

**Java Institute for Advanced Technology**

UNIT NAME= Business Component Development

(BCD)

NAME- G.M.M.A KALPA SANDARUWAN MOHOTTI

SCN NO- 177590797

NIC- 962512127v

BRANCH- Kandy



**Q1. EJB provides a server-side platform for developing distributed business applications, which designed for Enterprise beans are used in large-scale business applications and systems. Do you agree? Critically Evaluate your Answer with EJB Architecture diagram.**

**Yes, I agree,**

When we use EJB it provides a server-side platform for developing distributed business applications. EJB means Enterprise Java Beans. These enterprise java beans used in large-scale business applications and systems.

EJB 1.0 was released in 1998. The most current release, EJB 3.2.3, has been adopted for inclusion in Jakarta EE, where it will be renamed Jakarta Enterprise Beans.

EJB is a server-side software element that can summarize the business logic of an application. EJB has a web container. it provides the runtime environment for web-related software components, it provides computer security, Java servlet lifecycle management, transaction processing, and other web services.

This EJB platform uses to develop scalable, robust, and secured enterprise applications in java. When we develop distributed application we can use EJB.

**What is distributed application,**

A distributed application consists of one or more local or remote clients that communicate with one or more servers on several machines linked through a network. When we use this type of application we can conducts business operations from any place in the world. This distributed application has main two parts. There are client-side and server-side.

from the client-side, we can access data from the server or cloud environment. we can see from the server-side process the data and send it back to the client. These distributed applications allow multiple users to access at once. In industry, many developers, IT professionals choose to store distributed apps in the cloud environment. Because when we use a cloud environment we can identify cloud elasticity and scalability, as well as its ability to handle large applications or workloads.

EJB and distributed applications

This application type has main three parts. these are the presentation layer, business layer, database layer. J2EE applications identify from the following phases.

Unified application model across tiers with enterprise

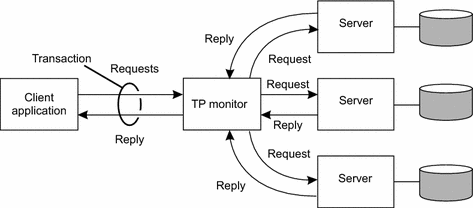
beans

Simplified response and request mechanism with JSP

pages and servlets

Enterprise application integration with message-driven

beans and JMS, JTA, and JNDI



In Java EE architecture, modern enterprise applications have a number of layers. It depends on the responsibility of the application.

mainly we can identify three layers. these are

* Presentation layer
* Business layer
* Database layer

**Presentation layer-**

This layer responsible for presenting the user interface and handling interactions with the end user. This presentation layer has main two parts. Presentation layer communicate with business logic layer. This layer is very important because client can access our project by this layer.

* **Client layer-**

The user accesses the application through the client layer. as an example, web browser, applet, Java application client,

* **Web layer-**

This layer includes Servlet, JSP pages, JSF components.

**Business Logic Layer-**

This layer has main two parts.

* **Business logic layer**

This layer responsible for presenting a user interface and handling interactions with the end-users. This layer contains business rules and other business functions are encapsulated within this layer. This business layer includes main three components.

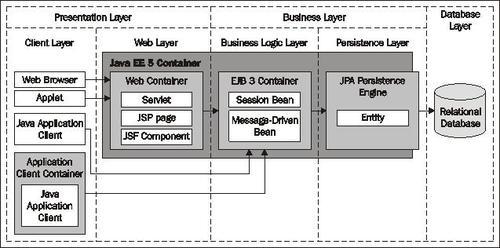
1. Session bean
2. Entity bean
3. Message-Driven bean

* **Persistence Layer**

This layer includes the JPA persistence engine.

**Database Layer-**

This layer responsible for the storage of business data. Generally, a relational database management system is used for this layer. Simulation Of real-world entities along with the business flow of the organization is also carried out at this.



**Q2. What is the suitable Session bean type (stateful, stateless, singleton) for developing a shopping cart facility for an online e-commerce system? Compare and contrast your implementation among other bean types.**

We can identify three main session bean types. these are stateful, stateless, singleton.

When we going to develop a shopping cart we should use a stateful bean.

**What is a session bean?**

A session bean is an encapsulated business logic. It can be invoked by local, remote, web service clients. These session beans have a lifecycle. it maintains by the application server. (EJB container). There are three main session bean types. There is a stateless, stateful, singleton. These session beans are mainly used when we want to wish to give control to a client, but allow them to execute something on a server. When a client or user requests a session bean, a separate instance of the bean is created and provided to the caller. A number of method calls from the bean create a conversation between the client and the bean. A session bean represents work performed by single a user and holds the state between method calls.

session beans are

* transient functions.
* it represents a "conversational" state.
* typically one per request
* data is non-persistent
* Lifetime is limited by the client’s: once the

client exits, the session bean, and data are

gone.

* Simple and easy to program.
* Light-weight.

**How session bean differs from other bean types?**

It mainly depends on the scope of their life. because session bean and other bean types have a different lifetime. as its instance is comparatively a short-lived object. Session beans have a lifetime equivalent of a session or client, calling the instance of a session bean.

**Stateful session bean-**

We can define this session bean using @Stateful annotation.

which preserves the conversational state with the client. This type of session bean as per its name keeps the associated client state in its instance variables. EJB container creates a separate stateful session bean to process the client's request. After the request scope is over, stateful session bean is destroyed. This session bean type is used to develop a shopping cart because from using this we can preserve the conversational state with the client.

