EJB

Introduction

What is Enterprise JavaBeans?

Enterprise JavaBeans (EJB) is Sun Microsystems' specification for a distributed object system similar to CORBA and Microsoft Transaction Server, but based on the Java platform. EJB specifies how developers should build components that can be accessed remotely and how EJB vendors should support those components. EJB components, called enterprise beans, automatically handle transactions, persistence, and authorization security, so that the developer can focus on the business logic.

What is an enterprise bean?

An enterprise bean is a server-side component - defined in the Java technology - which adheres to the Enterprise JavaBeans server-side component model. A server-side component is business object that can be accessed remotely. Many server-side component models exist: CORBA specifies CORBA objects; Microsoft Transaction Server (MTS) defines COM/DCOM; and EJB specifies enterprise beans.

Enterprise beans can be developed to represent business concepts like Employee, Order, Travel Agent, etc. Enterprise beans can be assembled into applications that solve enterprise business problems.

Benefits of EJB

The following are the list of benefits of EJB:

- EJB applications are easy to develop because the application developer can concentrate on the business logic. At the same time, the developer uses the services provided by the EJB container, such as transactions and connection pooling.
- EJBs are components. Chances are good that there are EJB vendors who sell components that encapsulate the functionality that you need. By purchasing a third-party's EJBs, you can avoid needing to develop your own beans, which means your application development is more rapid. The EJB specification makes sure that beans developed by others can be used in your application.
- There is a clear separation of labor in the development, deployment, and administration of an EJB application. This makes the development and deployment process even faster.
- The EJB container manages transactions, state management details, multithreading, connection pooling, and other low-level APIs without you, the developer, having to understand them.
- The EJB container provides security for the applications.
- The EJB architecture is compatible with other Java APIs.

Restriction on EJB

What restrictions are imposed on an EJB? or What can't an EJB do?

- An enterprise Bean must not use read/write static fields. Using read-only static fields is allowed. Therefore, it is recommended that all static fields in the enterprise bean class be declared as final.
- An enterprise Bean must not use the AWT functionality to attempt to output information to a display, or to input information from a keyboard.
- An enterprise bean must not use the java.io package to attempt to access files and directories in the file system.
- The enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread; or to change a thread's priority or name. The enterprise bean must not attempt to manage thread groups.
- The enterprise bean must not attempt to directly read or write a file descriptor.
- The enterprise bean must not attempt to load a native library.
- The enterprise bean must not attempt to define a class in a package.

Types of EJB

How many enterprise beans?

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

What are Entity Bean and Session Bean?

Entity Bean is a Java class which implements an Enterprise Bean interface and provides the implementation of the business methods. There are two types: Container Managed Persistence (CMP) and Bean-Managed Persistence (BMP).

Session Bean is used to represent a workflow on behalf of a client. There are two types: Stateless and Stateful. Stateless bean is the simplest bean. It doesn't maintain any conversational state with clients between method invocations. Stateful bean maintains state between invocations.

What is message-driven bean?

A message-driven bean combines features of a session bean and a Java Message Service (JMS) message listener, allowing a business component to receive JMS. A message-driven bean enables asynchronous clients to access the business logic in the EJB tier.

Timer Service

Timer Service is a mechanism using which scheduled application can be build. For example, salary slip generation on 1st of every month. EJB 3.0 specification has specified @Timeout annotation which helps in programming the ejb service in a stateless or message driven bean. EJB Container calls the method which is annotated by @Timeout.
EJB Timer Service is a service provided by Ejb container which helps to create timer and to schedule callback when timer expires.

Using the EJB timer service

Enterprise applications often require certain business logic to run at specific times or at specific intervals that are triggered by time notifications. With the release of the EJB 2.1 specification, developers now have the ability to incorporate timers into their EJBs. Using the new timer service allows you to concentrate on the business logic instead of trying to develop a robust timing mechanism.

What is the advantage of using Entity bean for database operations, over directly using JDBC API to do database operations? When would I use one over the other?

Entity Beans actually represents the data in a database. It is not that Entity Beans replaces JDBC API. There are two types of Entity Beans Container Managed and Bean Managed. In Container Managed Entity Bean - Whenever the instance of the bean is created the container automatically retrieves the data from the DB/Persistence storage and assigns to the object variables in bean for user to manipulate or use them. For this the developer needs to map the fields in the database to the variables in deployment descriptor files (which varies for each vendor). In the Bean Managed Entity Bean - The developer has to specifically make connection, retrieve values, assign them to the objects in the ejbLoad() which will be called by the container when it instantiates a bean object. Similarly in the ejbStore() the container saves the object values back the persistence storage. ejbLoad and ejbStore are callback methods and can be only invoked by the container. Apart from this, when you use Entity beans you don't need to worry about database transaction handling, database connection pooling etc. which are taken care by the ejb container.

What is the difference between session and entity beans? When should I use one or the other?

An entity bean represents persistent global data from the database; a session bean represents transient user-specific data that will die when the user disconnects (ends his session). Generally, the session beans implement business methods (e.g. Bank.transferFunds) that call entity beans (e.g. Account.deposit, Account.withdraw)

Are Enterprise JavaBeans and JavaBeans the same thing?

Enterprise JavaBeans and JavaBeans are not the same thing; nor is one an extension of the other. They are both component models, based on Java, and created by Sun Microsystems, but their purpose and packages (base types and interfaces) are completely different.

JavaBeans
The original JavaBeans specification is based on the java.beans package which is a standard package in the JDK. Components built on the JavaBeans specification are ***intra*****process** components that live in one address space and are typically used for Graphical User Interface (GUI) as visual widgets like buttons, tables, HTML viewers, etc.

