IS502052: Enterprise Systems Development Concepts

Lab 6: Stateless & Stateful

I. Introduction

In previous lab tutorial, we implemented a beginning EJB application by using stateless approach, which includes both server-side and client-side. In this lab tutorial, we will clarify the difference between ***stateless*** and ***stateful*** EJB, by modifying the last EJB application.

II. Stateless vs. Stateful

1. Definition

A stateless session bean is a type of enterprise bean, which is normally used to perform independent operations. A stateless session bean as per its name does not have any associated client state, but it may preserve its instance state. EJB Container normally creates a pool of few stateless bean's objects and use these objects to process client's request. Because of pool, instance variable values are not guaranteed to be same across lookups/method calls.

A stateful session bean is a type of enterprise bean, which preserve the conversational state with client. A stateful session bean as per its name keeps associated client state in its instance variables. EJB Container creates a separate stateful session bean to process client's each request. As soon as request scope is over, stateful session bean is destroyed.

2. What are stateless and stateful, really?

In the above definition, we can easily conclude what stateless and stateful are meaning, but do we really understand it. Since we must answer the important question, *"what is state?"*.

In network programming, we have the server-client model. In software development, a software contains two parts: the implementation and data; in which, the implementation is an instruction set to process the data. Therefore, if a software designed under server-client approach, the instruction set should lie on server-side. The client just sends the data, and then receives the results.

Stateless approach doesn't store the client's data on server. That means, when client sends data to server, and then, server completely processes the data, after that, the "relationship" between client and server is terminated and server doesn't store any data. Therefore, in EJB, **state** is corresponding to **data**.

Conversely, in stateful approach, server need to store the client's data, that means, the "relationship" between client and server will be maintained after processing the client's request. The data, stored on server-side, can be used (i) as an input for another process or (ii) for any purpose based on business logic.

III. Stateless EJB

1. Steps to create a stateless EJB

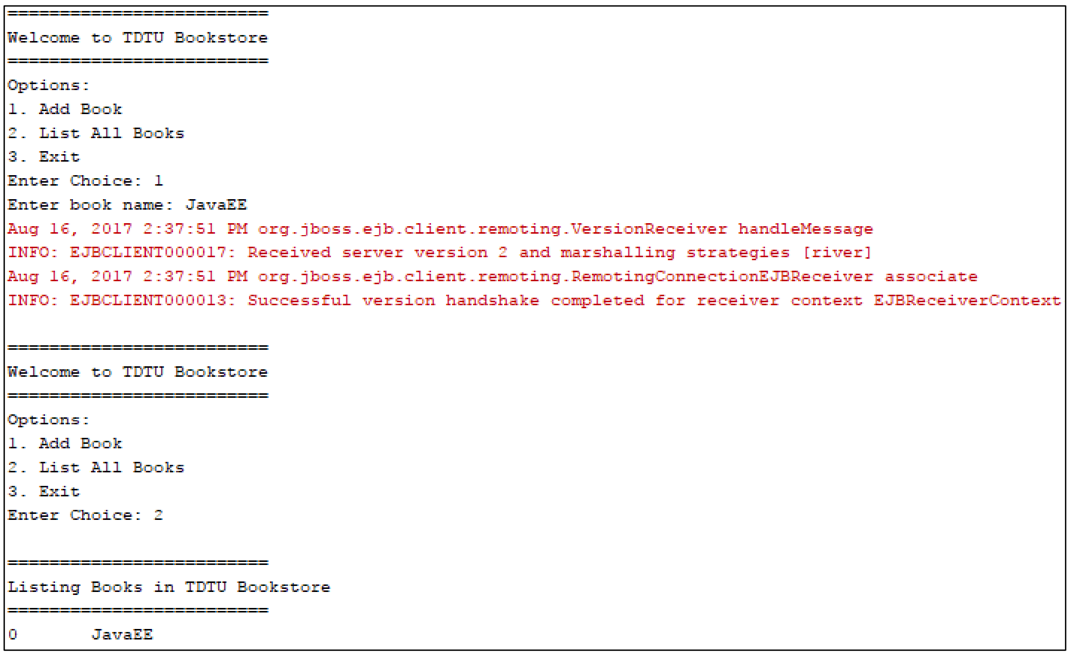
To create a stateless EJB in NetBeans, we should do the following steps:

- Create a remote/local interface exposing the business methods.
- This interface will be used by the EJB client application.
- Use *@Local* annotation, if EJB client is in same environment where EJB session bean is to be deployed.
- Use *@Remote* annotation, if EJB client is in different environment where EJB session bean is to be deployed.
- Create a stateless session bean, implementing the above interface.
- Use *@Stateless* annotation to signify it a stateless bean. EJB Container automatically creates the relevant configurations or interfaces required by reading this annotation during deployment.

Note that, the data (of Session Bean) is still maintained in the server until you turn off the server.

