Especialista en Aplicaciones y Servicios Web con Java Enterprise

Enterprise JavaBeans

Sesión 5:

Transacciones

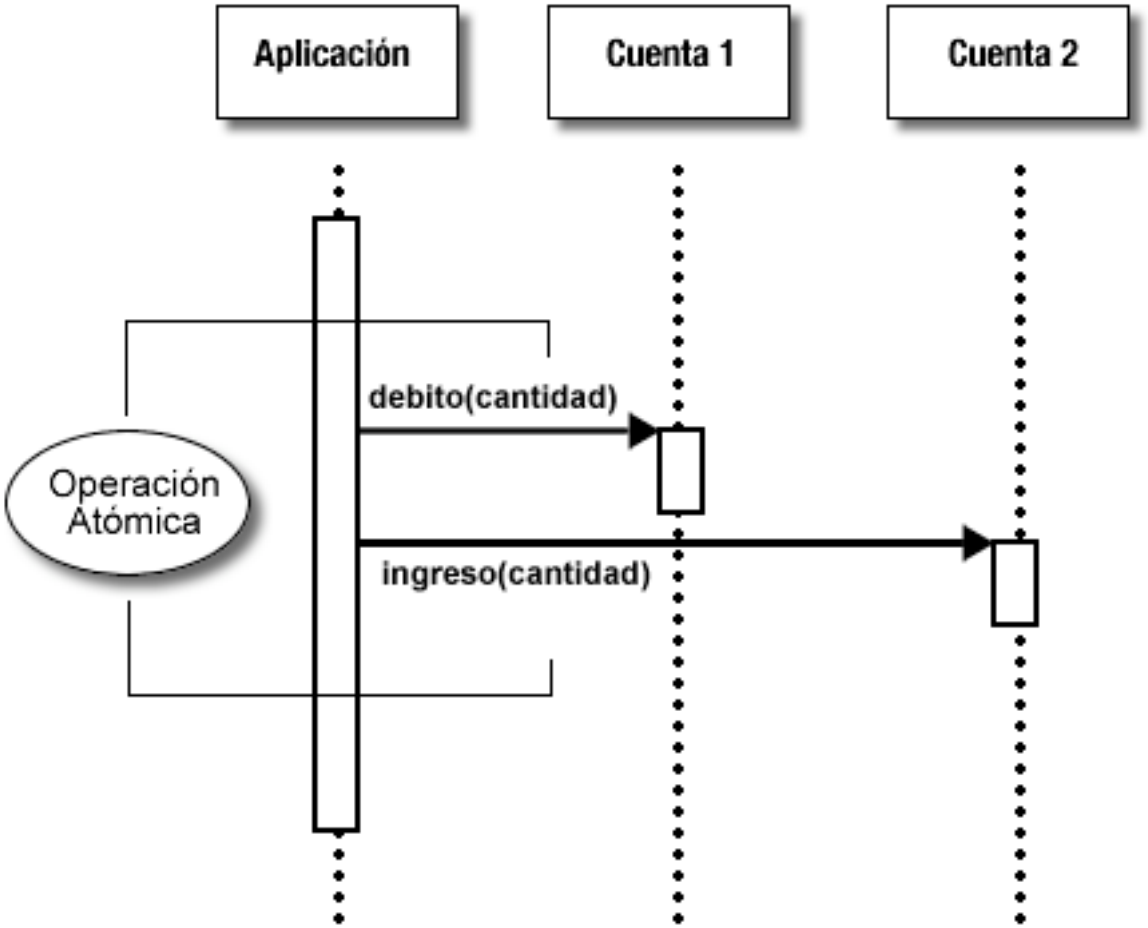
Definición de transacción

Una transacción es un grupo de operaciones que representa una unidad de trabajo y que deben ejecutarse como tal unidad.

Las operaciones se realizan sobre recursos (como lectura, escritura o actualización) y transforman un estado consistente del sistema en un nuevo estado consistente.

Una transacción debe ser ACID (Atomic, Consistent, Isolated and Durable).

Atomicidad: transferencia entre cuentas



Más carácterísticas (ACID)

- Consistencia
  - Responsabilidad del programador
- Aislamiento (o seriabilidad)
  - El resultado de la ejecución de un conjunto concurrente de transacciones debe ser equivalente a una ejecución en serie de las misma.
  - Los datos que participan en una transacción deben quedar ocultos (aislados) hasta el commit.
- Durabilidad

Alcance de las transacciones

- Transacciones locales
  - Usan un único gestor de recursos, por ejemplo una transacción que accede a una única base de datos o recurso.
- Transacciones globales
  - Son transacciones que usan un único gestor de transacciones y más de una gestor de recursos. Se debe usar el protocolo two-phase commit para realizar un commit de la transacción.
- Transacciones distribuidas
  - Son transacciones en las que intervienen más de un gestor de transacciones. Al igual que en las transacciones globales, se usa también el protocolo two-phase-commit para asegurar la correcta ejecución de la transacción.

