Enterprise JavaBeans

Sesión 1: Características básicas de EJB

Índice

- Desarrollo basado en componentes
- Servicios del servidor de aplicaciones
- Funcionamiento de los componentes
- Tipos de beans

Desarrollo basado en componentes

- Los componentes (*enterprise beans*) permiten reusar código y datos.
- Los componentes se despliegan en un servidor (contenedor EJB).
- Componente = objeto + servicios del contenedor EJB
- Ventajas del desarrollo basado en componentes:
  - Reusabilidad
  - Modularidad
  - Interoperabilidad

Características de los componentes

- Permiten el desarrollo de aplicaciones débilmente acopladas.
- Su comportamiento está especificado por interfaces.
- Su funcionamiento se puede configurar en tiempo de despliegue de forma declarativa (descriptor de despliegue).
- Los componentes EJB son multiplataforma (WODE: Write Once Deploy Everywhere).

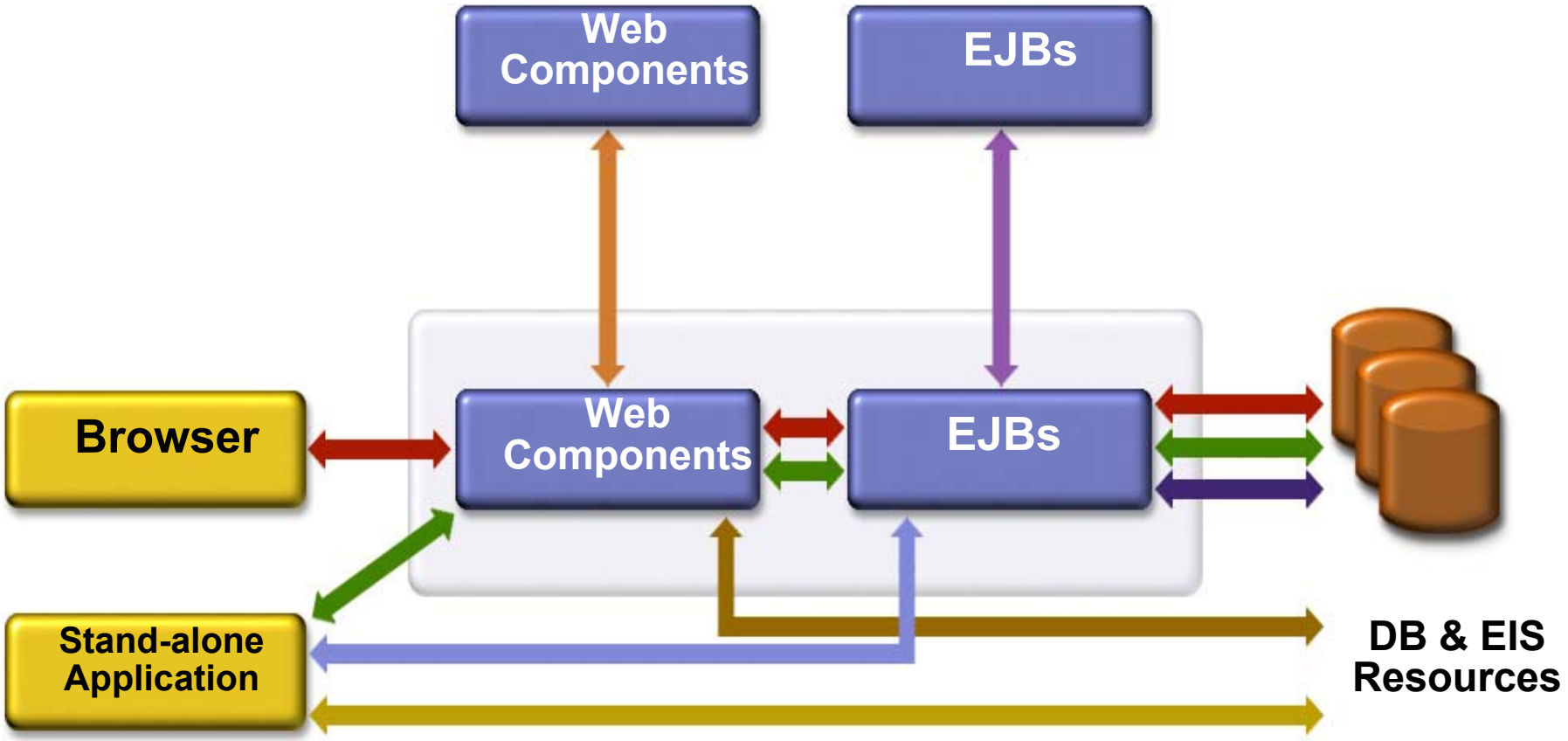
Un ejemplo de componentes

- **Un ejemplo bastante cercano: la universidad**
- **Al menos cuatro aplicaciones de gestión académica:**
  - ***Matriculación*** **(aplicación web)**
  - ***Campus virtual*** **(aplicación web)**
  - ***Ágora*****: gestión de matrícula, expedientes, actas (aplicación interna)**
  - ***Academia*****: Gestión de horarios, ocupación de aulas, carga docente de los profesores y departamentos (aplicación interna)**

