La capa de Negocio de la Arquitectura Java EE

Autores: Simon Pickin
Florina Almenárez Mendoza
Natividad Martínez Madrid
Pablo Basanta Val

Dirección: Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
España

Versión: 1.0

Agradecimientos: Bill Burke, JBoss Group, Tal Cohen, IBM Haifa

Software de Comunicaciones

3. Enterprise JavaBeans 3.0

Software de Comunicaciones

Introducción

- La capa de negocio implementa la lógica de negocio
  - Esto es, la funcional nuclear de una aplicación empresarial
- Enterprise JavaBeans (EJB) es una especificación completa de una arquitectura de servicios basada en componentes
  - Ejecutan lógica de negocio
  - Acceso a bases de datos
  - Se integra con otros sistemas
- Ventajas de los EJBs:
  - El desarrollador se concentra en la lógica de negocio y usa los servicios del contenedor
    - transacciones, seguridad, gestión de ciclo-de-vida, multi-hilo, almacén de conexiones, etc.
  - Componentes: reutilización y abstracción
  - Compatible con otras APIs Java

Servicios del contenedor de EJBs (1/5)

Contenedor: entorno de ejecución para todos los componentes EJB instalados

- Gestión de transacciones:
  - Secuencia de acciones (y accesos a datos) ejecutados “atómicamente”
    - ACID (Atomic, Consistent, Isolated, Durable)
    - Evita problemas que pueden surgir del acceso a datos de forma concurrente
    - La secuencia entera puede ser deshecha (“rolled back”) en caso de fallo
  - El contenedor provee protocolos de manejo de transacciones
    - Por ejemplo: un protocolo de aceptación de dos fases
  - Transacciones gestionadas por el bean (BMT)
    - El desarrollador del bean codifica explícitamente el inicio de la transacción, la finalización, el proceso de deshacerla, … usando JTA (Java Transaction API)
  - Transacciones manejadas por el contenedor(CMT)
    - El desarrollador del bean no codifica las transacciones explícitamente
    - Los métodos que debe de ejecutar en una transacción están especificados con anotaciones o en el descriptor de despliegue (archivo `xml`).

Servicios del contenedor de EJBs (2/5)

- Gestión de recursos y ciclo de vida:
  - recursos: hilos, conexiones, conexiones a la base de datos,…
  - Ciclo de vida: creación y destrucción de instancias, activación y pasivación de instancias…
  - Estados exactos en el ciclo de vida y procesos que dependen del tipo de bean utilizado
- Accesibilidad remota / objetos distribuidos:
  - El desarrollador del bean no codifica explícitamente el código de acceso remoto
  - El servidor de EBJs provee protocolos de comunicación para acceder a objetos remotos distribuidos.
    - Debe soportar RMI-IIOP (c.f. especificación CORBA) y SOAP 1.2 (vía API JAX-WS o la antigua API JAX-RPC)
    - Puede soportar también otros protocolos de comunicación
  - El contenedor implementa llamadas distribuidas usando la infraestructura de comunicación
    - Por ejemplo genera sustitutos y esqueletos

Servicios de contenedor de EJBs (3/5)

- Seguridad:
  - Autenticación
    - Validación de la entidad del usuario
  - Autorización (de acceso a componentes)
    - Política de seguridad especificando lo que un usuario puede y no puede hacer
    - Concepto declarativo basado en roles de usuario
    - Roles y sus derechos de acceso a métodos de negocio definido con anotaciones o en el descriptor de despliegue del fichero `xml`.
    - El desarrollador asigna roles a usuarios
      - El servidor de EJBs gestiona usuarios y roles
      - El contenedor trata con el control de acceso
  - Comunicaciones seguras
- Concurrencia (“multi-hilo”)
  - El desarrollador de beans no codifica explícitamente el multi-hilo
  - Por ejemplo, las dos maneras de manejar peticiones concurrentes
    - Mantener un pool de instancias y peticiones directas a instancias del bean del repositorio
    - Serializar peticiones a una instancia de un bean.

Servicios de contenedor de EJBs (4/5)

- Servicio de nombramiento y directorio:
  - Asociación de nombres a referencias a objetos en una estructura de directorio jerárquica
    - API JNDI (Java Naming and Directory Interface) :
  - Contexto de nombramiento del entorno de un bean:
    - Espacio de nombres de JNDI (directorio privado) específico a cada clase de bean
    - Para acceder a las propiedades, recursos, otros beans del contenedor
    - Referenciado desde dentro de un bean vía `java:comp/env`
- Mensajería:
  - El contenedor provee acceso al servicio de mensajería
    - API JMS (Java Messaging Service)
  - Comunicaciones asíncronas entre dos o más participantes
    - a través de un sistema de colas de mensajes
  - Los receptores de los mensajes deben ser beans dirigidos por mensajes
    - Cualquier bean empresarial puede ser un emisor de mensajes

Servicios de contenedor de EJBs (5/5)

- Temporizador/Planificador
  - Planifica notificaciones de planificación para ser enviadas a EJBs en instantes específicos de tiempo.
  - c.f. cron de Unix
- Persistencia (EJB2):
  - Los beans de entidad EJB2.1 pueden ser usados como alternativa a las entidades de JTA
    - El contenedor EJB3 debe soportar beans de entidad EJB2.1
  - Las instancias de los beans de entidad en memoria están enlazadas a datos de negocio
    - El contenedor garantiza la consistencia (con carga y almacenamiento periódico)
  - Persistencia manejada por el bean (BMP)
    - El desarrollador de beans codifica el acceso a la base datos explícitamente usando JDBC pero el contenedor decide cuando llamar a ese código.
    - La instancia del bean usa conexiones JDBC provistas por el contenedor
  - Persistencia manejada por el contenedor (CMP)
    - El desarrollo del bean no codifica el acceso a la base de datos explícitamente
    - Generalmente soporta conexión a bases de datos relacionales
    - El significado exacto de la persistencia depende del contenedor y es independiente del bean

Modelando aplicaciones empresariales

- Las aplicaciones empresariales están organizadas en componenentes que implementan entidades de negocio o procesos de negocio
  - Las entidades de negocio representan información empresarial
  - Los procesos de negocio representan la manipulación de dicha información

Entidades de negocio

- Las entidades de negocio son objetos de negocio:
  - Representan información mantenida por la compañía
  - Tienen estado persistente (mantenido en una base de datos)
- Ejemplos:
  - cliente, orden de compra, cuenta, empleado,...
- Pueden tener asociadas “reglas de negocio”:
  - Restringiendo valores del estado de la entidad
    - Por ejemplo, códigos postales con 5 dígitos (en España, por lo menos!)
  - Manteniendo relaciones entre entidades
    - Por ejemplo, relacionando un cliente a varias compras

Procesos de negocio

- Objetos de negocio que encapsulan una interacción entre un usuario y una entidad de negocio
  - Actualizan un estado de entidades de negocio
  - Mantienen una entidad única a través del ciclo de vida
- Pueden tener estado propio
  - Estado persistente: proceso dividido en etapas y puede involucrar múltiples actores: *proceso de negocio colaborativo*
    - Ejemplo: procesado de una petición de préstamo
  - Estado transitorio: proceso completado en una conversación con un actor: *proceso de negocio conversacional*
    - Ejemplo: retirada de dinero de un cajero