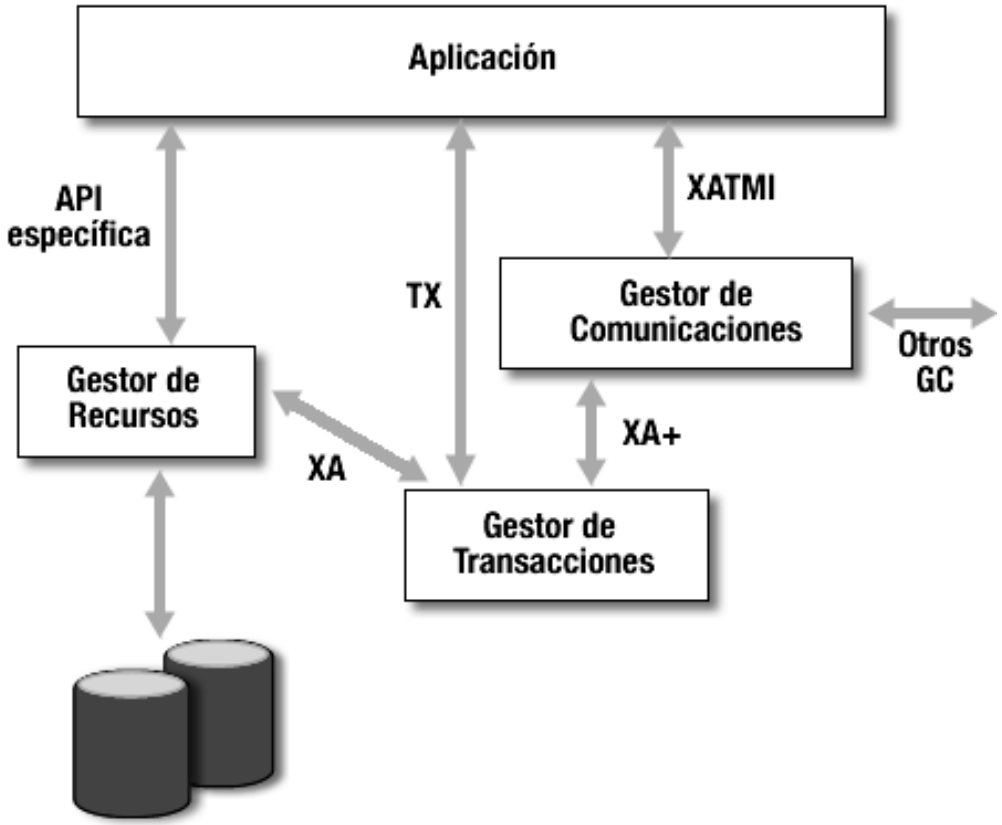
TP Monitor

- Un TP Monitor es una aplicación que realiza el control de las transacciones en una arquitectura de 3 capas
- Mercado maduro: existe desde los años 80
- Los servidores de aplicaciones son la siguiente evolución de los monitores de procesamiento de transacciones

Estándares en transacciones

- La existencia de unos estándares comunes garantizan la interoperabilidad entre distintos gestores de transacciones y gestores de recursos
- Estándares más comunes:
  - Modelo DTP (Distributed Transaction Processing) de X/Open
  - OTS (Object Transaction Service) que define los servicios de transacciones en CORBA

Modelo DTP de X/Open



Demarcación de transacciones

- En la mayor parte de los modelos de gestión de transacciones, éstas no son declarativas, sino que se notifican al gestor de transacciones según se van ejecutando.
- Cuando usamos un gestor de transacciones, para marcar el comienzo y el final de las transacciones se llama a los métodos begin y commit de un UserTransaction.

Ejemplo de demarcación de transacción

```
try {
   userTran.begin();                          // comienza la transaccion
   cliente.grabaPedido(pedido);                   // operacion 1
   tarjetaCredito = cliente.tarjetaCredito(); // operacion 2
   tarjetaCredito.carga(pedido.precio);           // operacion 3
   almacen.descuenta(pedido);                     // operacion 4
   userTran.commit();                         // fin de la transaccion
} catch (Exception e) {
   userTran.rollback();                       // se ha producido un error
}
```

JTA

- JTA (Java Transaction API)
  - Conjunto de interfaces basadas en el DTP de X/Open
  - La implementación la proporcionan servidores de aplicaciones

