**Terminologies**

3 types of beans - entity beans, session beans(stateful and stateless), Message driven beans

EJB client is the one who have the interfaces to interact with the ejb beans

EJB local vs EJB Remote- if EJB client app and EJB beans are in same JVM then we can use EJB local

if EJB client and EJB main beans are in different JVM then , we should use EJB Remote interface (EJB Remote is best that EJB local )

1) we create EJB and we will maintain in different JVM app server and we can re-use those beans - this is like create REST producer

EJB (like REST producer)

EJB client (like REST consumer)

Stateful Session Beans

These beans are designed to maintain conversational state/user interaction with a client.

all the client state/ clients interaction will be maintained in his particular session bean global variable

ex:- shopping cart app, what ever uses browses / which ever product he see all those will be saved into his stateful bean

For 100 application users to maintain state using stateful session beans, the EJB container will create up to 100 stateful beans.

this stateful session beans are like prototype scoped beans- will be created 1 per user

Each client interaction can modify the bean's internal state.

The container associates a unique instance of the stateful bean with each client session.

**EJB Remote vs EJB local Bean**

These remote beans present in App server can be accessible even from different JVM also

If the EJB are local then, those beans (EJB stateful or stateless beans) can be accessible only from same JVM / same app server

|  |  |
| --- | --- |
| EJB Remote | EJB local bean |
| Designed for access by clients running in a different Java Virtual Machine (JVM), which could be on a separate server   Invocations involve network communication, which introduces overhead.   * uses the @Remote annotation. * Applications requiring services from a separate server. * Building distributed systems where components reside on different machines. | Designed for access by clients running within the same JVM as the EJB  If u want to access the ejb from same JVM/ same app server then prefer making EJB as EJB local bean   * Intended for interactions between components within the same application server. * Invocations are more efficient, as they involve direct method calls within the same JVM. * Avoids network overhead and serialization. * uses the @Local annotation. * Interactions between EJBs and other components (e.g., servlets) within the same application. * Optimizing performance for intra-application communication. |

|  |  |
| --- | --- |
| import javax.ejb.Stateless;  @Stateless  public class CurrencyConverterBean implements CurrencyConverter {  @Override  public double convert(double amount, String fromCurrency, String toCurrency) {  // Simple example (replace with actual conversion logic)  if (fromCurrency.equals("USD") && toCurrency.equals("EUR")) {  return amount \* 0.9; // Example conversion rate  } else {  return 0.0; // Handle other conversions  }  }  } | when u mark as @Stateless, EJBcontainer will create the beans and keep those bean in JNDI registry  the JNDI registry holds the reference to the EJB, not the actual bean instances themselves.  When an EJB is deployed, the EJB container registers it in the JNDI registry. bec client generally fetches ejb's from JNDI registry only  Clients use JNDI to look up the EJB and obtain a reference to it.  - same like spring @Autowired which can fetch and inject the bean from container, we should use JNDI registry lookup to fetch the ejb from reg |

import javax.naming.Context;

import javax.naming.InitialContext;

import javax.naming.NamingException;

public class CurrencyConverterClient {

public static void main(String[] args) throws NamingException {

Context context = new InitialContext();

CurrencyConverter converter = (CurrencyConverter) context.lookup("java:global/CurrencyConverterBean/CurrencyConverterBean!CurrencyConverter");

double amount = 100.0;

String fromCurrency = "USD";

String toCurrency = "EUR";

double convertedAmount = converter.convert(amount, fromCurrency, toCurrency);

System.out.println(amount + " " + fromCurrency + " = " + convertedAmount + " " + toCurrency);

}

}

In this example, the client application can be running on a different machine than the EJB server. The remote interface enables the client to invoke the currency conversion service as if it were running locally.

So in the case of a stateless bean, the JNDI registry is used to locate the EJB, and then the EJB container provides an instance of that EJB from it's pool of instances.

so internally ejb container maintains a pool of stateless session bean instances, same like connection pool instances, this will also maintain a pool