Componentes Enterprise JavaBeans

Sesión 3 - Enterprise beans y JPA

Repasamos: entidad Autor

```java
package org.expertojava.ejb;

@Entity
public class Autor {
    @Id
    @GeneratedValue
    @Column(name = "autor_id")
    Long id;
    @Column(name="email", nullable = false, unique = true)
    private String correo;
    private String nombre;
    @OneToMany(mappedBy = "autor", cascade = CascadeType.ALL)
    private Set<Mensaje> mensajes = new HashSet<Mensaje>();

    public Long getId() { return id; }

    public String getCorreo() { return correo; }
    public void setCorreo(String correo) { this.correo = correo; }

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public Set<Mensaje> getMensajes() { return mensajes; }
    public void setMensajes(Set<Mensaje> mensajes) { this.mensajes = mensajes; }

    public Autor() {
    }
```

Mensaje

```java
@Entity
public class Mensaje {

    @Id
    @GeneratedValue
    @Column(name = "mensaje_id")
    private Long id;
    @Column(nullable = false)
    @Size(min = 3)
    private String texto;
    private Date fecha;
    @ManyToOne
    @JoinColumn(name = "autor", nullable = false)
    private Autor autor;

    public Long getId() { return id; }

    public String getTexto() { return texto; }
    public void setTexto(String texto) { this.texto = texto; }

    public Date getFecha() { return fecha; }
    public void setFecha(Date fecha) { this.fecha = fecha; }

    public Autor getAutor() { return autor; }
    public void setAutor(Autor autor) { this.autor = autor; }
```

persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="
        http://java.sun.com/xml/ns/persistence
        http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="mensajes">
        <jta-data-source>java:/datasources/MensajesDS</jta-data-source>
        <properties>
            <property name="hibernate.hbm2ddl.auto" value="create" />
            <property name="hibernate.show_sql" value="true" />
        </properties>
    </persistence-unit>
</persistence>
```

Repaso: transacciones con JTA

- JTA (Java Transaction API)
  - Conjunto de interfaces basadas en el estándar de transacciones distribuidas X/Open
  - La implementación la proporcionan servidores de aplicaciones
  - La clase más importante es javax.transaction.UserTransaction

```
void begin()
void commit()
int getStatus()
void rollback()
void setRollbackOnly()
void setTransactionTimeout (int seconds)
```

Uso de JTA

- Podemos usar JTA para gestionar transacciones que abarquen varias llamadas a beans de sesión
  - En los servlets
  - En los métodos de negocio de los enterprise beans (BMT: *Bean Managed Transaction*)
  - En general, en cualquier código desde el que podamos acceder a un UserTransaction gestionado por el servidor de aplicaciones

Programación de transacciones con JTA

- Pedimos un objeto `UserTransaction` al servidor de aplicaciones
- Lo utilizamos para demarcar la transacción
- Se puede usar en cualquier clase que se ejecuta en el servidor de aplicaciones

```
...
        try {
            tx.begin();
            autorServicio.nuevoAutor(correo, nombre);
            autorServicio.nuevoMensaje(texto, autor.getId());
            tx.commit();
        } catch (Exception e) {
            try {
                tx.rollback();
            } catch (SystemException e1) {
                e1.printStackTrace();
                throw new RuntimeException(e1);
            }
            throw new RuntimeException(e);
        }
    }
}
```

Utilizando JTA en los métodos de los beans: BMT

- BMT: Bean Managed Transactions

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class PedidoService {
    @Resource
    private UserTransaction userTransaction;

    public void hacerPedido(Item item, Cliente cliente) {
        try {
            userTransaction.begin();
            if (itemService.disponible(item)) {
                pedidoService.add(cliente,item);
                clienteService.anotaCargo(cliente,item);
                itemService.empaqueta(cliente,item);
            }
            userTransaction.commit();
        } catch (Exception e) {
            userTransaction.rollback();
            e.printStackTrace();
        }
    }
}
```

¿Cómo obtener el UserTransaction? (sólo BMT)

- Utilizando inyección de dependencias

```
@Resource UserTransaction utx;
```

- Obteniendo el contexto JNDI y buscando el objeto (debemos conocer su nombre JNDI local o global y es dependiente del servidor de aplicaciones)

```
InitialContext ic = new InitialContext();
UserTransaction userTransaction = (UserTransaction)
                  ic.lookup("UserTransaction");
```

- A través del contexto del EJB:

```
@Resource
private EJBContext context;
...
context.getUserTransaction();
```

CMT en enterprise beans

- CMT: Container Managed Transactions
- Los métodos de los beans demarcan transacciones XA. Los recursos transaccionales que se utilicen dentro del método (bases de datos, colas de mensajes, otros EJB) se ejecutan automáticamente en un contexto transaccional.
- Cuando sucede un error, hay que marcar la transacción para rollback. El contenedor hará el rollback más adelante.
- Es posible definir de forma declarativa distintos tipos de gestión de la transacción por parte del EJB y de cada método.

