Los Enterprise JavaBeans de sesión con estado

Índice

1. Implementación de los beans de sesión con estado

Vamos a construir ahora un ejemplo completo de bean de sesión con estado. En concreto, vamos a implementar el bean, `CartBean`, que define un carrito de la compra de una librería on-line. El cliente del bean podrá añadir un libro al carrito, eliminar un libro o consultar el contenido del carrito. Tal y como hemos visto en las sesiones anteriores, para construir el enterprise bean `CartBean`, se necesitan los siguientes ficheros:

- Clase de implementación bean (`CartBean`)
- Interface home (`CartHome`)
- Interface componente (`Cart`)

El estado del bean va a estar definido por sus variables de instancias. Estas variables de instancia deben declararse en la clase `CartBean` que implementa la lógica de negocio, y serán actualizadas por los métodos del bean, como consencuencia de la invocación del cliente.

Además de estas clases, vamos a usar las clases adicionales `BookException` con la que definimos excepciones relacionadas con la aplicación y `BookVO` que implementa un *value object* y sirve para pasar la información entre las distintas capas de la aplicación.

Tal y como vimos en las sesiones anteriores, los métodos de las tres clases que definen el bean están relacionados. La siguiente tabla muestra un resumen con los métodos que debemos relacionar en estas clases. Recordemos que cada clase define los siguientes métodos:

- Interfaz componente: definición (sin implementar) de métodos de negocio,
- Interfaz home: definición (sin implementar) de los métodos de creación, busqueda y borrado de beans
- Clase bean: definición de la implementación de los métodos de negocio y de creación, búsqueda y borrado de beans.

	Interfaz home	Interfaz componente	Clase bean
Extiende o implementa la interfaz	Extiende `EJBHome` o `EJBLocalHome`	Extiende `EJBObject` o `EJBLocalObject`	Implementa `SessionBean`
Creación	`Cart create(args) throws CreateException, RemoteException;`	No disponible	`public void ejbCreate(args)`
Uso	No disponible	Métodos de negocio (deben arojar la	Métodos de negocio con la misma signatura

		excepción RemoteException)	(no deben declarar RemoteException: *las instancias de la clase SessionBean no son objetos remotos, lo son los "cortafuegos" EJBObject y EJBHome*)
Borrado	`public void remove();`	`public void remove()`	`public void ejbRemove();`

Vamos a definir una implemetación remota del bean. Para hacer una versión local bastaría con sustituir las clases que `EJBObject` y `EJBHome` por `EJBLocalObject` y `EJBLocalHome`.

El código fuente de este ejemplo está en el fichero `sesion03-ejercicios.zip`en la página web del módulo de EJB.

1.1. La clase bean de sesión

Siguiendo las indicaciones de nomenclatura que comentamos en la sesión pasada, la clase bean se llama `CartBean`. Esta clase implementa la lógica de negocio del bean, y debe cumplir los siguientes requisitos:

- Implementa la interfaz `SessionBean`.
- La clase se define como `public`.
- La clase no puede definirse como `abstract` ni `final`.
- Implementa uno o más métodos `ejbCreate`.
- Implementa los métodos remotos definidos en la interfaz componente del bean.
- Contiene un constructor `public` sin parámetros.