Algunos componentes

- **Alumno**
  - **Datos: nombre, dirección, cuenta bancaria, ...**
  - **Aplicaciones: todas menos Academia**
- **Asignatura**
  - **Datos: nombre, créditos, plan de estudios, grupos, ...**
  - **Aplicaciones: todas**
- **Profesor**
  - **Datos: nombre, departamento, tipo, ...**
  - **Aplicaciones: Campus virtual y Academia**

EJB en J2EE


Web Components
EJBs
Browser
Web Components
EJBs
Stand-alone Application
DB & EIS Resources

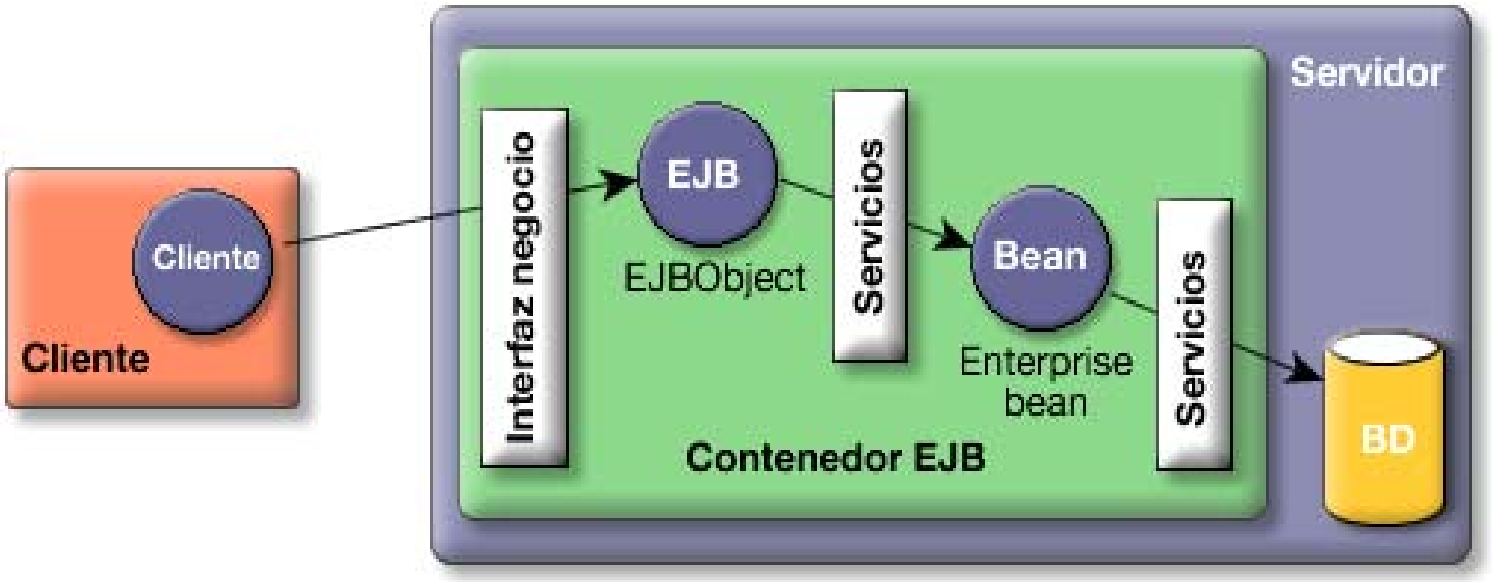
Servicios del contenedor EJB

- Transacciones
- Seguridad
- Concurrencia
- Threading
- Gestión y pooling de recursos
- Persistencia
- Gestión de mensajes
- Escalabilidad

Funcionamiento de los componentes EJB

- El cliente de un componente **nunca** hablará directamente con la implementación del componente (enterprise bean).
- Un objeto remoto hará de "cortafuegos" y permitirá que el contenedor interponga sus servicios en cada llamada.
- El objeto remoto ofrece los mismos métodos de negocio que el bean.