2. Sample application

To illustrate the stateless EJB, we should look back the sample application in previous lab tutorial. Now, you need to run this application again, and focus on the output, as in **Figure 1**.



```
==========================
Welcome to TDTU Bookstore
==========================
Options:
1. Add Book
2. List All Books
3. Exit
Enter Choice: 1
Enter book name: JavaEE
Aug 16, 2017 2:37:51 PM org.jboss.ejb.client.remoting.VersionReceiver handleMessage
INFO: EJBCLIENT000017: Received server version 2 and marshalling strategies [river]
Aug 16, 2017 2:37:51 PM org.jboss.ejb.client.remoting.RemotingConnectionEJBReceiver associate
INFO: EJBCLIENT000013: Successful version handshake completed for receiver context EJBReceiverContext

==========================
Welcome to TDTU Bookstore
==========================
Options:
1. Add Book
2. List All Books
3. Exit
Enter Choice: 2

==========================
Listing Books in TDTU Bookstore
==========================
0       JavaEE
```

Figure 1 Output of sample program

In this sample application, we have two main functions: (1) add new book, (2) list all books. For adding new books, we create a session bean, named *libBean*, to revoke methods on server-side. For listing all books, we create another session bean, named *libBean2*, which also connects to the same server.

Regardless the *libBean* or *libBean2*, we always get the same results from server. This is because we define stateless session beans, that means, the server doesn't store and process a specific data of current session bean.

IV. Stateful EJB

1. Steps to create Stateful EJB

Following are the steps required to create a stateful EJB:

- Create a remote/local interface exposing the business methods.
- This interface will be used by the EJB client application.
- Use *@Local* annotation if EJB client is in same environment where EJB session bean need to be deployed.
- Use *@Remote* annotation if EJB client is in different environment where EJB session bean need to be deployed.
- Create a stateful session bean, implementing the above interface.
- Use *@Stateful* annotation to signify it a stateful bean. EJB Container automatically creates the relevant configurations or interfaces required by reading this annotation during deployment.

2. Sample application

In this section, we will implement a stateful session bean application. For simplicity, we take advantage of the previous lab's application and modify some details to illustrate a stateful application.

Let's create a test EJB application to test stateful EJB, by following these steps:

- Create a project with a name ***SampleEJB2*** under a package ***edu.tdt.stateful*** as explained in previous lab.
- Create ***LibraryStatefulSessionBean.java*** and ***LibraryStatefulSessionBeanRemote.java*** as explained in last lab tutorial. Keep rest of the files unchanged.
- *Clean and Build* the application to make sure business logic is working as per the requirements.
- Then, *deploy* the application in the form of *jar* file on WildFly Application Server.
- Now create the EJB client, named ***EJBTester.java***, a console based application in the same way as explained previous lab.

Steps to create ***EJBTester.java*** are the same as last lab tutorial, but, the content is different, as shown below.

```
 package edu.tdt.test;


import edu.tdt.stateful.LibrarySessionBean;
import edu.tdt.stateful.LibrarySessionBeanRemote;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.Properties;
import java.util.Scanner;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

    private Properties props;
    private InitialContext ctx;

    public EJBTester()
    {
        readJNDI();
    }

    /**
     * Read the JNDI properties file
     */
    private void readJNDI()
    {
        props = new Properties();

        try
        {
            props.load(new FileInputStream("jndi.properties"));
        } catch(IOException e)
        {
            System.err.println(e.toString());
        }

        try
        {
            ctx = new InitialContext(props);
            //ctx.close();
        } catch (NamingException ex)
        {
            ex.printStackTrace();
        }
    }

    /**
     * Construct and return the JNDI address of called class
     * @return String
     */
    private String getJNDI()
    {
        String appName = "";
        String moduleName = "SampleEJB2";
        String distinctName = "";
        String sessionBeanName = LibrarySessionBean.class.getSimpleName();
        String viewClassName = LibrarySessionBeanRemote.class.getName() + "?stateful";

        return "ejb:"+appName+"/"+moduleName+"/"+distinctName+"/"+sessionBeanName+"!"+viewClassName;
    }

    /**
     * Show the GUI in console window
     */
    private void showGUI()
    {
        System.out.println("\n=========================");
        System.out.println("Welcome to TDTU Bookstore");
        System.out.println("=========================");
        System.out.print("Options: \n1. Add Book \n2. List All Books (Current Session) \n3. List All
Books (New Session) \n4. Exit \nEnter Choice: ");
    }

    /**
     * Declare a bean to invoke getBooks() method in LibrarySessionBeanRemote
     */
    private void getAllBooks()
    {
        try
        {
            // We can define another bean to access the LibrarySessionBeanRemote
            LibrarySessionBeanRemote libBean2 = (LibrarySessionBeanRemote) ctx.lookup(getJNDI());
            List<String> booksList = libBean2.getBooks();

            // Print all books
            if(booksList.isEmpty())
            {
                System.out.println("There is no book in the bookstrore!");
                return;
            }

            System.out.println("\n=========================");
            System.out.println("Listing Books in TDTU Bookstore");
            System.out.println("=========================");
            for (int i = 0; i < booksList.size(); i++)
            {
                System.out.println(i + "\t" + booksList.get(i));
            }
            System.out.println();

        } catch (NamingException ex)
        {
            ex.printStackTrace();
        }
    }

    /**
     * Test the Stateless EJB
     */
    public void testStatefulEJB()
    {
        try
        {
            // Scanner definition
            Scanner sc = new Scanner(System.in);

            // Lookup the LibrarySessionBeanRemote
            LibrarySessionBeanRemote libBean = (LibrarySessionBeanRemote) ctx.lookup(getJNDI());

            int choice = 0;
            while(choice != 4)
            {
                this.showGUI();

                // Use this approach to avoid the error cause by nextInt() follows by nextLine()
                choice = Integer.parseInt(sc.nextLine());

                if(choice == 1)
                {
                    // Add a book
                    System.out.print("Enter book name: ");
                    String bookName = sc.nextLine();
                    libBean.addBook(bookName);
                }
                else if(choice == 2)
                {
                    // Print all books (using current session bean)
                    List<String> booksList = libBean.getBooks();

                    if(booksList.isEmpty())
                    {
                        System.out.println("There is no book in the bookstrore!");
                    }

                    System.out.println("\n=========================");
                    System.out.println("Listing Books in TDTU Bookstore");
                    System.out.println("=========================");
                    for (int i = 0; i < booksList.size(); i++)
                    {
                        System.out.println(i + "\t" + booksList.get(i));
                    }
                    System.out.println();
                }
                else if(choice == 3)
                {
                    // Print all books (using new session bean)
                    getAllBooks();
                }
                else
                {
                    // Exit
                    break;
                }
            }

            sc.close();

        } catch (NamingException ex)
        {
            ex.printStackTrace();
        }
    }

}
```