El código fuente de la clase `SessionBean` sigue a continuación:

```
package sesion3.beans;

import java.util.*;
import javax.ejb.*;

import sesion3.datos.BookVO;

public class CartBean implements SessionBean {

        private static final long serialVersionUID = 1L;
        String customerName;
        String customerId;
        Collection<BookVO> contents;
```

```
        double precio;

        public void ejbCreate(String person) throws CreateException {
                if (person == null) {
                        throw new CreateException("Null person not
allowed.");
                } else {
                        customerName = person;
                }
                customerId = "0";
                contents = new ArrayList<BookVO>();
                System.out.println("CartBean creado");
        }

        public void ejbCreate(String person, String id) throws
CreateException {
                if (person == null) {
                        throw new CreateException("Null person not
allowed.");
                } else {
                        customerName = person;
                }
                contents = new ArrayList<BookVO>();
                System.out.println("CartBean creado");
        }

        public void addBook(BookVO book) throws BookException {
                contents.add(book);
                System.out.println("Añadido libro: " +
book.getTitle());
        }

        public void removeBook(BookVO book) throws BookException {
                boolean result = false;
                String title = book.getTitle();
                Iterator it = contents.iterator();
                while (it.hasNext()) {
                        BookVO bookIt = (BookVO) it.next();
                        if (title.equals(bookIt.getTitle())) {
                                it.remove();
                                result = true;
                                System.out.println("Borrado libro: " +
book.getTitle());
                                break;
                        }
                }
                if (result == false) {
                        throw new BookException(title + " not in
cart.");
                }
        }

        public ArrayList<BookVO> getAllBooks() {
```

```
                return (ArrayList<BookVO>) contents;
        }

        public double getOrderPrice() {
                double price = 0.0;
                Iterator it = contents.iterator();
                while (it.hasNext()) {
                        BookVO book = (BookVO) it.next();
                        price = price + book.getPrice();
                }
                return price;
        }

        public CartBean() {
        }

        public void ejbRemove() {
                System.out.println("Objeto CartBean borrado");
        }

        public void ejbActivate() {
        }

        public void ejbPassivate() {
        }

        public void setSessionContext(SessionContext sc) {
        }
}
```

1.1.1. Las variables de instancia

Las variables de instancia definen el estado de una instancia de enterprise bean. En este caso son

- `String customerName`
- `String customerId`
- `Collection<BookVO> contents`

Las variables de instancia se inicializan al crearse una instancia del bean, copiándose en ellas los argumentos del método `ejbCreate`. Después son modificadas por las llamadas de los clientes a los métodos de negocio `addBook()` y `removeBook()`.

1.1.2. La interfaz SessionBean

La interfaz `SessionBean` extiende la interfaz `EnterpriseBean`, que a su vez extiende el interfaz `Serializable`. La interfaz `SessionBean` declara los métodos `ejbRemove`, `ejbActivate`, `ejbPassivate`, y `setSessionContext`. Estos métodos se llaman

desde el contenedor EJB cuando el bean va pasando de un estado a otro en su ciclo de vida. La clase `CartBean` no usa estos métodos, pero debe implementarlos porque están declarados en la interfaz `SessionBean`.

1.1.3. Los métodos ejbCreate

Como vimos en la sesión pasada un cliente no puede instanciar directamente un bean, ya que éste reside en el contenedor EJB. La forma de solicitar que se ejecuten métodos en del bean es a través de los objetos remotos `EJBObject` y `EJBHome`. En concreto, la petición de creación del bean se realiza mediante una llamada al objeto `EJBHome`, que a su vez la solicita al contenedor EJB. En el ejemplo del carrito de la compra esto sucede de la siguiente forma:

1. El cliente invoca un método `create` en el objeto `EJBHome`:

```
Cart shoppingCart = home.create("Domingo Gallardo","123");
```

2. El EJBHome solicita al contenedor EJB la creación de una instancia del enterprise bean. El contenedor EJB selecciona alguno de los beans sin usar de la reserva de beans.

3. El contenedor EJB invoca el método `ejbCreate` apropiado (puede haber más de un método ejbCreate, según los argumentos) en `CartBean`:

Al tratarse de un bean de sesión con estado, el método `ejbCreate` inicializa este estado.

Un enterprise bean debe tener uno o más métodos `ejbCreate`. Las signaturas de estos métodos deben cumplir los siguientes requisitos:

- El modificador de control de acceso debe ser `public`.
- El tipo de retorno debe ser `void`.
- Si el bean permite un acceso remoto, los argumentos deben ser tipos legales de RMI.
- El modificador no puede ser `static` ni final.