Lifecycle of stateful bean

The number of instances is equal to the number of clients. it's mean if 500 clients connect to the server, the EJB container will create 500 instance, of the stateful bean. This could be a big problem for the server. but that problem can handle using activation and passivation. if 500 clients are connected to the server, but only 100 of them are actually calling methods. In this case, the server passivates the remaining 400 client objects into the virtual memory. If any passivate client requests, the new instance is activated.

**Stateless session bean-**

We can define this session bean using @Stateless annotation.

This session bean also the type of enterprise bean. This session bean type uses to perform independent operations. This stateless session bean as per its name does not have any associated client state, but it may preserve its instance state. For this beans EJB container creates a pool with few stateless bean objects. We can use these objects to process the client's request. Because of the pool, instance variable values are not guaranteed to be the same across lookups/method calls.

Lifecycle of stateless bean

When nobody is calling a session bean, there are no instances. This type of session bean instance is created when a client is connected and demands. When a new request comes EJB container decides to create or not to instance. It depends on the number of users. If the count of users is increasing the EJB server automatically creates a new instance.

Stateless bean has four callback methods.

* post construct
* pre destroy
* init
* remove

**Singleton session bean-**

We can define this session bean using @Singleton annotation.

These types of session beans are designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients. These beans maintain their state between client invocations. But these beans not required to maintain their state across server crashes or shutdowns. We can give a guarantee that there is at most one instance in the application. These beans are also the same as stateless beans.

**Q3. Dependency Injection (DI) makes developers life easier to implement session beans with**

**@EJB annotation, but InitialContext.Lookup() is also important since DI limited to Instance**

**variables, do you agree? Take a sample scenario and elaborate your answer with critical**

**analysis.**

Yes, I agree, these two types make developers' life easier.

There are two ways to obtain an EJB instance.

* Dependency injection in servlets and EJBs via the @EJB annotation
* JNDI lookup via Context.lookup

The client of an enterprise bean obtains a reference to an instance of an enterprise bean through either dependency injection, using Java programming language annotations, or JNDI lookup, using the Java Naming and Directory Interface syntax to find the enterprise bean instance.

Dependency injection is the most easiest way to obtain an enterprise bean. Clients that run within a Java EE server management environment, JAX-RS web services, other enterprise beans, java EE applications or Java server faces web applications support dependancy injection using the javax.ejb.EJB annotation.

When applications run outside a Java EE server management environment , such as java SE applications, must perform an explicit lookup. JNDI supports a global syntax for identifying java EE components to simplify this explicit lookup.

These two types are made developer life easier. using this development method developers can call the EJB easily.

**What is Dependency Injection?**

In software engineering, dependency injection is a technique. In this technique when an object receives other objects that it depends these objects are called dependencies. The "injection" refers to the passing of a dependency into the object that would use it.

EJB 3.0 provides the following annotations.

These are the annotations for dependency injection.

* @EJB- used to inject another EJB reference.
* @Resource- This used to inject data source or singleton service like session context, time service, etc.

Dependency Injections are

* works with servlets and EJBs only
* convenient syntax
* container independent

**Example-**

How to use @EJB annotation

public class LibraryMessageBean implements MessageListener {

LibraryPersistentBeanRemote libraryBean;

//dependency injection on method.

@EJB(beanName="com.tutorialspoint.stateless.LibraryPersistentBean")

public void setLibraryPersistentBean(

LibraryPersistentBeanRemote libraryBean)

{

this.libraryBean = libraryBean;

}

...

}

How to use @Resource annotation

public class LibraryMessageBean implements MessageListener {

@Resource

private MessageDrivenContext mdctx;

...

}

**What is JNDI lookup via Context.lookup.**

public class InitialContext extends Object implements Context. This class is starting context for performing naming operations. All naming operations are relative to a context. The initial context implements the Context interface and provides the starting point for the resolution of names. The Java RMI and Java EE APIs use the JNDI API to look up objects in a network. These API provide mechanisms to bind an object to a name.

Context.lookup are

* can instantiate different implementations of the EJB interface programatically at run time.
* works from anywhere - e.g. POJOs.
* depends on naming convention of container

**Example -**

try {

Context context = new InitialContext();

TopicConnectionFactory factory = (TopicConnectionFactory) context.lookup("java:comp/env/mychatappFactory");

Connection connection = factory.createConnection();

Topic topic = (javax.jms.Topic) context.lookup("java:comp/env/mychatapp");

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

MessageConsumer consumer = session.createConsumer(topic);

MessageListenerImpl messageListener = new MessageListenerImpl();

consumer.setMessageListener(messageListener);

connection.start();

connection.start();

} catch (Exception e) {

e.printStackTrace();

}

**Different between these two**

These two types achieve the same result. It's more a matter of coupling. When we use @ EJB annotation, we achieve loose coupling and it's easier to mock and test. When we use direct lookup, you depend on the initial context. it may be inconvenient sometimes.

lookup does not work everywhere. For instance in Glassfish, a lookup on a local EJB from a POJO will work only if has been "imported" previously with @EJBs(...) on one of the session beans that use the POJO.

**Conclusion**

After evaluate all the information I think you can understand Dependency Injection (DI) and InitialContext.Lookup() makes developers life easier.