Reglas de negocio

- Distribuidas entre los componentes que implementan las entidades y los procesos de negocio
  - De acuerdo a si la regla se aplica a la entidad o al proceso
- Ejemplos:
  - Entidad:

    El balance de una cuenta no puede ser negativo

    (independiente del proceso que causa que ocurre)
  - Proceso:

    La máxima cantidad que puede ser retirara de un cajero es 500€

    (independiente del estado de la entidad de la cuenta)

Enterprise Java Beans

- Beans de sesión:
  - Procesos ejecutados como respuesta a una petición de un cliente (e.g. transacciones bancarias, cálculos, implementación de compras …)
  - Proceso colaborativo: hace uso de entidades JPA / beans de entidad
  - Recibe llamadas síncronas a métodos definidos sobre la interfaz de negocio
- Entidades JPA (EJB3) / Beans de entidad (EJB2):
  - Objetos persistentes asociados a datos
    - Por ejemplo: cuenta bancaria, compra de producto, …
  - Soporte a la información pasiva vía métodos para las operaciones sobre datos
  - Reciben llamadas síncronas a métodos definidos en la interfaz de negocio
- Beans dirigidos por eventos:
  - Procesos ejecutados como respuesta a la recepción de un mensaje
  - Reciben llamadas asíncronas a través de un canal

Entidades JPA (EJB3) / Beans de entidad (EJB 2.x)

- Modelan conceptos / objetos de negocio con estado persistente
  - Por ejemplo, datos en la base de datos
- Pueden ser utilizados por varios clientes conjuntamente y simultáneamente
- Entidad visible externamente: *clave primaria*
  - La instancia puede ser accedida por otros programas
- Larga vida (tiempo de vida de sus datos asociados)
  - El estado persistente típicamente cambia transaccionalmente
  - El estado sobrevive el reinicio del contenedor, el servidor o la computadora
- Sólo para beans de entidad: la persistencia puede ser manejada por el bean o por el contenedor (no se ve desde el cliente la diferencia)
- Solo para entidades JPA: pueden ser desacopladas de la capa de persistencia, y re-acopladas (mezcladas) a la capa de persistencia.

Software de Comunicaciones

Beans dirigidos por mensajes

- Los beans dirigidos por mensajes son receptores de mensajes
- Usan un servicio de mensajería
  - Intermediario entre el emisor y el bean dirigido por mensajes
  - Mensajes entrantes capturados por el contenedor y redirigidos a una instancia de bean
  - Patrón publicador-subscriptor:
    emisor y receptor (bean dirigido por eventos) mutuamente anónimos.
- Comunicación asíncrona
  - Beans de entidad y sesión: (bloqueo) invocaciones síncronas a métodos
- Los beans dirigidos por mensajes no tienen entidad
  - Como los beans de sesión sin estado:
    no pueden mantener información sobre el estado del emisor

Beans de sesión

- Ciclo de vida del bean de sesión
  - Instancia creada / enlazada cuando un cliente llama al bean
    - Asociada a un cliente como recurso privado durante la duración de un proceso cliente
  - Instancia eliminada / liberada cuando un proceso cliente finaliza
- Beans de sesión sin estado:
  - Un proceso cliente implica una operación simple
  - No almacenar datos específicos de cliente entre invocaciones
    - Usar los datos pasados como parámetros u obtenidos de la base de datos
  - Todas las entidades tienen la misma instancia
- Beans de sesión con estado:
  - Un proceso de cliente involucra múltiples invocaciones
  - Mantienen el estado a través de múltiples invocaciones
    - Estado dependiente del cliente y llamado *conversational state*
  - El estado no es persistente: perdido cuando un cliente libera el bean
  - Cada instancia tiene una entidad diferente

Software de Comunicaciones

Beans de sesión, vista cliente

- La también llamada “interfaz de negocio”
  - Es un POJI (Plain Old Java Interface)
  - Declara métodos que pueden ser llamados por beans cliente
  - Puede ser declarada como local con la anotación `@Local`
    - Después puede ser accedida por otros beans en el mismo contenedor EJB
  - Puede ser declarada como remota a través de la anotación `@Remote`
    - Después puede ser accedida por las aplicaciones fuera del contenedor de EJBs
    - No necesita declarar excepciones remotas de RMI
  - Puede ser generada desde una clase de bean por el contenedor
    - Si todos los métodos de la clase del bean van a estar disponibles para los clientes
- Cuando un bean invoca un método en la interfaz de negocio
  - La interacción es con un sustituto, no con la clase del bean **IMPORTANTE!**
  - El sustituto encamina la invocación y responde a través del contenedor
  - El contenedor inyecta servicios del middleware basados en metadatos del bean
    - Metadatos especificados como anotaciones o en el descriptor de despliegue XML

Ejemplo 3: Bean de sesión sin estado, v1

```
package examples.session.stateless;

// Esta es la interfaz de negocio Hello
public interface Hello {
  public String hello();
}
```

```
package examples.session.stateless;

import javax.ejb.Remote;
import javax.ejb.Stateless;

// Bean de sesión sin estado
@Stateless
@Remote(Hello.class)
public class HelloBean implements Hello {
  public String hello() {
    return "Hello, World";
  }
}
```

Ejemplo 3: Bean de sesión sin estado, v2

```
package examples.session.stateless;

// Esta es la interfaz de negocio Hello
@Remote public interface Hello {
  public String hello();
}
```

```
package examples.session.stateless;

import javax.ejb.Remote;
import javax.ejb.Stateless;

// Bean de sesión sin estado
@Stateless
public class HelloBean implements Hello {
  public String hello() {
    return "Hello, World";
  }
}
```

Ejemplo 3: Bean de sesión sin estado, v3

- El contenedor genera la interfaz de negocio (todos los métodos serán expuestos)

```
package examples.session.stateless;

import javax.ejb.Remote;
import javax.ejb.Stateless;

// Bean de sesión sin estado
@Stateless
@Remote
public class HelloBean {
  public String hello() {
    return "Hello, World";
  }
}
```

Software de Comunicaciones

Ejemplo 4: Bean con estado en EJB 3

- Un método debería ser anotado con `@Remove`

```
@Remote public interface ShoppingCart  {
   public void addItem(int prodId, int quantity);
   public void checkout();
}

@Stateful public class ShoppingCartBean
                              implements ShoppingCart {

   @Remove
   public void checkout()  {
     …   }
}
```

Software de Comunicaciones

Inyección de dependencia en EJBs vía anotaciones de entorno

- Referenciando otros EJBs vía **`@EJB`**
  - Con anotación de clase: no es inyección
    - Es enlazar el nombre a una referencia en ENC (ver más tarde)
  - Con anotación de campo: inyección de campo
    - El contenedor inyecta una referencia a EJB dentro de un campo
  - Con una anotación de métodos set: inyección de método set
    - El contenedor invoca un método setter con una referencia inyectada como parámetro
  - Elementos involucrados **`name`**, **`beanName`**, **`beanInterface`**,...
- Referenciar otros recursos vía **`@Resource`**
  - Para recursos distintos de EJBs, unidades de persistencia / contextos
  - Con anotación de clase: no es inyección
    - Es unir un nombre a una referencia en ENC (ver más tarde)
  - Con anotación de campo / método set: inyección de campo / método set
  - Elementos involucrados **`name`**, **`type`**, **`authenticationType`**, ...

