Definitions and some examples

# EJB

By: Hamid Mosavi-Porasl

A typical EJB consists of at least four files, i.e. home, remote, implementation and deployment.

EJB consists of 3 different kinds of beans, i.e. entity, session and Message driven beans.

An ***entity bean*** might represent a customer, a piece of equipment, an item in inventory, or even a place. In other words, entity beans model real-world objects; these objects are usually persistent records in some kind of database.

***Session bean*** are an extension of the client application and are responsible for managing processes or tasks.

**Message-driven beans** are responsible for coordinating tasks involving other session and entity beans. The major difference between a message-driven bean and a session bean is how they are accessed. While a session bean provides a remote interface that defines which methods can be invoked, a message-driven bean doesn't. Instead, the message driven bean subscribes or listens to a specific asynchronous messages to which it responds by processing the message and managing the activities of other beans in response to those messages.

An entity bean has persistent state; the session and message-driven beans model interactions but don't have persistent state.

### Asynchronous Messaging

An asynchronous messaging system allows tow or more applications to exchange information in the form of message. A message, in this case, is self-contained package of business data and network routing headers. Messages are transmitted from one application to another one on a network using message-oriented middleware (MOM). In all modern enterprise messaging systems, applications exchange messages through virtual channels called *destinations*. When sending a message, it's addressed to a destination, not a specific application.

### Message-Driven Beans

Message-driven bean is a kind of standard JMS bean. It can receive and send messages. It can receive and send asynchronous JMS messages, because it is co-located with other kind of RMI beans (entity and session beans). It can also interact with RMI components. Like other RMI based components, message-driven beans are considered business objects

## Architectural Overview

Enterprise Java Bean is a component model for component transaction monitors.

### Classes and Interfaces

To implement entity and session enterprise beans, you need to define the

1. ***Component Interfaces:***
    - ***Remote interface***

The remote interface for an enterprise bean defines the bean's business methods that can be accessed from applications outside the EJB container. The remote interface extends ***javax.ejb.EJBObject***, which in turn extends ***java.rmi.Remote***. The remote interface is one of the bean's component interfaces and is used by session and entity beans in conjunction with the remote home interface.

- ***Remote Home interface***

The home interface defines the bean's life cycle methods that can be accessed from applications outside the EJB container: the life-cycle methods for creating new beans, removing beans, and finding beans. The home interface extends ***javax.ejb.EJBHome***, which extends ***java.rmi.Remote***. This interface is used by entity and session beans.

- ***Local Interface*** (EJB2.0)

The local interface for an enterprise bean defines the bean's business methods that can be used by other co-located in the same EJB container. This interface extends ***javax.ejb.EJBLocalObject***. This interface is used by entity and session beans.

- ***Local Home Interface*** (EJB2.0)

The home interface defines the life cycle methods that can be used by other beans co-located in the same EJB container. The local home interface extends ***javax.ejb.EJBLocalHome.*** This interface is used by entity and session beans.

## *2. A bean class:*

The session and entity bean classes actually implement the bean's business and life-cycle methods. Note that the bean class for session and entity beans usually doesn't implement any of the bean's component interfaces directly. But they should contain methods matching the signatures of the methods defined in the remote and local interfaces and must have method corresponding to some of the methods in both the remote and local home interfaces. An ***entity bean*** must implement ***javax.ejb.EntityBean***; a ***Session bean*** must implement ***javax.ejb.sessionBean***. The ***EntityBean*** and ***SessionBean*** extend ***javax.ejb.EnterpriseBean***.

The ***message-driven*** bean in EJB2.0 doesn't use any or the component interfaces, because it is never accessed by method calls from other applications or beans. Instead, the message-driven bean contains a single methos, ***onMessage(),*** which is called by the container when a new message arrives. The message-driven bean implements the ***javax.ejb.MessageDrivenBean*** and ***javax.jms.MessageListener*** interfaces. The JMS MessageListener interface is what makes a message-driven bean specific to JMS, instead of some other protocol. EJB 2.0 requires the use of JMS. ***MessageDrivenBean***, like ***EntityBean*** and ***SessionBean***, extends the ***javax.ejb.EnterpriseBean*** interface.

## *3. A primary key*

The primary key is a very simple class that provides a pointer into the database. Only entity bean need a primary key; the only requirement for this class is that it implements ***java.io.Serializable***.

## Deployment Descriptors and JAR files

Much of the information about how beans are managed are managed at runtime isn't addressed in the interfaces and classes. Deployment descriptors serve a function very similar to property files. They allow us to customize behavior of software at runtime without having to change the software itself.

When a bean class and its interface have been defined, a deployment descriptor for the bean is created and populated with data about the bean. When a bean class and its interfaces have been defined, a deployment descriptor for the bean is created and populated with data about the bean. After that developer has set all the properties for a bean, the deployment descriptor is complete and saved to a file the bean can be packaged in a JAR file for deployment.

The ***deployment descriptors*** help the development tools to add beans to the EJB container once the bean is deployed, the properties described in the deployment descriptor will continue to be used to tell the EJB container how to manage the bean.

## Deploying a bean

The EJB objects and EJB homes are generated during he deployment process.