Java Beans

¡Comencemos!

¿Qué aprenderás?

- Conocer Java Bean y la biblioteca de clases.

Introducción

En esta sección vamos a conocer a los Enterprise Java Beans, los componentes que dan vida a los grandes sistemas empresariales. Al principio leer estas descripciones son algo confusas, pero se intentará ser claro en su definición, ya que en la industria incluso veteranos desarrolladores java no pueden explicar de forma sencilla que es un enterprise java bean (EJB desde ahora).

La tecnología EJB es la arquitectura de componentes del lado del servidor para java EE7. La versión 3.2 de EJB permite un rápido y simplificado desarrollo de aplicaciones distribuidas, transaccionales y portátiles basados en el lenguaje java.

Los JavaBean en estricto rigor son un conjunto de normas o paradigmas que especifican el diseño de componentes de softwares reutilizables. Cualquier clase de Java es un potencial candidato a convertirse en un java bean, siempre y cuando sigan las especificaciones establecidas por la guía de especificación.

Las reglas para crear un java bean son precisas, como veremos un poco más adelante. Además de estas normas, hay una biblioteca de API de java beans que ayuda en el proceso de transformación de un POJO (plain old java object) en un componente java bean. La API está contenida en el paquete java.beans.

Una vez que se crea un bean java, se puede conectar como una entidad independiente a una aplicación que requiere el servicio definido por el componente. En este capítulo veremos algunos de los aspectos clave de la transformación de un POJO en un componente de bean y como la biblioteca de API de Java Bean ayuda en el proceso.

Los JavaBean y las bibliotecas de clases

Si los bean son clases java normales en esencia, ¿podemos crear clases como bean? Técnicamente sí se puede, pero no tendría sentido en el mayor de los casos. Los beans generalmente son apropiados para crear componentes de software visuales que son manipulados y personalizados para cumplir con requisitos específicos. Por ejemplo, pensemos en un componente visual como un marco o un botón, cuya forma y color se pueden personalizar de acuerdo con nuestras propias necesidades. Entonces, esas clases de java son perfectas para ser rebautizadas como java bean desde pojos.

Las bibliotecas de clases, por otro lado, son pojos que proporcionan funcionalidades para los programadores. No requieren de manipulación visual, por ejemplo las clases de la API de JDBC son clases de biblioteca que no requieren manipulación visual. Pero, siempre se puede hacer un bean de acceso a la base de datos sobre las clases jdbc, y en ese caso sería mejor crearlo como un componente javabean, de modo que se pueda usar o reutilizar en cualquier momento.

Existen ventajas y desventajas de los java beans:

Ventajas

- Los beans son persistentes, y se pueden recuperar.
- Podemos controlar qué propiedades, métodos y eventos deben exponerse a la aplicación que desea usarla.
- Un bean no solo puede registrarse para recibir eventos enviados por otro objeto, sino también generar eventos.

Desventajas

- Los beneficios de los objetos inmutables están ausentes en los beans porque son mutables por defecto.
- Debido a que cada bean Java debe tener un constructor sin argumentos (constructor nulo), esto puede llevar a que un objeto sea instanciado en un estado inválido.
- Las clases de bean contienen mucho código repetitivo porque, según la especificación, la mayoría o todas las propiedades privadas deben exponerse a través de captadores y definidores públicos. Esto a menudo lleva a escribir muchos códigos innecesarios.

Reglas para los JavaBean

En primer lugar, para crear un JavaBean, las siguientes reglas básicas deben mantenerse en una clase de Java. Estas reglas son en realidad convenciones que se deben respetar porque otras bibliotecas, cuando usan este componente, esperan que estas reglas se mantengan.

- Todas las propiedades privadas consideradas de acceso programático deben ser accesibles a través de un método de obtención y establecimiento público. Por ejemplo, si el salario es propiedad de una clase de empleado, debe estar expuesto a través de los métodos *getSalary* y *setSalary*.

- No importa cuántos constructores polimórficos se suministran, pero una clase de bean debe tener un constructor público sin argumentos.

- Una clase de bean debe ser serializable; o, en otras palabras, debe implementar una interfaz java.io.Serializable o java.io.Externalizable. Esto proporciona la capacidad de persistencia de un componente de bean.

Una clase que se desvíe de las reglas no será un bean porque JavaBean es un estándar. Una vez que sigue las reglas y diseña la clase de manera adecuada, el usuario que también conoce el estándar puede adaptar el componente a su aplicación fácilmente. Este es el objetivo de JavaBean: crear componentes de software reutilizables.

Características Importantes

Hay tres características importantes asociadas con JavaBeans:

- **Propiedades**: Atributos con nombre asociados con un bean.
- **Métodos**: Métodos POJO de los cuales el bean puede elegir exponer un subconjunto de sus tipos públicos.
- **Eventos**: Notifique a otros componentes si ocurre algo interesante.

Cuando usar los EJB

Como desarrollador JEE, debes considerar el uso de los EJB si tu aplicación tiene los siguientes requerimientos.

- **La aplicación debe ser escalable**: Para dar servicios a un gran número de usuarios, por lo cual necesitas distribuir los componentes de la aplicación a través de múltiples máquinas.
- **Las transacciones deben asegurar la integridad de los datos**: Los EJB soportan transacciones, el mecanismo que administra el acceso concurrente de objetos compartidos.
- **La aplicación tendrá una gran variedad de clientes**: Con pocas líneas de código, clientes remotos pueden fácilmente localizar EJB (dentro de un entorno empresarial).