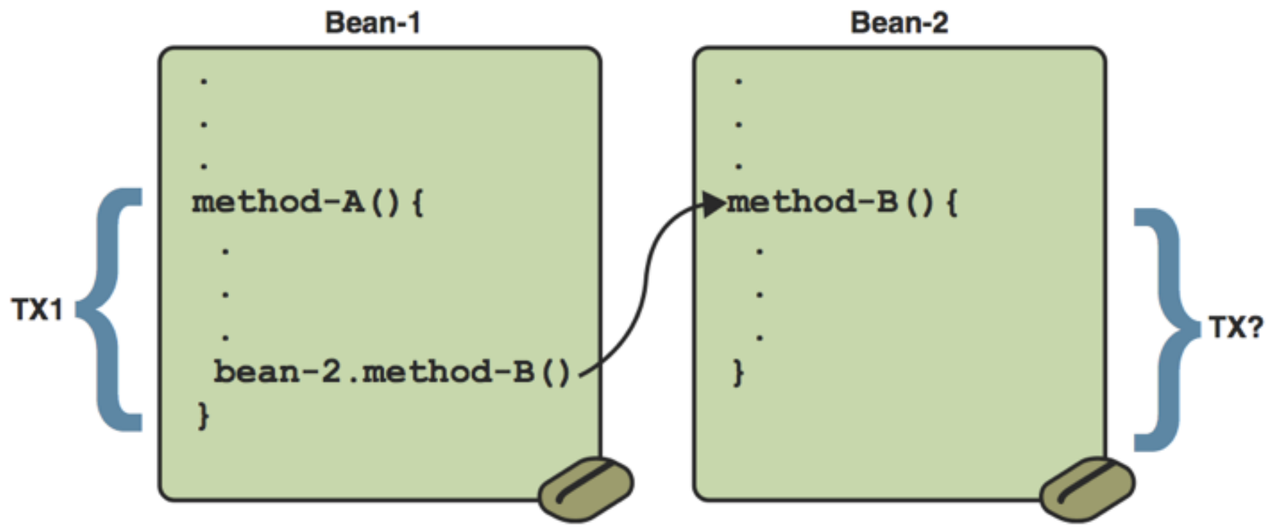
Ejemplo de método con CMT

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class PedidoService {

   @TransactionAttribute(TransactionAttributeType.REQUIRED)
   public void hacerPedido(Item item, Cliente cliente) {
      if (itemService.disponible(item)) {
         pedidoService.add(cliente,item);
         clienteService.anotaCargo(cliente,item);
         itemService.empaqueta(cliente,item);
      }
   }
}
```

Propagación de transacciones

- Cuando el metodo es llamado el contenedor generalmente hace una de las siguientes tres cosas:
  - Ejecutar el método en la transacción del llamador (caller)
  - Suspender la transacción del llamador e iniciar una nueva transacción
  - Lanzar una excepción porque el llamador no tiene una transacción



Propagación de transacciones

- El bean declara un atributo de transacción para cada método:
  - `TransactionAttributeType.REQUIRED`
  - `TransactionAttributeType.REQUIRES_NEW`
  - `TransactionAttributeType.SUPPORTS`
  - `TransactionAttributeType.MANDATORY`
  - `TransactionAttributeType.NOT_SUPPORTED`
  - `TransactionAttributeType.NEVER`

Atributo Required

- Si el llamador tiene abierta la transacción A
  - El método se ejecuta dentro de la transacción A

- Si el llamador no tiene abierta ninguna transacción
  - El contenedor crea una transacción nueva

Atributo RequiresNew

- Si el llamador tiene abierta la transacción A
  - El contenedor suspende la transacción A hasta que el método se termina. El método se ejecuta en una nueva transacción B.

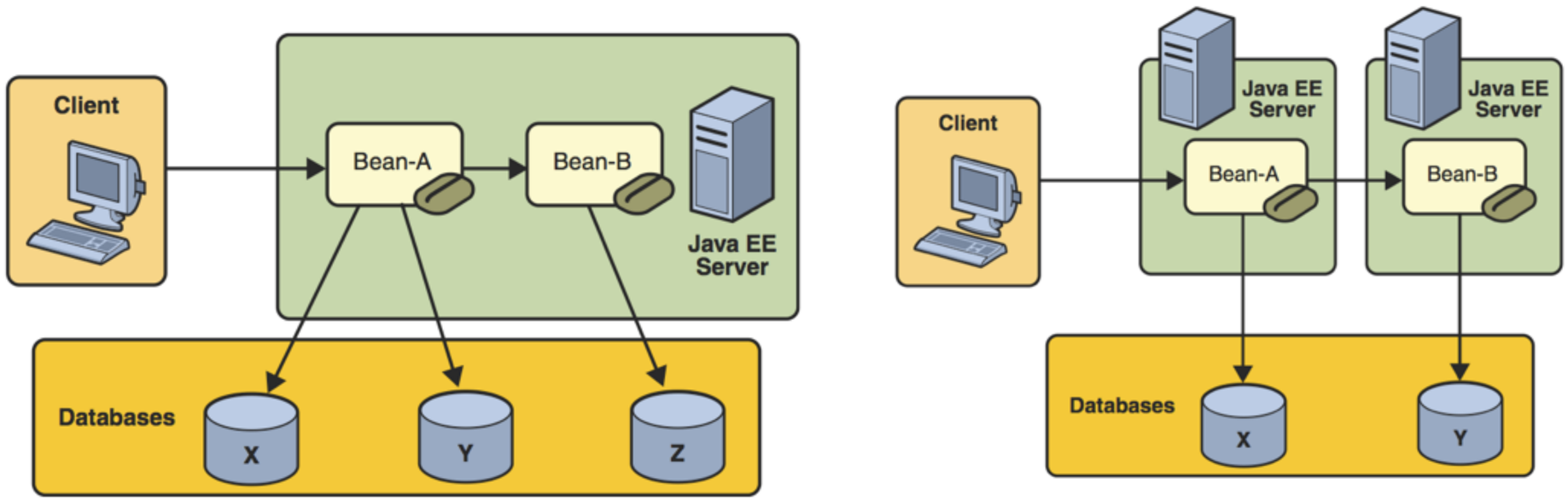
- Si el llamador no tiene abierta ninguna transacción
  - El contenedor crea una transacción nueva

Atributo Mandatory

- Si el llamador tiene abierta la transacción A
  - El método se ejecuta dentro de la transacción A

- Si el llamador no tiene abierta ninguna transacción
  - El contenedor lanza una excepción