Software de Comunicaciones

El contexto de nombres empresarial en JNDI (1/2)

- Cada bean tiene su contexto de nombramiento empresarial (ENC)
  - Su propia parte de espacio de nombres de JNDI
  - Para guardar referencias a recursos, otros EJBs,…
  - Identificado por un bean de instancia vía **`java:comp/env`**
- Populando el ENC (enlazado)
  - Puede ser especificado vía instrucciones en el descriptor de despliegue
  - 'efecto-colateral' de la inyección de dependencia a través de anotaciones de entorno
  - Entradas colocadas en el ENC de componentes por el contenedor en el despliegue
- Accediendo al ENC (búsqueda)
  - Vía JNDI **`javax.naming.InitialContext.lookup()`**
  - Vía **`EJBContext.lookup()`** (ligeramente más sencilla)
    - El objeto **`EJBContext`** puede obtenerse vía inyección (**`@Resource`**)
    - Los beans de sesión: usan **`SessionContext`** (que extiende **`EJBContext`**)

Software de Comunicaciones

El contexto de nombres empresarial en JNDI (2/2)

- Inyección de dependencias a través de anotaciones de entorno
  - Es una alternativa a obtener dependencias mediante búsqueda en ENC
  - Pero tiene un ‘efecto-colateral’ de enlazado de nombre en ENC
- El enlazado de nombre en ENC como ‘efecto-lateral’ de la inyección de dependencia
  - Permite que una inyección sea sobrescrita en el descriptor de despliegue
  - Qué nombre se asocia a una referencia en ENC
    - Se usa el nombre de la construcción anotada: por defecto
    - Se especifica un nombre usando el elemento **`name`** de la anotación

Ejemplo 5: Inyección de dependencia

- La clase del bean especifica dependencias y no búsqueda
  - Es posible probar EJBs fuera del contenedor
  - Varias anotaciones diferentes son posibles

```
@Stateful public class ShoppingCartBean
                                implements ShoppingCart {

   @Resource private SessionContext ctx;

   @EJB(name="CreditProcessorEJB")
   private CreditCardProcessor processor;

   private DataSource jdbc;

   @Resource(name="java:/DefaultDS")
   public void setDataSource(DataSource db) {
        this.jdbc = db;
   }
}
```

Software de Comunicaciones

Gestión de recursos en beans de sesión sin estado: repositorio de instancias (1/2)

- No hay razón para tener una instancia separada para cada cliente
  - Los clientes EJB no acceden a instancias de EJBs directamente
    - Accedidas a través de un objeto representante
  - Los beans de sesión sin estado no tienen estado dependiente del cliente, aunque
    - Pueden tener variables de método
    - Pueden obtener información del ENC de JNDI o una base de datos
- Reutilización de instancia e intercambio de instancia
  - El servidor mantiene un almacén de instancias pre-creadas
  - La instancia se asocia a un representante dinámicamente (por invocación)
    - Invocaciones sucesivas de un cliente pueden ser servidas por instancias diferentes
    - La misma instancia de bean puede ser intercambiada entre clientes / representantes
  - Unos pocos beans de sesión pueden servir muchos clientes simultáneos
  - Gestión de recurso:
    - Alta dinamicidad: ahorra tiempo de creación de una instancia en su invocación
    - Uso eficiente de memoria: minimiza el número de instancias

Gestión de recursos en beans de sesión sin estado: repositorio de instancias (2/2)

- El repositorio de instancias se usa también en beans dirigidos por mensaje
  - Los beans se suscriben a un canal de mensajes específico
  - Los productores entregan mensajes a uno de los canales
  - El contenedor crea un repositorio de beans para cada canal
- El repositorio de instancias es también usado en los beans de entidad de EJB2.1
  - El bean de entidad en el almacén no está asociado a datos persistentes
  - No está desacoplado de la capa de persistencia como en las entidades JPA
    - Los beans de entidad del repositorio tienen los atributos sin inicializar

Gestión de recursos en beans de sesión con estado: activación y pasivación

- No hay repositorio de instancias para los beans de sesión con estado
  - El estado de la conversación con el cliente se mantiene
- El contenedor puede usar activación / “pasivación”
  - Gestión de recursos: gana espacio y pierde tiempo
    - Adecuado para un uso alto de memoria y baja dinamicidad de aplicación
  - El mecanismo es transparente al cliente
- “Pasivación”:
  - Desasociación de un objeto EJB (a una clase representante de un bean) desde una estancia de bean
  - La serialización del estado de estancia a un almacenamiento secundario
- Activación:
  - Deserialización de un estado de una instancia desde un almacenamiento secundario
  - Restauración de una asociación a un objeto EJB

Activación / “Pasivación” en EJB2

- Término también aplicado a los beans de entidad de EJB2
  - Se refiere a moverse desde un o a un repositorio de instancias
- “Pasivación”
  - Cortar la conexión con el objeto EJB (representante para la clase del bean)
  - Almacenar los datos del bean en la base de datos subyacente
    - Los valores atributo del bean del repositorio no tienen sentido por más tiempo
  - Liberar cualquiera recursos que estén siendo usados por el bean
  - Colocar el bean en el repositorio
- Activación
  - Coger un bean anónimo del pool
  - Tomar los recursos que serán usados por el bean
  - Cargar datos desde la base de datos subyacente dentro de los atributos del bean
  - Restaurar la conexión con el objeto EJB

Métodos de retro-llamada y del ciclo de vida en el bean de sesión

- Los métodos llamados por el contenedor de EJBs
  - El bean de sesión opcionalmente se registra para métodos de retro-llamada sobre eventos del ciclo de la vida
    - Anotando métodos de la clase del bean
  - Restricciones sobre los métodos
    - Deben de ser declarados `public`, no tener argumentos y retornar `void`
    - No pueden retornar excepciones de aplicación
  - Pueden ser colocados en una clase escuchadora a parte
    - Declarados a través de la siguiente anotación `@Interceptors`
- Los métodos del ciclo de vida de retro-llamada para cualquier tipo de bean de sesión
  - Métodos anotados con `@PostConstruct`
    - Llamados por el contenedor justo después de crear una nueva instancia de la clase
  - Métodos anotados con `@Predestroy`
    - Llamados por el contenedor, después de que el método `@Remove` finalice
- Un método anotado con `@Remove`
  - No es un método de retro-llamada
  - Le dice al contenedor que un bean puede ser eliminado cuando el método finaliza

Métodos de retro-llamada en beans de sesión con estado

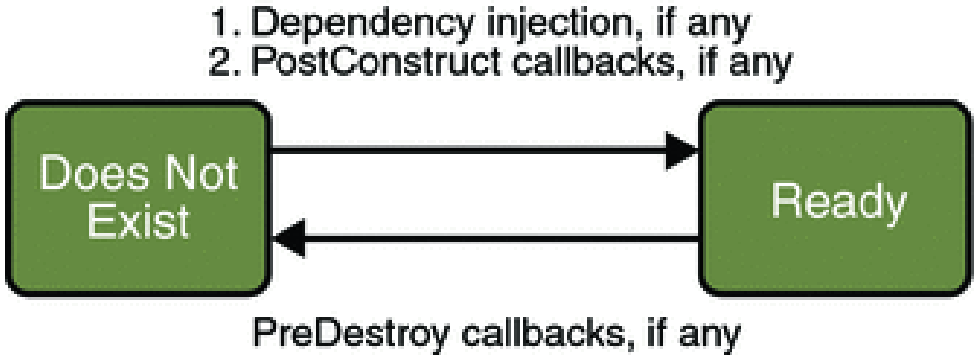
- Los métodos de retrollamada del ciclo de vida son específicos a los beans de sesión con estado
  - Un método anotado con **`@PrePassivate`**
    - Se llama por el contenedor justo antes de *pasivizar* el bean
    - Por ejemplo, para renunciar a recursos como sockets, conexiones a la base de datos
  - Un método anotado con **`@PostActivate`**
    - Se llama por el contenedor justo después de activar el bean
    - Por ejemplo, para restaurar recursos como sockets, conexiones a la base de datos