In the above file, the highlighted functions are changed from the previous version. Let's see the differences of this new version.

The *showGUI()* method.

This method prints the application functions to users. Comparing with previous version, we add a new option, thus, the GUI is now having 4 options:

- Add book.
- List all books, using current session bean.
- List all books, using a new session bean.
- Exit application.

The *getJNDI()* method.

Two changes in this method are:

- The *moduleName* is *"SampleEJB2"*.
- The *viewClassName* must append *"?stateful"*, since we need to tell the WildFly Application Server that we are using stateful approach.

The *testStatefulEJB()* method.

There are some changes, in this method, as follows:

- As we add a new option, we need to modify the *choice* decision.
- In the second choice, we list all books in the *bookShelf* variable by using the current session bean, named *libBean*.
- In the third choice, we also list all books in the *bookShelf* variable by defining a new session bean, just call *getAllBooks()* method.

3. Running application

Let's run the application follows the steps:

- Enter **1** to add a new book.
- Enter **2** to list all books by using current session bean.
- Then, enter **3** to list all books by defining a new session bean.

Now, you should notice the difference between using the current and defining a new session bean.

The output console after running this application.

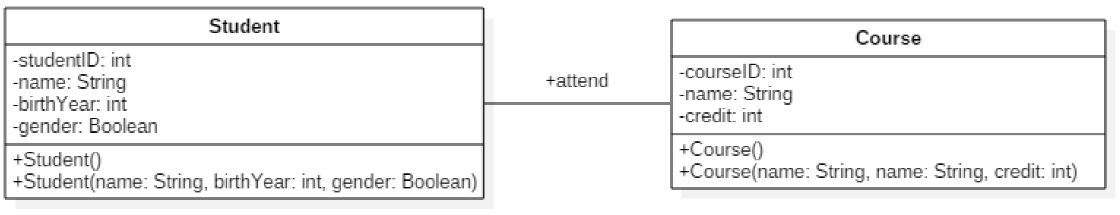
```
=========================
Welcome to TDTU Bookstore
=========================
Options:
1. Add Book
2. List All Books (Current Session)
3. List All Books (New Session)
4. Exit
Enter Choice: 1
Enter book name: Java EE

=========================
Welcome to TDTU Bookstore
=========================
Options:
1. Add Book
2. List All Books (Current Session)
3. List All Books (New Session)
4. Exit
Enter Choice: 2
```

```
=========================
Listing Books in TDTU Bookstore
=========================
0       Java EE


=========================
Welcome to TDTU Bookstore
=========================
Options:
1. Add Book
2. List All Books (Current Session)
3. List All Books (New Session)
4. Exit
Enter Choice: 3
There is no book in the bookstore!

=========================
Welcome to TDTU Bookstore
=========================
Options:
1. Add Book
2. List All Books (Current Session)
3. List All Books (New Session)
4. Exit
Enter Choice: 4

BUILD SUCCESSFUL (total time: 4 minutes 14 seconds)
```

From the above output, we can easily see that, when using the current session bean, we can get the name(s) of all books, since the stateful stores the data of current session. Therefore, when we create a new session bean, ***libBean2***, we can't get the same results, as we have not had any data in ***libBean2*** session.

V. Exercises

1. We are going to develop an EJB application simulating the student management system. The system is defined as below diagram:



Implementing the follows functions:

a. Pre-define a list of students and attended courses.
b. Client can create a new student and course.
c. Client can modify (insert, delete, update) a student's information and course's information.
d. Client can register a specific course for student.
e. Client can list all students, along with corresponding attended courses.
f. Client can list all courses, along with the list of students in those courses.

For simplicity, we don't consider the semester.