Uso de transacciones en BD distribuidas



Repaso de JPA: Contexto de persistencia

- **Unidad de persistencia**: configuración concreta de un conjunto de entidades persistentes en un sistema de base de datos concreto, definida en el fichero `persistence.xml`. Existe de forma estática, definido por la BD.
- **Contexto de persistencia**: caché en memoria que contiene el conjunto de instancias de entidad gestionadas. Sus clases entidad son las definidas en la unidad de persistencia. Se crea al crear un entity manager y se desconecta al cerrarlo.
- **Entity manager**: responsable de la gestión del contexto de persistencia. Sincroniza con la BD las entidades incluidas en él. Se crea de forma dinámica.

JPA en EE

- Gestión de transacciones con JPA: no tenemos que abrir y cerrar la transacción en el código, el contenedor de EJBs es el que se encarga de hacerlo automáticamente
- Nos aprovechamos de la gestión de transacciones declarativas de los EJB
- Declaración de los entity managers con anotaciones
- Entity managers también gestionados por el contenedor EJB: no hay que crear ni cerrar los entity managers de forma explícita

Obtención de entity managers en beans de sesión

```java
@Stateless
public class AutorServicio {

    @Inject
    AutorDao autorDao;
    @Inject
    MensajeDao mensajeDao;

    public Autor createAutor(String nombre,
                             String correo) {
         Autor autor = new Autor(nombre, correo);
         autor = autorDao.create(autor);
        return autor;
    }
    ...
}
```

```java
public class AutorDao extends Dao<Autor, Long> {
    String FIND_ALL_AUTORES = "SELECT a FROM Autor a ";

    @Override
    public Autor find(Long id) {
        return em.find(Autor.class, id);
    }

    public List<Autor> listAllAutores() {
        Query query = em.createQuery(FIND_ALL_AUTORES);
        return (List<Autor>) query.getResultList();
    }
}
```