La clásula throws puede incluir la excepción `javax.ejb.CreateException` y cualquier otra excepción específica de la aplicación. El método `ejbCreate` arroja normalmente una excepción `CreateException` si un parámetro de entrada no es válido.

Es posible definir más de un método ejbCreate en la clase bean si quieres tener más de una forma de crear el bean, cada una con distintos tipos de argumentos. Eso sí, también habría que definir los mismos métodos en la interfaz home.

1.1.4. Métodos de negocio

El propósito fundamental de un bean de sesión es ejecutar tareas de negocio a petición de aplicaciones cliente. El cliente invoca métodos de negocio en la referencia remota del objeto que es devuelta por el método `create`. Recuerda que esta referencia es una instancia de un stub que implementa la interfaz componente.

Desde la perspectiva del cliente, los métodos de negocio parecen ejecutarse de forma local, pero realmente lo que se ejecuta en el cliente es un método del stub de que envía la petición al EJBObject que reside en el contenedor EJB.

Los parámetros de los métodos de negocio son normalmente *value objetcs*, objetos Java puros (no son objetos remotos) que contienen la información necesaria para realizar la función. En nuestro caso se trata de objetos de la clase `BookVO` que contienen el título y el precio de un libro.

Un ejemplo del punto de vista del cliente:

```
Cart shoppingCart = home.create("Domingo Gallardo", "123");
BookVO book = new BookVO("Yo robot", 22.30);
shoppingCart.addBook(book);
...

// Imprimo los libros guardados hasta el momento
Collection<BookVO> lista = shoppingCart.getAllBooks();
Iterator it = bookList.iterator();
while (it.hasNext()) {
   BookVO book = (BookVO) it.next();
   System.out.print(book.getTitle());
   System.out.println(" "+book.getPrice());
}
```

La signatura de un método de negocio debe cumplir los siguientes requisitos:

- El nombre del método no debe entrar en conflicto con ningún otro definido por la arquitectura EJB. Por ejemplo, un método de negocio no se puede llamar `ejbCreate` o `ejbActivate`.
- El modificador de control de acceso debe ser `public`.
- Si el bean permite acceso remoto, los argumentos y los tipos devueltos deben ser tipos legales de RMI.
- El modificador no puede ser `static` ni `final`.

La cláusula `throws` puede incluir excepciones que se definen en la aplicación. Por ejemplo, el método `removeBook` arroja la excepción `BookException` si el libro no está en el carrito.

Para indicar un problema de nivel de sistema, como la imposibilidad de conectar con una base de datos, un método de negocio debería arrojar una excepción

`javax.ejb.EJBException`. Cuando un método de negocio arroja una excepción `EJBException`, el contenedor EJB la envuelve en una `RemoteException`, que es capturada por el cliente. El contenedor no envuelve excepciones de aplicación como `BookException`. Debido a que `EJBException` es una subclase de `RuntimeException`, no es necesario incluirla en la cláusula `throws` del método de negocio.

1.2. La interfaz home

La interfaz home extiende la interfaz `javax.ejb.EJBHome`. Para un bean de sesion, el propósito de la interfaz home es definir los métodos `create()` (puede haber más de uno, dependiendo de los argumentos) que el cliente remoto puede invocar para crear instancias del bean. El cliente `CartCliente`, por ejemplo, invoca este método `create`

```
Cart shoppingCart = home.create("Domingo Gallardo", "123");
```

Por cada método create en la interfaz home debe existir un método `ejbCreate` correspondiente en la clase de implementación del bean (clase `CartBean`). En nuestro ejemplo, definíamos dos signaturas de métodos `ejbCreate`:

```
public void ejbCreate(String person) throws CreateException
...
public void ejbCreate(String person, String id)  throws
CreateException
```

Y el código correspondiente en el fichero `CartHome.java` es el siguiente:

```
package sesion3.beans;

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

public interface CartHome extends EJBHome {

    public Cart create(String person) throws RemoteException,
CreateException;
    public Cart create(String person, String id) throws
RemoteException,
                                                CreateException;
}
```