Ciclo de vida del bean de sesión

- Creación del bean de sesión
  - El contenedor decide instanciar el bean
    - Llama a `Class.newInstance()`
  - El contenedor inyecta cualquier dependencia de contexto requerida
  - El contenedor llama a métodos de retrollamada opcionales etiquetados con `@PostConstruct`
- Uso de un bean de sesión
  - El contenedor puede llamar a métodos de negocio en nombre de clientes
    - Recordar: el cliente llama a un representante, el contenedor llama a la instancia del bean
- Destrucción del bean de sesión
  - El contenedor decide eliminar el bean
  - El contenedor llama a todos los métodos de retrollamada etiquetados con `@PreDestroy`
    - No se llama cuando se cae el servidor → es necesario que la aplicación guarde proactivamente sus datos
  - La instancia del bean está lista para la recolección de basura

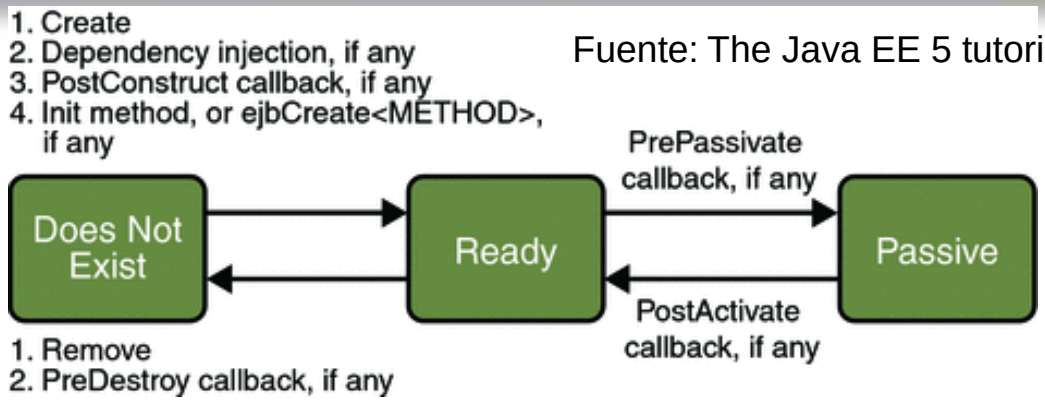
Ciclo de bean del bean de sesión sin estado

Fuente: The Java EE 5 tutorial. Sun Microsystems



- Creación y destrucción
  - Creación:
    - Si no hay reutilización: cuando un cliente busca obtener una referencia de bean
    - Si la hay: cuando la política del contenedor lo requiere
  - Destrucción:
    - Si no hay reutilización: cuando la invocación retorna
    - Si lo hay: cuando el contendor lo requiere

Ciclo de vida de un sesión bean con estado



Fuente: The Java EE 5 tutorial. Sun Microsystems

- Creación y destrucción
  - Creación:
    - Cuando un cliente busca obtener una referencia a un bean
  - Destrucción:
    - Cuando un cliente llama a un método anotado con **`@remove`** o cuando vence el temporizador
- Activación y “pasivación”
  - “pasivación”,después de llamar a cualquier método opcional **`@PrePassivate`**
    - Cuando se alcanza un límite en el número de beans instanciados
  - Activación, después de llamar a cualquier método opcional **`@PostActivate`**
    - Cuando un cliente invoca un método de negocio

Transacciones

- Transacción:
  - Un conjunto de tareas que se ejecutan atómicamente
    - Si una o más tareas fallan: *deshacer la transacción*
    - Si todas las tareas son exitosas: *comprometerla*
  - Propiedades ACID
    - Atomicidad, Consistencia, aIslamiento, Durabilidad

- Dos maneras de manejar transacciones en EJBs
  - *Transacciones declarativas* (política por defecto)
    - Demarcación de transacciones gestionada por el contenedor (CMT)
  - *Transacciones programáticas*
    - Demarcación de transacción manejada por el bean (BMT)
    - Transacciones iniciadas por el cliente
      - pros: un cliente es consciente de si la transacción ha sido deshecha o comprometida
      - cons: problemas de rendimiento

Transacciones manejadas por el contenedor

- Facilidad de uso
- Comportamiento transaccional
  - Independiente de la lógica de negocio
- Uno de atributos transaccionales para controlar la propagación
  - Declarado como anotaciones o en el descriptor de despliegue
  - Asociado con cada método de EJB
- El contenedor usa el API JTS para gestionar automáticamente
  - El comienzo y el fin de una transacción
  - La iteración con la base de datos
  - Creación y propagación del contexto durante la transacción

Transacciones manejadas por el bean

- Dificultad de uso
  - Pero un control más fino
- Comportamiento transaccional
  - Puede depender de la lógica de negocio
- Un uso explícito de
  - El API de transacciones de Java (JTA)
  - JDBC, si es necesario
    - Por ejemplo un bean de sesión envolviendo código legado
- El desarrollador explícitamente codifica (en un cliente o en un EJB)
  - El comienzo de la transacción
  - El completar una transacción o abortarla

Las interfaces de *Java Transaction*

- Java Transaction Service (JTS)
  - Conjunto de APIs de bajo nivel
  - No usadas por los desarrolladores de aplicación
    - Usado por los desarrolladores de gestores de transacción, servidores de aplicación, contenedores de EJBs, etc.
  - Integridad transaccional no garantizada
    - Garantizada a la aplicación por un contenedor
- Java Transaction API (JTA)
  - API de más alto nivel
  - Utilizada por los desarrolladores de aplicación
  - Especifica interfaces entre el gestor de transacciones y todos los objetos involucrados
    - Interfaz principal: **`UserTransaction`**

Definiciones

- Contexto transaccional
  - Define un ámbito transaccional y los objetos participantes y sus operaciones
  - Por defecto, se propaga entre objetos transaccionales como EJBs
- Objetos transaccionales
  - Objetos cuyos métodos son invocados en una transacción
  - Pueden ser asociados sólo con una transacción al tiempo
- Atributos transaccionales
  - Especiación por método en la gestión de transacción (mediante el contenedor) de la invocación
- Cliente transaccional
  - Un agente que invoca métodos sobre objetos transaccionales
- Gestor de transacción
  - Un agente que coordina el procesado de la transacción

CMT: Atributos transaccionales (1/4)

- Especificado a través de la anotación **`@TransactionAttribute`**
  - Como una anotación de método
  - Como una anotación de clase: se aplica a todos los métodos de la clase

  o a través de un descriptor de despliegue

- **`TransactionAttributeType`** admite 6 valores posibles
  1. **`Required`**
  2. **`RequiresNew`**
  3. **`NotSupported`**
  4. **`Supports`**
  5. **`Mandatory`**
  6. **`Never`**

– El valor por defecto (sin la anotación **`TransactionAttribute`** o descriptor de despliegue)
  - **`Required`**

