoked when the bean is first created, after any dependency injection is done.

PreDestroy - is invoked when the bean is removed from the pool or destroyed.

**Stateful session bean callbacks methods**

PostConstruct - is invoked when the bean is first created, after any dependency injection is done.

PreDestroy - is invoked when the bean is removed from the pool or destroyed. It will happen before any @Remove annotated method is invoked.

PostActivate

PrePassivate

**Entity Beans**

Entity beans are an OO way of looking at data in a persistence store.

Represents a thing in a persistent store.

An instance of an entity bean represents a row in a table.

Container-Managed Persistence (CMP) means the container takes care of all the database access code for synchronization, including adding and deleting entities (records / rows in the database).

Bean- Managed Persistence (BMP) means YOU write the database access code (the JDBC statements), for when the container tells you its time to go to the database.

**EJB 3.0 callback methods**

Entity bean callbacks

PrePersist - Is invoked right before the entity is created in the database. Will cascade to all entities to which this operation is cascaded.

PostPersist - Is invoked right after the entity is created in the database. Will cascade to all entities to which this operation is cascaded.

PreRemove - Is invoked right before the entity is deleted in the database. Will cascade to all entities to which this operation is cascaded.

PostRemove - Is invoked right after the entity is deleted in the database. Will cascade to all entities to which this operation is cascaded.

PreUpdate - Takes place right before the database is updated.

PostUpdate - Takes place immediately after the database has been updated.

PostLoad - Takes place right after data has been loaded from the database and associated with the entity

**Transaction**

An EJB Transaction is an atomic unit of work.

**Transaction Attributes**

Required

RequiresNew

Mandatory

Supports

NotSupported

Never

You can apply the @TransactionAttribute annotation at the class-level to specify the default transaction attribute for all business methods of the enterprise bean.

You can apply this annotation at the method-level to specify the transaction attribute for that method.

Applying the annotation at the method-level overrides the class-level annotation (if any) for that method.

**EntityManager Interface**

An EntityManager instance is associated with a persistence context.

A *persistence context is a set of entity instances* in which for any persistent entity identity there is a unique entity instance.

Within the persistence context, the entity instances and their lifecycle are managed.

The set of entities that can be managed by a given EntityManager instance is defined by a persistence unit.

A persistence unit defines the set of all classes that are related or grouped by the application, and which must be co-located in their mapping to a single database.

A persistence unit defines a set of all entity classes that are managed by EntityManager instances in an application.

Persistence units are defined by the persistence.xml configuration file.

Container-Managed Entity Managers

Application-Managed Entity Managers

@PersistenceUnit

EntityManagerFactory emf;

Then, obtain an EntityManager from the EntityManagerFactory instance:

EntityManager em = emf.createEntityManager();