Las signaturas de ambos conjuntos de métodos son similares, pero difieren en aspectos importantes. Un resumen de las reglas que debemos aplicar al crear ambas signaturas es el siguiente:

- El número y tipos de argumentos en el método create debe corresponderse con los definidos en el método `ejbCreate` asociado.
- Los tipos de los argumentos y valores devueltos deben ser tipos válidos RMI.
- Cuando usemos llamadas remotas, los métodos `create` de la interfaz home siempre debe devolver el tipo de la interfaz componente del bean. Cuando un cliente invoque este método se le devolverá una referencia remota del `EJBObject` del bean recién creado. El método `ejbCreate` de la clase bean, sin embargo, debe devolver `void`. Es el contenedor EJB el que se encarga de crear el bean, llamar al método `ejbCreate` y devolver la referencia remota.
- La cláusula `throws` del método `create` debe incluir las excepciones `java.rmi.RemoteException` y `java.ejb.CreateException`.

1.3. Interfaz componente

La interfaz componente siempre debe extender la interfaz `javax.ejb.EJBObject`. Esta interfaz define los métodos de negocio que el cliente remoto puede invocar. A continuación se muestra el código fuente del fichero `Cart.java`:

```
package sesion3.beans;

import java.util.*;
import javax.ejb.EJBObject;
import sesion3.datos.BookVO;

import java.rmi.RemoteException;

public interface Cart extends EJBObject {

   public void addBook(BookVO book) throws BookException,
RemoteException;
   public void removeBook(BookVO book) throws BookException,
RemoteException;
   public ArrayList<BookVO> getAllBooks() throws RemoteException;
   public double getOrderPrice() throws RemoteException;
}
```

La definición de los métodos en la interfaz componente debe seguir estas reglas:

- Cada método en la interfaz componente debe corresponderse con un método equivalente en la clase del enterprise bean. Las signaturas de ambos deben ser iguales.
- Los tipos de los argumentos y valores devueltos deben ser tipos válidos RMI.
- La cláusula `throws` debe incluir la excepción `java.rmi.RemoteException`.

1.4. Los ficheros descriptores de despliegue

El fichero `ejb-jar.xml` se muestra a continuación

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" 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">
    <enterprise-beans>
        <session>
            <ejb-name>CartBean</ejb-name>
            <home>sesion3.beans.CartHome</home>
            <remote>sesion3.beans.Cart</remote>
            <ejb-class>sesion3.beans.CartBean</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
</ejb-jar>
```

Y el fichero `weblogic-ejb-jar.xml` es el siguiente. Los únicos datos que se definen son el nombre del EJB y el nombre JNDI del bean.

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-ejb-jar xmlns="http://www.bea.com/ns/weblogic/90"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
http://www.bea.com/ns/weblogic/90/weblogic-ejb-jar.xsd">
    <weblogic-enterprise-bean>
        <ejb-name>CartBean</ejb-name>
        <jndi-name>CartBean</jndi-name>
    </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

1.5. Clases de apoyo

El bean de sesión `CartBean` tiene la clase de apoyo `BookException`. La excepción `BookException` es arrojada por el método `removeBook` cuando se intenta borrar del carrito un libro que no existe. Esta clase debe residir en el mismo directorio y fichero JAR en el que se reside el enterprise bean.

A continuación se muestran el código de la clase:

```
public class BookException extends Exception {

    public BookException() {
    }

    public BookException(String msg) {
        super(msg);
```

```
    }
}
```

La otra clase de apoyo es la que define el BookVO:

```
package sesion3.datos;

import java.io.Serializable;

public class BookVO implements Serializable{

    private static final long serialVersionUID = 1L;

    String title;
    double price;

    public BookVO(String title, double price) {
        this.title = title;
        this.price = price;
    }

    public boolean equals(BookVO another) {
        return (title.equals(another.title));
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getTitle(){
        return this.title;
    }

    public double getPrice() {
        return this.price;
    }
}
```