Software de Comunicaciones

CMT: Atributos transaccionales (2/4)

- **`Required`** (valor por defecto):
  - Si el invocador tiene una contexto transaccional, se propaga al bean
  - Si no, el contenedor crea un nuevo contexto transaccional
  - Un método siempre se ejecuta dentro de un contexto transaccional
    - Pero no se crea una nueva transacción innecesariamente
  - Uso: Métodos que actualizan la base de datos u otro recurso que soporte transacciones
- **`RequiresNew`**:
  - Si un invocador tiene un contexto transaccional, se suspende durante la duración de la ejecución de ese método
  - En todo caso, se crea una nueva transacción
  - Se usa cuando no se quiere que un fallo en una transacción cause un fallo en una transacción más amplia

CMT: Atributos transaccionales (3/4)

- **`Supports`**:
  - Si un invocador tiene un contexto transaccional, se propaga al bean
  - Si no, no se usa el contexto de la transacción
  - Se usa para situaciones “no te preocupes” (uso “no te preocupes” cuidadosamente! variando el comportamiento transaccional) lo que puede ser
    - Por ejemplo un método llevando a cabo una operación simple de actualización
- **`NotSupported`**:
  - Si un invocador tiene un contexto transaccional, este se suspende durante la duración de la ejecución de ese método
  - Si no, no se usa un contexto transaccional
  - En todo caso, un método ejecuta sin un contexto transaccional
  - Ejemplo de uso: manejadores de recursos que no propagan la transacción

CMT: Atributos transaccionales (4/4)

- **`Mandatory`**:
  - Si un invocador tiene un contexto transaccional, se propaga al bean
  - Si no, se lanza una excepción
    - **`TransactionRequiredException`**
    - **`TransactionRequiredLocalException`**
  - Se usa cuando un invocador tiene que proveer la transacción
    - No necesariamente implica BMT o transacciones gestionadas por el cliente: un invocador puede ser un método diferente en el mismo EJB
- **`Never`**:
  - Si un invocador tiene contexto de transacción, se lanza una excepción
    - **`RemoteException`** o **`EJBException`**
  - Si no, un método procede normalmente, sin un contexto
  - Usado para recursos no-transaccionales

Software de Comunicaciones

CMT: Atributos transaccionales, resumen

Atributo Transaccional	Contexto de transacción del Invocador	Contexto transaccional del método de negocio
`Required`	T Ninguno	T Contexto nuevo creado por el contenedor
`RequiresNew`	T Ninguno	Contexto nuevo creado por el contenedor Contexto nuevo creado por el contenedor
`Supports`	T Ninguno	T Ninguno
`NotSupported`	T Ninguno	Ninguno Ninguno
`Mandatory`	T Ninguno	T Excepción
`Never`	T Ninguno	Excepción Ninguno

Ejemplo: Configurando atributos transaccionales en un descriptor de despliegue

```
// Fuente: Enterprise Java Beans 3.0, 5th edition, Burke et al.
<ejb-jar ...>
 ...
 <assembly-descriptor>
  ...
  <container-transaction>
    <method>
      <ejb-name>TravelAgentEJB</ejb-name>
      <method-name> * </method-name>
    </method>
    <trans-attribute>NotSupported</trans-attribute>
  </container-transaction>
  <container-transaction>
    <method>
       <ejb-name>TravelAgentEJB</ejb-name>
       <method-name>listAvailableCabins</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
  ...
 </assembly-descriptor>
 ...
</ejb-jar>
```

Software de Comunicaciones

Atributos transaccionales en entidades y beans dirigidos por evento

- En la recomendación de la especificación EJB3 relativa a entidades JPA
  - Los gestores de entidad deberían ser invocados dese una transacción JPA activa
    - No deberían utilizar las transacciones locales de JPA (**`EntityTransaction`** interface)
    - Algunas excepciones
  - Las entidades no usan atributos de transacción
- En los EJBs que manejan entidades persistentes
  - En CMT: Solamente se usan **`Required`**, **`RequiresNew`**, o **`Mandatory`**
    - Asegura todo acceso a la base de datos dentro de una transacción JTA
  - Nuevo contexto de persistencia creado si no existe otro ya
    - Caso usual: un gestor de entidad que tiene un contexto transaccional demarcado
    - Excepción: beans de sesión con estado y contexto de persistencia extendido
- En los beans dirigidos por mensajes
  - CMT: usan solamente **`NotSupported`** o **`Required`**
    - Porque otros tipos se aplican a transacciones iniciadas por el cliente (no hay cliente !!)

CMT

- Métodos de la clase **`EJBContext`** para usarlos solamente con CMT:
  - **`setRollBackOnly`**
    - Llamado por un método de negocio para instruir al contenedor para que deshaga la transacción
    - Usualmente llamado antes de lanzar una excepción
  - **`getRollBackOnly`**
    - Comprueba si el contenedor ha marcado la transacción en curso para que se deshaga
    - Evita ejecutar trabajo que podría no ser comprometido
- Beans de sesión con estado usando CMT
  - Pueden utilizar la interfaz **`SessionSynchronization`**

BMT

- Un método de negocio puede iniciar una nueva transacción
  - Declarar el uso de BMT a través de la anotación **`@TransactionManagement`**
  - Obtener un objeto **`UserTransaction`**
    - A través de **`EJBContext.getUserTransaction()`** o por inyección
  - Llamar a su método **`begin`** para asociar una transacción al hilo actual
- Objeto **`UserTransaction`**
  - Propagado a otros EJBs en la invocación a método
    - A no ser que se declare el uso de un BMT
  - Métodos:
    - **`begin`**, **`commit`**, **`rollback`**, **`setRollbackOnly`**,…
    - No hay método **`getRollBackOnly`**
- La instancia del bean que ha creado la transacción
  - Es responsable de cerrarla (compromiso o vuelta atrás)
    - Beans de sesión sin estado: el que inicia un método, lo finaliza
  - No puede comenzar una nueva transacción hasta que la previa se ha completado

Software de Comunicaciones

Ejemplo 6: Transacciones programáticas

```
[...]
InitialContext cxt = new InitialContext();
userTx = (javax.transaction.UserTransaction)
          cxt.lookup("java:comp/UserTransaction");

try{
    userTx.begin();
    beanA.setX(1);
    beanA.setName("uno");
    beanB.setY(2);
    beanB.setName("dos");
    userTx.commit();
} catch(Exception e){
   try{
     System.out.println(e.getMessage());
     userTx.rollback();
   } catch(Exception ex){}
}
[...]
```

Software de Comunicaciones

Beans de sesión con estado y la interfaz `SessionSynchronization`

- Beans de sesión con estado usando CMTs pueden recibir notificaciones de eventos de transacción
  - El bean puede sincronizar su estado con la base de datos
    - Permite al bean tener una “cache” antes de aplicar los cambios a la base de datos
  - Un bean simplemente implementa la interfaz **`SessionSynchronization`**
- **`SessionSynchronization`** tiene tres métodos:
  - **`afterBegin`**
    - Notifica al bean que una nueva transacción se ha iniciado
  - **`beforeCompletion`**
    - Notifica a la instancia del bean que una transacción está próxima a comprometerse (sirve para que el bean escriba datos en almacenamiento de forma persistente)
  - **`afterCompletion(boolean committed)`**
    - Notifica a un instancia de bean que un protocolo de compromiso de transacción se ha completado, y si se ha comprometido o ha sido vuelto atrás (si se ha deshecho, el bean no escribirá datos a la base de datos)