Enterprise JavaBeans

The EJB specification is based on the javax.ejb package, which is a standard extension package. Components built on the EJB specification are ***inter*****process** components that live in multiple address spaces as distributed object. These components are used as transactional business objects that are accessed as remote objects.

How is persistence implemented in enterprise beans?

Persistence in EJB is taken care of in two ways, depending on how you implement your beans: container managed persistence (CMP) or bean managed persistence (BMP).

For CMP, the EJB container which your beans run under takes care of the persistence of the fields you have declared to be persisted with the database - this declaration is in the deployment descriptor. So, anytime you modify a field in a CMP bean, as soon as the method you have executed is finished, the new data is persisted to the database by the container.

For BMP, the EJB bean developer is responsible for defining the persistence routines in the proper places in the bean, for instance, the ejbCreate(), ejbStore(), ejbRemove() methods would be developed by the bean developer to make calls to the database. The container is responsible, in BMP, to call the appropriate method on the bean. So, if the bean is being looked up, when the create() method is called on the Home interface, then the container is responsible for calling the ejbCreate() method in the bean, which should have functionality inside for going to the database and looking up the data.

What is the difference between a Server, a Container, and a Connector?

To keep things (very) simple:

An EJB server is an application, usually a product such as BEA WebLogic, that provides (or should provide) for concurrent client connections and manages system resources such as threads, processes, memory, database connections, network connections, etc.

An EJB container runs inside (or within) an EJB server, and provides deployed EJB beans with transaction and security management, etc. The EJB container insulates an EJB bean from the specifics of an underlying EJB server by providing a simple, standard API between the EJB bean and its container.

A Connector provides the ability for any Enterprise Information System (EIS) to plug into any EJB server which supports the Connector architecture.

What's the difference between container-managed and bean-managed persistence?

In container-managed persistence, entity bean data is automatically maintained by the container using a mechanism of its choosing. For example, a container implemented on top of an RDBMS may manage persistence by storing each bean's data as a row in a table. Or, the container may use Java programming language serialization for persistence. When a bean chooses to have its persistence container managed, it specifies which of its fields are to be retained.

In bean-managed persistence, the bean is entirely responsible for storing and retrieving its instance data. The EntityBean interface provides methods for the container to notify an instance when it needs to store or retrieve its data.

What is the difference between Message Driven Beans and Stateless Session beans?

In several ways, the dynamic creation and allocation of message-driven bean instances mimics the behavior of stateless session EJB instances, which exist only for the duration of a particular method call. However, message-driven beans are different from stateless session EJBs (and other types of EJBs) in several significant ways: Message-driven beans process multiple JMS messages asynchronously, rather than processing a serialized sequence of method calls. Message-driven beans have no home or remote interface, and therefore cannot be directly accessed by internal or external clients. Clients interact with message-driven beans only indirectly, by sending a message to a JMS Queue or Topic. Only the container directly interacts with a message-driven bean by creating bean instances and passing JMS messages to those instances as necessary. The Container maintains the entire lifecycle of a message-driven bean; instances cannot be created or removed as a result of client requests or other API calls.

What is a container?

Enterprise beans are software components that run in a special environment called an EJB container. The container hosts and manages an enterprise bean in the same manner that a Java WebServer hosts a Servlet or an HTML browser hosts a Java applet. An enterprise bean cannot function outside of an EJB container. The EJB container manages every aspect of an enterprise bean at run time including remote access to the bean, security, persistence, transactions, concurrency, and access to and pooling of resources.

What are Timers?

The EJB Timer service allows you to specify a method (appropriately called the *timeout method*) that is automatically invoked after a specified interval of time. The container invokes the timeout method on your behalf when the time interval you specify elapses. As we will see, we can use the Timer service to register for callbacks triggered once at a specific time or at regular intervals.
We can only use Timers in Stateless Session Beans and Message Driven Beans because of their asynchronous, stateless nature. However, unlike Stateless Session Beans and MDBs, Timers are persistent and can survive a container crash or restart. Timers are also *transactional*, that is, a transaction failure in a timeout method rolls back the actions taken by the Timer.

Difference between RMI and EJB

Both RMI and EJB, provides services to access an object running in another JVM (known as remote object) from another JVM. The differences between RMI and EJB are given below:

RMI	EJB
In RMI, middleware services such as security, transaction management, object pooling etc. need to be done by the java programmer.	In EJB, middleware services are provided by EJB Container automatically.
RMI is not a server-side component. It is not required to be deployed on the server.	EJB is a server-side component, it is required to be deployed on the server.
RMI is built on the top of socket programming.	EJB technology is built on the top of RMI.

Enterprise Java Beans	**!= Java Beans**
EJBs need a Container	JBs do not need a container
EJBs are deployable components	JBs are development components
EJBs are assembled to form a complete Appl	JBs are Classes with no argument constructor
EJBs are based on RMI IIOP and JNDI Technologies	JBs have a *get* and a *set* method on them

Stateful	**Stateless**
A stateful bean contains a conversational state that is retained across method calls and transactions.	A stateless bean does not have any state between calls to its methods.
The create method takes arguments e.g. create(String id) , create(int I , String id) There can be one or more arguments in a create method	The create method does not take arguments e.g create() There can be no arguments in a create method
e.g An EJB that unzips 100 bytes of data An EJB that checks to see if a stock symbol is valid	e.g An EJB that books a flight and rents a car at a travel agent's web site.