Tipos de EJB

La especificación define tres tipos de EJB: Session beans, Entity Beans y Message-driven-beans. Los Entity Bean son los herederos de las versiones viejas del framework, en la actualidad están casi obsoletos y fueron reemplazados por tecnologías ORM como JPA (Java persistence api).

Los Message-driven beans son listener o escuchadores que esperan ser notificados por el container (contenedor de aplicaciones, por ejemplo Jboss) ante la llegada de un nuevo mensaje a una cola o tópico de mensajería. Finalmente están los Session Beans, que se dividen en dos grandes categorías, stateful y stateless. Estos son los verdaderos EJB. El siguiente gráfico resume lo expuesto.

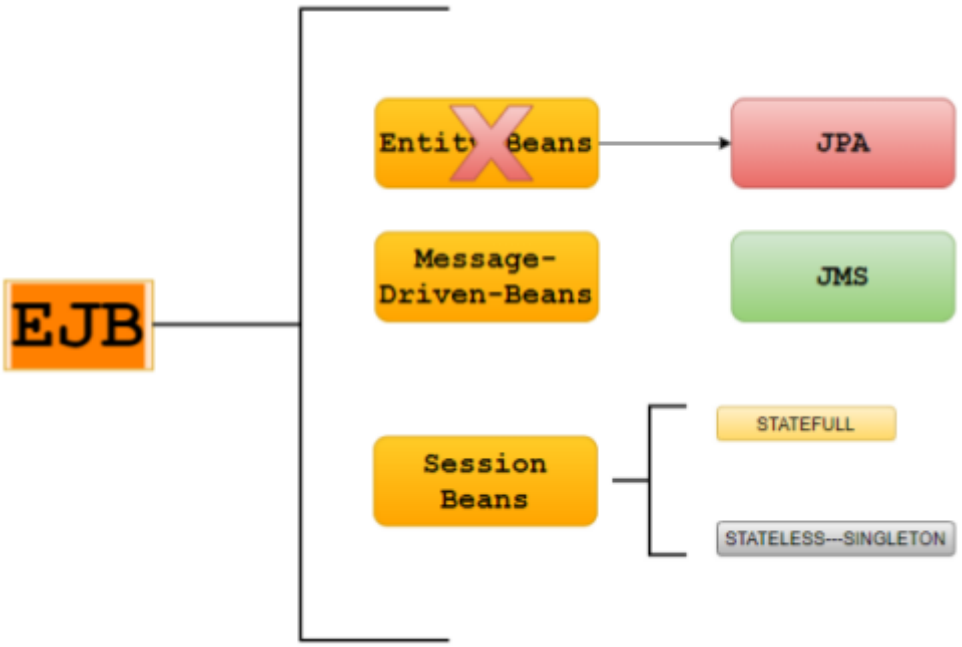

Imagen 1. Tipos de EJB.
Fuente: Desafío Latam.

Primer EJB

Para crear un EJB, en eclipse seleccionar *new-->project-->ejb*. Luego, creamos una clase de nombre *CalculadoraBean* e implementamos el siguiente código:

```
package com.ejb.sesionbean;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

/**
 * Session Bean implementation class CalculadoraBean
 */
@Stateless
@LocalBean
public class CalculadoraBean {

    public double sumar(double a, double b){
        return a+b;
    }

}
```

Este EJB no tiene mucha lógica, solamente suma dos números, pero aunque es una lógica de negocio que podemos implementar en un contenedor de aplicaciones como Jboss, WebLogic o algún otro. Este EJB se deploya en el servidor y queda disponible para que los clientes puedan acceder a él.

La anotación @Stateless indica que la clase es de tipo 'sin estado', lo que significa que no persisten sus valores y se borrarán si se reinicia el proceso. Este ejb es conocido como un Stateless Bean.

Clientes EJB

El EJB es un componente server-side que vive y se ejecuta en un contenedor. Se conoce como cliente ejb a cualquier proceso local o remoto que necesite de los servicios de este EJB. Existen 3 clientes:

- **Clientes Web**: Un servlet.
- **Clientes EJB**: Un EJB puede solicitar los servicios de otro EJB.
- **Clientes StandAlone**: Un proyecto java clásico con método main.

Cuando se accede a un EJB desde un servlet o desde otro EJB, se dice que el acceso es local, ya que estos componentes residen en la misma máquina. Pero si se desea acceder al EJB desde otra máquina, hay que considerar otros factores adicionales como por ejemplo, los parámetros de conexión JNDI. Para completar el ejemplo se necesita de un cliente web, el cual será un servlet de nombre TestEJB.

```
    @EJB
    private CalculadoraBean ejb;

    @WebServlet("/TestEJB")
    public class TestEJB extends HttpServlet {
      private static final long serialVersionUID = 1L;

      protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
            PrintWriter out = response.getWriter();
            out.print("<html><body>");

            int a = Integer.parseInt(request.getParameter("a"));
            int b = Integer.parseInt(request.getParameter("b"));

            out.print(ejb.sumar(a,b));

            out.print("</body></html>");
            out.close();
      }
    }
```