EJB Architecture

The overall architecture of Java EE and EJB 3.0 reflects the simplification of the EJB 3.0 model:

Architecture

The underlying concept of the EJB 3.0 specification centers on a plain old Java object (POJO) programming model that uses Java annotations to capture information that deployment descriptors used to contain. Deployment descriptors are now optional in most cases. Using default values liberally also means that you need to write and maintain less supporting code. This greatly simplifies the programming experience in creating and using EJB 3.0 components. Main features of this simplified model include

- EJBs are now plain old Java objects (POJO) that expose regular business interfaces (POJI), and there is no requirement for home interfaces.
- Use of metadata annotations, an extensible, metadata-driven, attribute-oriented framework that is used to generate Java code or XML deployment descriptors.
- Removal of the requirement for specific interfaces and deployment descriptors (deployment descriptor information can be replaced by annotations).
- Interceptor facility to invoke user methods at the invocation of business methods or at life cycle events.
- Default values are used whenever possible (¿configuration by exception¿ approach).
- Reduction in the requirements for usage of checked exception.
- A complete new persistence model (based on the JPA standard), that supersedes EJB 2.x entity beans

Comparison of steps to create beans in EJB 2.1 and EJB 3.0

Steps to define a stateless session bean in EJB 2.x	**Steps to define a stateless session bean inEJB 3.0**

To create a stateless session bean according to EJB 2.x specification, define the following components:
• **EJB component interface**: Used by an EJB client to gain access to the capabilities of the bean. This is where the business methods are defined. The component interface is called the EJB object. There are two types of component interfaces:
  ◦ Remote component interface (EJBObject):Used by a remote client to access the EJB through the RMI-IIOP protocol.
  ◦ Local component interface (EJBLocalObject): Used by a local client (that runs inside the same JVM) to access the EJB.
• **EJB home interface**: Used by an EJB client to gain access to the bean. Contains the bean life cycle methods of create, find, or remove. The home interface is called the EJB home. The EJBHome object is an object which implements the home interface, and as in EJBObject, is generated from the container tools during deployment, and includes container-specific code. At startup time, the EJB container instantiates the EJBHome objects of the deployed enterprise beans
	To declare a stateless session bean according to EJB 3.0 specification, simply define a POJO:
	`@Stateless`
	`public class MySessionBean implements MyBusinessInterface {`
	`business methods according to MyBusinessInterface .....}`
	• To expose the same bean on the remote interface, use the @Remote annotation:
	`@Remote(MyRemoteBusinessInterface.class)`
	`@Stateless`
	`public class MyBean implements MyRemoteBusinessInterface {`
	`// ejb methods`
	`.....`
	`}`

and registers the home in the naming service. An EJB client accesses the EJBHome objects using the Java Naming and Directory Interface (JNDI). There are two types of home interfaces:
    - Remote home interface (EJBHome):Used by a remote client to access the EJB through the RMI-IIOP protocol.
    - Local home interface (EJBLocalHome): Used by a local client (that runs inside the same JVM) to access the EJB.
- **EJB bean class**: Contains all the actual bean business logic. Is the class that provides the business logic implementation. Methods in this bean class associate to methods in the component and home interfaces.

Comparison of EJB 2.1 class plus Deployment Descriptor file with equivalent EJB 3.0 class

EJB 2.1	EJB 3.0

Java Class
 public class AccountBean implements javax.ejb.SessionBean {
 	
SessionContext ctx;
DataSource accountDB;
 	
public void setSessionContext(SessionContext ctx) {
this.ctx = ctx;
}
 	
public void ejbCreate() {
accountDB = (DataSource)ctx.lookup(
"jdbc/accountDB");
 	
}
public void ejbActivate() { }
public void ejbPassivate() { }
public void ejbRemove() { }
 	
public void setAccountDeposit(int empId,
double deposit) {
...
Connection conn = accountDB.getConnection();
...
}
...
}
	Java Class
	@Stateless
	public class AccountBean implements Account
	{
	@Resource private DataSource accountDB;
	
	public void setAccountDeposit(int customerId,
	double deposit) {
	...
	Connection conn = accountDB.getConnection();
	...
	}
	...
	}

Deployment Descriptor	**Deployment Descriptor** EJB 3.0 uses metadata annotations instead of deployment descriptors

```
<session>
  <ejb-name>AccountBean</ejb-name>
  <local-home>AccountHome</local-home>
  <local>Account</local>
  <ejb-class>com.example.AccountBean</ejb-class>
  <session-type>Stateless</session-type>
  <transaction-type>Container</transaction-type>
  <resource-ref>
    <res-ref-name>jdbc/accountDB</res-ref-name>
    <res-ref-type>javax.sql.DataSource</res-ref-type>
    <res-auth>Container</res-auth>
  </resource-ref>
</session>
...
<assembly-descriptor>...</assembly-descriptor>
```