Funcionamiento de un bean



Tipos de beans

- **Beans de sesión (*Session Beans)***
  - **Beans de sesión sin estado**
  - **Beans de sesión con estado**
- **Beans de entidad (*Entity Beans*)**
  - **Con persistencia gestionada por el bean (BMP)**
  - **Con persistencia gestionada por el contenedor (CMP)**
- **Beans dirigidos por mensajes (*Message Driven Beans*)**

Beans de sesión

- Proporcionan un conjunto de funcionalidades y servicios a clientes
- No son persistentes, no representan datos existentes en un almacén de datos, aunque pueden acceder a ellos
- Modelan procesos de negocio, como solicitar un listado, enviar una notificación, ...
- Suelen recibir nombres como `ServicioBroker` o `GestorContratos`

Beans de sesión sin estado

- Ejecutan una petición del cliente sin guardar ninguna información del mismo
- Ejemplos:

```
Broker.compraAccion(accion)

Calculator.calcCuotaPrestamo()
```

Beans de sesión con estado

- Definen variables de instancia que almacenan datos específicos del cliente
- Estos datos se guardan con operaciones denominadas `setXXX` como `setNombre` o `setDireccion`
- Se suelen usar para implementar métodos de negocios que requieren multiples pasos de ejecución
- El estado del bean dura el tiempo de sesión y no es persistente
- Ejemplo típico: carrito de la compra

Beans de sesión en el contenedor

- Los beans de sesión son muy eficientes y ligeros
- El contenedor tiene una reserva (pool) de beans de sesión que va reusando según es necesario
- Un bean de sesión sin estado puede ser compartido por más de un cliente

Uso de los beans de sesión

- **Para usar un bean de sesión un cliente debe:**
  - **Conseguir una instancia del bean llamando al método `create` de la interfaz home del bean**
  - **Llamar a los métodos de negocio de la interfaz componente del bean**
  - **Terminar el uso del bean llamando al método `remove` del bean**

Beans de entidad

- Los beans de entidad representan datos de negocio
- Proporcionan una vista orientada a objetos de una base de datos
- Una instancia de un bean de entidad corresponde a una fila de una tabla (o de más de una si están normalizadas)
- Los datos del bean se guardan en sus variables de instancia y se leen y escriben en la base de datos cuando el contenedor lo requiere

Ejemplo de bean de entidad

- **Estudiante**
  - **Los datos del estudiante deben persistir y se encuentran en una base de datos**
  - **Los datos de un cliente se comparten entre múltiples aplicaciones**
  - **Cada estudiante tiene un identificador único que sirve para localizarlo**

Tipos de beans de entidad

- **CMP (Persistencia gestionada por el contenedor)**
  - **La relación entre variables de instancia del bean y campos de la base de datos se define en el descriptor de despliegue**
  - **El contenedor se encarga de actualizar el bean y la base de datos**
  - **Muy eficiente**
- **BMP (Persistencia gestionada por el bean)**
  - **El programador se encarga de escribir el código que gestiona la actualización del bean y de la base de datos**

Bean de entidad en el contenedor

- Los beans de entidad son menos eficientes que los beans de sesión
- Es recomendable el uso de CMP ya que el contenedor optimiza los acceso a las bases de datos

Uso de los beans de entidad

- **Para usar un bean de entidad un cliente debe:**
  - **Obtener una referencia al bean de entidad que se requiere usando un método finder de la interfaz home del bean**
  - **El cliente interactúa con el bean usando los métodos get y set de su interfaz componente**
  - **Cuando el cliente termina la interacción el contenedor vuelca sus datos en la base de datos**

Desarrollo de los beans

- 1. Escribir la clase bean con la implementación de los métodos de negocio
- 2. Escribir las interfaces componente y home
- 3. Crear el descriptor de despliegue ejb-jar.xml
- 4. Crear el fichero EJB JAR
- 5. Desplegar el bean en el contenedor
- 6. Usar el bean desde los clientes

1. Escribir la clase bean

- La clase bean debe implementar la interfaz `javax.ejb.SessionBean`, `EntityBean` o `MessageBean`
- En la clase se deben implementar:
  - Funciones de la interfaz ejb relacionadas con su ciclo de vida (`ejbActivate`, `ejbPassivate`, ...)
  - Funciones de la interfaz componente: métodos de negocio
  - Funciones de la interfaz home: creación del bean