Beans de sesión con estado y contexto de persistencia extendido

- Contexto de persistencia transaccional (por defecto)
  - Una entidad desacoplada del contexto de persistencia después de la llamada a método
- Un bean de sesión con estado tiene estado conversacional
  - Los clientes recuperan múltiples entidades e interactúan con ellas a través de varias invocaciones
  - Requiere que las entidades cargadas permanezcan gestionadas (no desacopladas) entre llamadas a método
  - Solución: contexto de persistencia extendido (finaliza cuando se elimina el bean)
- En el contexto de persistencia extendido
  - Se anotan las declaraciones del gestor de entidades cómo sigue

    ```
    @PersistenceContext(type=EXTENDED)
    ```
  - Puede invocar **`persist()`**, **`merge()`**, **`remove()`** fuera de una transacción
    - Inserciones, actualizaciones y borrados están encolados hacia el contexto de persistencia metidos en una lista
    - Puede justificar el uso de **`NotSupported`** en algunos métodos de EJBs con entidades

Aislamiento transaccional

- Ver transparencias JDBC para más información sobre los niveles de aislamiento de transacción
- Nivel de aislamiento más alto
  - Menos problemas de concurencia
  - Rendimiento más bajo
- Nivel de aislamiento por defecto de JPA: Compromiso de lectura
- CMT
  - El nivel de asilamiento lo fija el desarrollador de una manera específica para el vendedor
- BMT: El nivel de aislamiento puede ser especificado desde EJB usando el API de la base da datos
  - JDBC: usando `Connection.setTransactionIsolation()`
- CMT y BMT
  - Pueden usar bloqueo programático: `EntityManager.lock()`

Bibliografía para la Sección 3

- *Mastering Enterprise JavaBeans 3.0.* Rima Patel Sriganesh, Gerald Brose, Micah Silverman. Wiley 2006.
  http://www.theserverside.com/tt/books/wiley/masteringEJB3/index.tss
- *Enterprise JavaBeans 3.0, 5th edition.* Bill Burke, Richard Monson-Haefel. OReilly 2006.
  http://proquest.safaribooksonline.com/059600978X

  Beginning EJB 3 application development: from novice to professional. Raghu Kodali, Jonathan Wetherbee, Peter Zadrozny. Apress 2006.
  http://books.google.com/books?id=xnbJkOIZfFgC
- *EJB 3 in Action.* Debu Panda, Reza Rahman, Derek Lane. Manning 2007
- *Enterprise JavaBeans Technology. Sun Microsystems.*
  http://java.sun.com/products/ejb/
- *JSR 220: Enterprise JavaBeans 3.0. The Java Community Process 2006.*
  http://jcp.org/en/jsr/detail?id=220
- *JBoss tutorials.* JBoss 2009
  http://www.jboss.org/ejb3/docs/

3. Apéndice: EJBs v2 (material no examinable)

Estructura de EJB 2.1 (1/2)

1. *La interfaz de la vista empresarial de un bean*: define
   - Interfaces remotas, para el acceso desde fuera del contenedor
     - Interfaz remota "home": métodos a nivel de clase
     - Interfaz remota de negocio: métodos del nivel de instancia
   - Interfaces locales, para el acceso desde dentro del contenedor
     - Interfaz local "home": métodos del nivel de clase
     - Interfaz local de negocio: métodos del nivel de instancia
2. *Clase del bean empresarial*: implementa
   - Métodos de negocio (a nivel instancia)
     - Llamado por un cliente (por ejemplo otro EJB) vía el API de la vista cliente
     - EJB 2: Métodos de negocio a nivel clase para beans de entidad
   - Métodos del ciclo-de-vida (a nivel clase)
     - Llamados por el contenedor
   - Otros métodos (a nivel instancia o nivel clase)
     - Llamados por la propia clase del bean

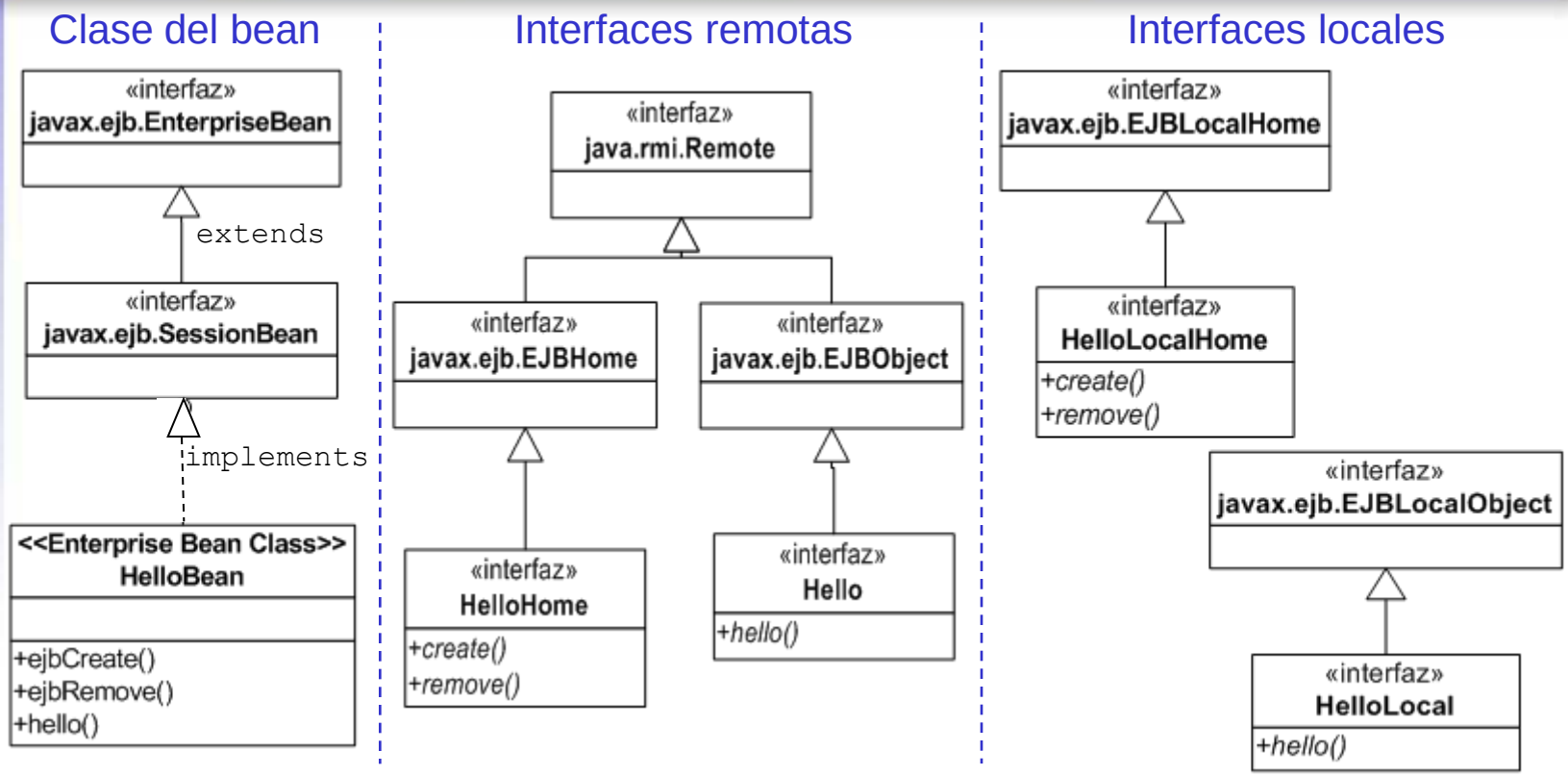
Estructura de EJB 2.1 (2/2)