```java
abstract class Dao<T, K> {

    @PersistenceContext(unitName = "mensajes")
    EntityManager em;

    public T create(T t) {
        em.persist(t);
        em.flush();
        em.refresh(t);
        return t;
    }
    ...
}
```

Ámbito de vida de los contextos de persistencia

- ¿Cuándo cierra el contedor el entity manager?
- Los entity managers pasan a estar gestionados por el contenedor EJB
- Dos tipos de ámbito de vida del contexto de persistencia del entity manager:
  - **Transacción** (por defecto): se cierra cuando termina la transacción actual

```
@PersistenceContext(type = PersistenceContextType.TRANSACTION))
EntityManager em;
```

  - **Extendido**: se cierra cuando termina el ciclo de vida del componente en el que vive (se utilizan en beans con estado)

```
@PersistenceContext(type = PersistenceContextType.EXTENDED))
EntityManager em;
```

Entity managers con ámbito de transacción

- El contexto de persistencia se mantiene mientras que dura la transacción
- Distintos beans que participan en la misma transacción utilizan el mismo contexto de persistencia
- Se pueden pasar como parámetro y devolver como resultados entidades conectadas que son compartidas por los distintos EJBs que participan en la transacción

Servicios = BO que gestionan entidades

- Enfoque eficiente: los servicios (también llamados Busines Objects) trabajan con **entidades**
- El contexto de persistencia se comparte entre todas las operaciones que están en la misma transacción
- Automático en los beans con transacciones gestionadas por el contenedor
- En la capa web se pueden usar este enfoque utilizando JTA para crear la transacción que englobe los contextos de persistencia

Ejemplo - AutorServicio (1)

```
@Stateless
public class AutorServicio {

    @Inject
    AutorDao autorDao;
    @Inject
    MensajeDao mensajeDao;
    @PersistenceContext
    EntityManager em;

    public Autor createAutorMensajeCompuesto(String nombre,
                                        String correo,
                                        String texto) {
        System.out.println(String.format("En createAutorMensajeCompuesto [%s, %s, %s]",
                nombre, correo, texto));
        System.out.println("Entity manager: " + em.toString());

        Autor autor = this.createAutor(nombre, correo);
        Mensaje mensaje = this.createMensaje(autor, texto);

        System.out.println("Saliendo de createAutorMensajeCompuesto");
        return autor;
    }

   public Autor createAutor(String nombre, String correo) {
        System.out.println(String.format("En createAutor [%s, %s]", nombre, correo));

        Autor autor = new Autor(nombre, correo);
        autor = autorDao.create(autor);
        return autor;
    }
```

Ejemplo - AutorServicio

```
public Mensaje createMensaje(Autor autor, String texto) {
    System.out.println(String.format("En createMensaje [%s, %s]", autor.toString(), texto));
    Mensaje mensaje = new Mensaje(texto, autor);
    System.out.println("¿Entity manager contiene autor?: " + em.contains(autor));

    // El bean validation provocará una excepción
    // si el mensaje tiene un tamaño de menos de 3 caracteres
    mensaje = mensajeDao.create(mensaje);

    // Otra forma de provocar una excepción: lanzándola nosotros.
    // Todas las excepciones RunTime hacen que se ejecute un rollback
    if (texto.equals("ERROR")) {
        throw new MensajesException(MensajesException.MENSAJE_ERROR);
    }
    return mensaje;
}
public List<Autor> listAllAutores() {
    System.out.println("Entrando en listAllAutores");
    System.out.println("Entity manager: " + em.toString());
    List<Autor> autores = autorDao.listAllAutores();
    System.out.println("Saliendo de listAllAutores");
    return autores;
}

public List<Mensaje> listAllMensajes() {
    System.out.println("Entrando en listAllMensajes");
    System.out.println("Entity manager: " + em.toString());
    List<Mensaje> mensajes = mensajeDao.listAllMensajes();
    System.out.println("Saliendo de listAllMensajes");
    return mensajes;
}
}
```

Test Arquillian

```java
@RunWith(Arquillian.class)
public class SaludoServicioTest {

    @EJB
    private AutorServicio autorServicio;

    @Deployment
    public static Archive<?> deployment() {
        return ShrinkWrap.create(WebArchive.class)
                .addPackage(Autor.class.getPackage())
                .addAsResource("META-INF/persistence.xml");
    }

    @Test
    public void deberiaDevolverNuevoAutor() {
        String correo = "pedro.picapiedra@gmail.com";
        String nombre = "Pedro Picapiedra";
        Autor autor = autorServicio.nuevoAutor(correo, nombre);
        assertTrue(autor.getCorreo().equals(correo) &&
                   autor.getNombre().equals(nombre));
    }
}
```

¿Preguntas?