1. Clase SaludoBean.java

```
package especialsta;
import javax.ejb.*;

Public class SaludoBean implements SessionBean {
   private String[] saludos = {"Hola", "Que tal?", "Como estas?",
   "Cuanto tiempo sin verte!", "Que te cuentas?", "Que hay de nuevo?"};

   public void ejbActivate() {
      System.out.println("ejb activate");
   }

   public void ejbPassivate() {
      System.out.println("ejb pasivate");
   }

   public void ejbRemove() {
      System.out.println("ejb remove");
   }

   public void setSessionContext(SessionContext cntx) {
      System.out.println("set session context");
   }

   public String saluda() {
      System.out.println("estoy en saluda");
      int random = (int) (Math.random() * saludos.length);
      return saludos[random];
   }
   public void ejbCreate() {
      System.out.println("ejb create");
   }
}
```

2. Escribir la interfaz componente

- La interfaz componente debe extender la interfaz `javax.ejb.EJBObject`
- En la interfaz se deben definir:
  - Métodos de negocio del bean

2. Clase Saludo.java

```
package especialista;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface Saludo extends EJBObject {
   public String saluda() throws
RemoteException;
}
```

2. Escribir la interfaz home

- La interfaz componente debe extender la interfaz `javax.ejb.EJBHome`
- En la interfaz se deben definir:
  - Métodos de creación del bean

2. Clase SaludoHome.java

```
package especialista;
import javax.ejb.*;
import java.rmi.RemoteException;

public interface SaludoHome extends EJBHome {
   public Saludo create() throws
CreateException, RemoteException;
}
```

3. Fichero de despliegue

- Define las características del bean que debe gestionar el contenedor EJB
- Fichero XML: `ejb-jar.xml`
- Las más importantes:
  - Transacciones
  - Seguridad
  - Nombre del bean
  - Nombre JNDI del bean
- El nombre JNDI del bean se define en un fichero descriptor no estándar que depende del contenedor (`weblogic-ejb-jar.xml`)

3. Fichero ejb-jar.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC
'-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
1.1//EN'
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>

<ejb-jar>
  <display-name>Ejb1</display-name>
  <enterprise-beans>

    <session>
      <display-name>SaludoBean</display-name>
      <ejb-name>SaludoBean</ejb-name>
      <home>especialista.SaludoHome</home>
      <remote>especialista.Saludo</remote>
      <ejb-class>especialista.SaludoBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>

  </enterprise-beans>
</ejb-jar>
```

4. Comprimir la estructura EJB JAR

- **El fichero EJB JAR es un fichero JAR que contiene la siguiente estructura:**

```
saludoEjb/especialista/Saludo.class
saludoEjb/especialista/SaludoBean.class
saludoEjb/especialista/SaludoHome.class
saludoEjb/META-INF/ejb-jar.xml
saludoEjb/META-INF/weblogic-ejb-jar.xml
```

5. Desplegar el bean

- Herramientas específicas del servidor de aplicaciones
- Se suele usar
  - Ant
  - Consola de administración del servidor de aplicaciones
  - Entorno gráfico independiente (WebLogic Builder)
  - Opción integrada en el entorno de desarrollo

6. Usar el bean en un cliente

- Obtener el objeto EJBHome (realmente un stub del objeto EJBHome) mediante JNDI
  - Obtener el contexto inicial
  - Realizar loockup
  - Realizar narrowing
- Mediante el objeto home obtener un objeto EJBObject (realmente un stub del objeto EJBObject)
- Invocar un método de negocio a través del objeto EJBObject

6. Fichero SaludoClient.java

```
 1. import java.io.*;
 2. import java.text.*;
 3. import java.util.*;
 4. import javax.servlet.*;
 5. import javax.servlet.http.*;
 6. import javax.naming.*;
 7. import javax.rmi.*;
 8. import especialista.*;
 9.
10. public class SaludoClient {
11.
12.   public static void main(String [] args) {
13.      try {
14.         Context jndiContext = getInitialContext();
15.         Object ref = jndiContext.lookup("SaludoBean");
16.         SaludoHome home = (SaludoHome)
17.            PortableRemoteObject.narrow(ref, SaludoHome.class);
18.         Saludo sal = (Saludo)
19.            PortableRemoteObjet.narrow(home.create(), Saludo.class);
20.         System.out.println("Voy a llamar al bean");
21.         System.out.println(sal.saluda());
22.         System.out.println("Ya he llamado al bean");
23.      } catch (Exception e) {e.printStackTrace();}
24.   }
25.
26.   public static Context getInitialContext()
27.     throws javax.naming.NamingException {
28.        Properties p = new Properties();
29.        p.put(Context.INITIAL_CONTEXT_FACTORY,
30.                "weblogic.jndi.WLInitialContextFactory");
31.        p.put(Context.PROVIDER_URL, "t3://localhost:7001");
32.        return new javax.naming.InitialContext(p);
33.   }
34. }
```