3. *Descriptor de despliegue*: documento XML declarando:
   - Información sobre el :
     - Nombre del EJB
     - Nombre de la clase del EJB
     - Tipo de EJB
     - Nombre de las interfaces “home” y de negocio
   - Información sobre el entorno del EJB
     - Servicios que el EJB espera de su contenedor
     - Dependencias sobre otros EJBs y gestores de recursos
     - c.f. noción de “interfaces requeridas” en el desarrollo basado en componentes

Características de la vista-cliente

- Interfaces remotas
  - *Interfaz "home"* (el contenedor genera un objeto por clase)
    - Crea y elimina beans de sesión y entidad, busca beans de entidad
    - Extiende **`javax.ejb.EJBHome`**
  - *Interfaz remota de negocio* (el contenedor genera un representante)
    - Exporta métodos de negocio (remotamente)
    - Extiende **`javax.ejb.EJBObject`**
- Interfaces locales
  - *local home Interface* (el contenedor genera 1 objeto por clase)
    - Crea y destruye una sesión y beans de entidad, encuentra beans de entidad
    - Extiende **`javax.ejb.EJBLocalHome`**
  - *Interfaz local de negocio* (el contenedor genera un representante)
    - Exporta los métodos de negocio (localmente)
    - Extiende **`javax.ejb.EJBLocalObject`**

Ejemplo 7: `HelloWorldEJB` (EJB 2)



Deployment Descriptor

name=HelloWorldEJB
class=HelloBean
home=HelloHome
Type=Session
Transaction=Container
…

Software de Comunicaciones

Ejemplo 7: Interfaces remotas (EJB 2)

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;

public interface HelloHome extends javax.ejb.EJBHome {
  // Métodos de creación
  Hello create() throws RemoteException, CreateException;

}
```

```
import java.rmi.RemoteException;

public interface Hello extends javax.ejb.EJBObject {

  public String hello() throws RemoteException;

}
```

Software de Comunicaciones

Ejemplo 7: Interfaces Remotas (EJB 2)

```
import javax.ejb.CreateException;

public interface HelloLocalHome
                              extends javax.ejb.EJBLocalHome {
  // Métodos de creación
  HelloLocal create() throws CreateException;

}
```

```
public interface HelloLocal
                            extends javax.ejb.EJBLocalObject {

  public String hello();

}
```

Software de Comunicaciones

Ejemplo 7: Clase `HelloBean` (EJB 2)

```
import javax.ejb.RemoveException;
import javax.ejb.SessionContext;

//  javax.ejb.SessionBean conocido como la interfaz de componente
public class HelloBean implements javax.ejb.SessionBean {
   // Métodos del ciclo de vida declarados en la interfaz home
   public void ejbCreate() {...}

   // retrollamadas de la implementación del componente
   public void ejbRemove() throws RemoveException {...}
   public void setSessionContext(SessionContext sc) {...}
   public void ejbActivate() {...}
   public void ejbPassivate() {...}

   // Métodos de negocio declarados en interfaces de negocio
   public String hello() {
      return "Hello, World!";
   }
}
```

Software de Comunicaciones

Ejemplo 7: Descriptor de despliegue (EJB 2)