Interfaces JTA

Módulo responsable de la implementación	Interfaces
Gestor de transacciones	• `javax.transaction.Status` • `javax.transaction.Transaction` • `javax.transaction.TransactionManager` • `javax.transaction.UserTransaction`
Gestor de recursos	• `javax.transaction.xa.Xid` • `javax.transaction.xa.XAResource`
Aplicación	• `javax.transaction.Synchronization`

javax.transaction.Status

- STATUS_ACTIVE
- STATUS_COMMITED
- STATUS_COMMITTING
- STATUS_MARKED_ROLLBACK
- STATUS_PREPARED
- STATUS_PREPARING
- STATUS_ROLLEDBAC

javax.transaction.UserTransaction

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

Programación de transacciones con JTA

```
Context ctx = getInitialContext();
UserTransaction utx = (UserTransaction)
  ctx.lookup("javax.transaction.UserTransaction");
utx.begin();
try {
   utx.begin();
   cliente.grabaPedido(pedido);
   tarjetaCredito = cliente.tarjetaCredito();
   tarjetaCredito.carga(pedido.precio);
   almacen.descuenta(pedido);
   utx.commit();
} catch (Exception e) {
   utx.rollback();
}
```

Transacciones en EJB

- Los métodos de los EJB 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.
- Si se produce una excepción EJBException dentro del método del bean, se ejecuta automáticamente un rollback y la transacción se anula. Si el método termina correctamente, la transacción se confirma.
- Es posible definir de forma declarativa distintos tipos de gestión de la transacción por parte del EJB y de cada método.

Ejemplo de método EJB

```
public TicketTO reservaPasaje(CreditCardTO tarjeta, double precio)
  throws EstadoConversacionalIncompleto {

  if (cliente == null || crucero == null || cabina == null) {
    throw new EstadoConversacionalIncompleto();
  }
  try {
    Reservas reservas =  resHome.create();
    PagosTarjeta pagos = ppHome.create();
    reservas.hazReserva(cliente,crucero,camarote,precio);
    pagos.procesaPago(cliente,tarjeta,precio);
    TicketTO ticket = new TicketTO(cliente,crucero,
                                   camarote,precio;
    return ticket;

  } catch(Exception e) {
      throw new EJBException(e);
  }
}
```

Clientes remotos

- El cliente remoto del EJB puede crear una transacción antes de llamar al bean:

```
UserTransaction utc =
(UserTransaction) jndiContext.lookup(
     "javax.transaction.UserTransaction");
utc.begin();
book.setPrice(precio);
...
utc.commit();
```

Propagación de transacciones

- El bean declara un atributo de transacción para cada método:
  - Required
  - RequiresNew
  - Mandatory
  - Supports
  - NotSupported
  - Never

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

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

Anotaciones WebLogic

- En WebLogic 9.2 se usan anotaciones EJBGen para simplificar el desarrollo del componente EJB:

```
@RemoteMethod(transactionAttribute =
      Constants.TransactionAttribute.REQUIRED)
  public void pruebaTransaccion(String login, Boolean excepcion) {
      ...
  }
```

- Posibles valores del atributo:

```
Constants.TransactionAttribute.UNSPECIFIED
Constants.TransactionAttribute.NOT_SUPPORTED
Constants.TransactionAttribute.SUPPORTS
Constants.TransactionAttribute.REQUIRED
Constants.TransactionAttribute.REQUIRES_NEW
Constants.TransactionAttribute.MANDATORY
Constants.TransactionAttribute.NEVER
```

Transacciones y concurrencia

- La concurrencia añade un elemento de complejidad adicional a la gestión de transacciones
- Supongamos las siguientes operaciones. Si se ejecutan concurrentemente 2 clientes es posible que X quede en un estado inconsistente.

Cliente 1

1. **Leer el dato X**
2. **Sumar 10 a X**
3. **Escribir el dato X**

Cliente 2

1. **Leer el dato X**
2. **Sumar 10 a X**
3. **Escribir el dato X**

Two-phase blocking

- Antes de usar un dato, éste se bloquea.
- Dos tipos de bloqueos: de lectura y de escritura.
- Un bloqueo de lectura entra en conflicto con otro de escritura. Un bloqueo de escritura entra en conflicto con otro de escritura.
- Una transacción puede obtener un dato si no se produce ningún conflicto. En caso de no poder obtener el dato, queda en espera hasta que el dato se libera.

Ejemplo

Cliente 1	Cliente 2
1. Bloqueo de escritura sobre X	1. Bloqueo de escritura sobre X
2. Leer el dato X	2. Leer el dato X
3. Sumar 10 a X	3. Sumar 10 a X
4. Escribir el dato X	4. Escribir el dato X
5. Liberar X	5. Liberar X

Control optimista

Cliente 1	Cliente 2
1. Clonar X, guardando el número de versión	1. Clonar X, guardando el número de versión
2. Sumar 10 a X	2. Sumar 10 a X
3. Si el número de versión de X es el mismo que el del clon: escribirlo	3. Si el número de versión de X es el mismo que el del clon: escribirlo
4. Sino: abortar la transacción	4. Sino: abortar la transacción

¿Preguntas?