```
<ejb-jar
   xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
   version="2.1">
  <enterprise-beans>
   <session>
    <ejb-name>HelloWorldEJB</ejb-name>
    <home>examples.ejb21.HelloHome</home>
    <remote>examples.ejb21.Hello</remote>
    <local-home>examples.ejb21.HelloLocalHome</local-home>
    <local>examples.ejb21.HelloLocal</local>
    <ejb-class>examples.ejb21.HelloBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
   </session>
  </enterprise-bean>
</ejb-jar>
```

Software de Comunicaciones

EJB2 y EJB3, resumen de diferencias (1/2)

EJB2	EJB3
Interfaz de negocio • Debe extender una interfaz especial • *La conexión con el bean se hace en el descriptor* • Interfaces locales y remotas	*Interfaz de negocio* • Un objeto normal Java (POJI) • Conexión con la clase del bean hecha con la palabra clave Java **`implements`** (POJI) • Una interfaz simple con **`@Remote`** y/o una anotación **`@Local`**
Interfaz "home" • Para **`create()`**, **`find()`**, etc. •Debe extender una interfaz especial • Interfaces locales y o remotas	*Interfaz "home"* • No se necesita una interfaz "home"
Clase del Bean • Acceso al entorno de bean vía el servicio de búsqueda de JNDI (usualmente en ENC) • Ni herencia ni polimorfismo • Debe implementar la interfaz componente (muchos métodos de retro-llamada a menudo vacíos)	*Clase del Bean* •Acceso simplificado al entorno del bean a través de inyección de dependencia • Solamente una clase Java normal (POJO) • POJO

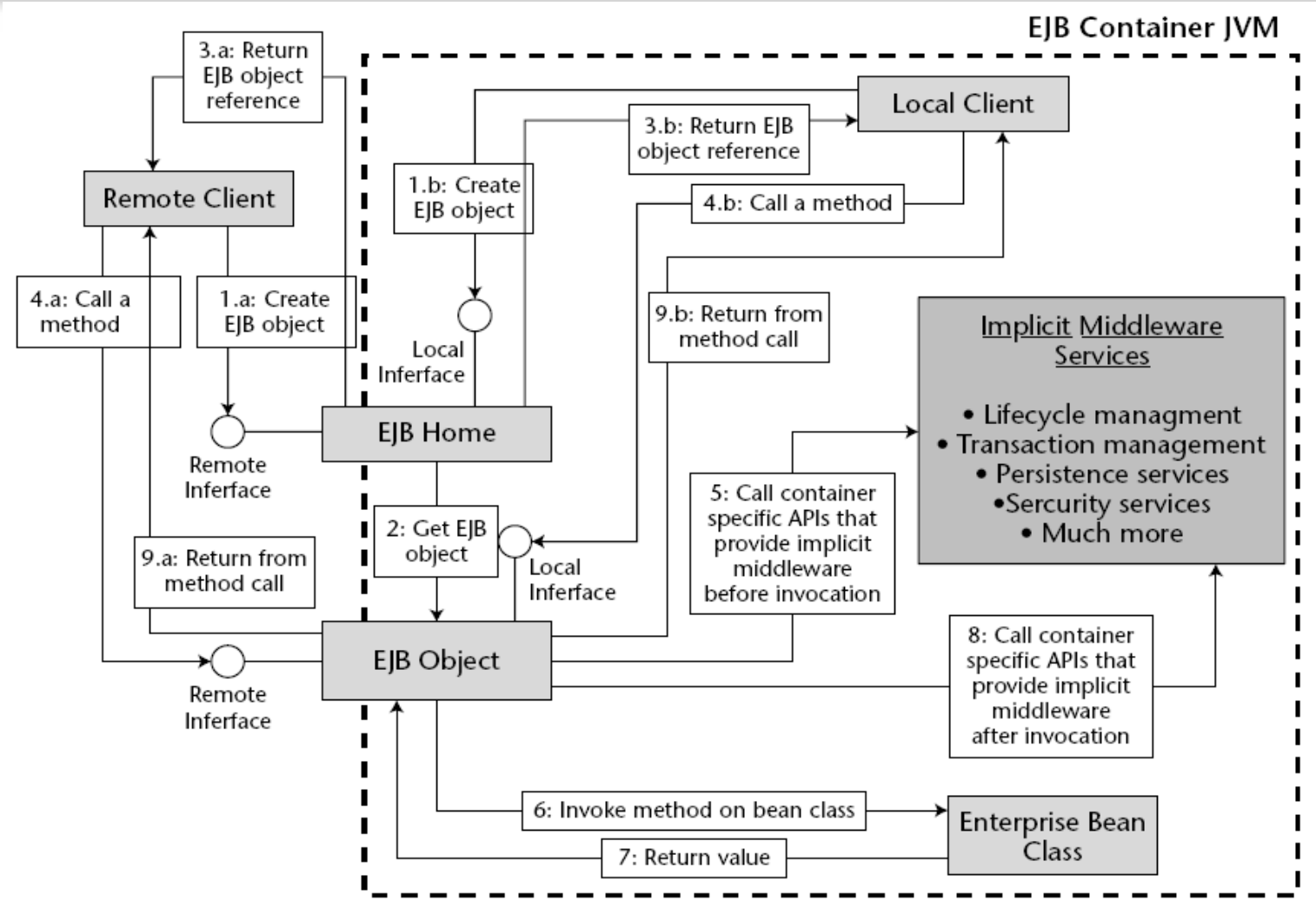
Software de Comunicaciones

EJB2 y EJB3, resumen de diferencias (2/2)

EJB2	EJB3
Descriptor XML de despliegue •Obligatorio para describir metadatos • cons: Complejo y verboso • pros: Todos los metadatos de aplicación en un archivo	*Descriptor XML de despliegue* •Se pueden utilizar anotaciones Java EE 5 • pros: anotaciones mucho más sencillas • pros: puede ser utilizado un descriptor de despliegue
Cliente del Bean • Necesita casting CORBA (`narrow`) • Acceso al bean vía búsqueda JNDI	*Cliente del Bean* • No se necesita el casting de CORBA • Acceso simplificado vía inyección de dependencias
Persistencia via beans de entidad •Beans de entidad BMP o CMP • Mucho mapeo a CMP implícito y no controlable por el desarrollador	*Persistencia vía JPA* • Entidades (POJOS) + `EntityManager` • Mapeo explícito y controlable por el desarrollador (pero con valores por defecto)
Pruebas • EJBs (incluído entity beans) no son probables fuera del contenedor	*Pruebas* • EJBs y entidades JPA probables fuera del contenedor (los beans son POJOs)

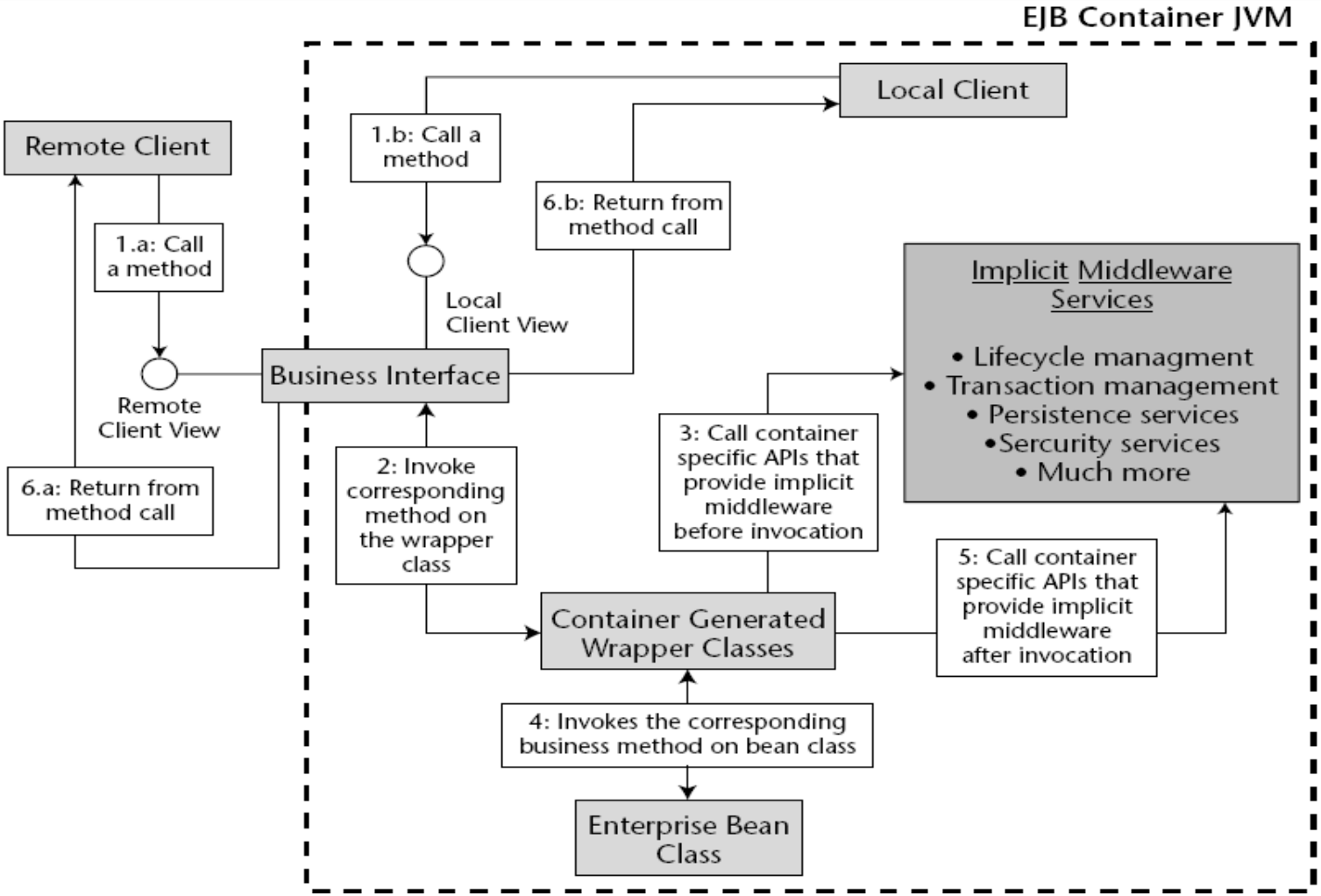
Software de Comunicaciones

El modelo de programación de EJB 2



Fuente: Mastering Enterprise JavaBeans 3.0, Patel et al.

El modelo de programa de EBJ 3



Fuente: Mastering Enterprise JavaBeans 3.0, Patel et al.

Software de Comunicaciones

Bibliografía para el apéndice

- *Enterprise JavaBeans, 4th edition.* Bill Burke, Richard Monson-Haefel. OReilly 2004.
  http://proquest.safaribooksonline.com/059600530X
- *Mastering Enterprise JavaBeans, 3rd edition.* Ed Roman, Rima Patel Sriganesh, Gerald Brose. Wiley 2005.
  http://www.theserverside.com/tt/books/wiley/masteringEJB/index.tss