EJB - DEPENDENCY INJECTION

EJB 3.0 specification provides annotations which can be applied on fields or setter methods to inject dependencies. EJB Container uses the global JNDI registry to locate the dependency. Following annotations are used in EJB 3.0 for dependency injection.

- **@EJB** - used to inject other EJB reference.
- **@Resource** - used to inject datasource or singleton services like sessionContext, timerService etc.

Steps to use @EJB

@EJB can be used on fields or on methods in following manner.

```
public class LibraryMessageBean implements MessageListener {
   //dependency injection on field.
   @EJB
   LibraryPersistentBeanRemote libraryBean;
   ...
}
```

```
public class LibraryMessageBean implements MessageListener {

   LibraryPersistentBeanRemote libraryBean;

   //dependency injection on method.
   @EJB(beanName="com.tutorialspoint.stateless.LibraryPersistentBean")
   public void setLibraryPersistentBean(
   LibraryPersistentBeanRemote libraryBean)
   {
      this.libraryBean = libraryBean;
   }
   ...
}
```

Steps to use @Resource

@Resource is normally used to inject EJB Container provided singletons.

```
public class LibraryMessageBean implements MessageListener {
   @Resource
   private MessageDrivenContext mdctx;
   ...
}
```

Example Application

Let us create a test EJB application to test Dependency Injection Service in EJB.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.timer* as explained in the *EJB - Create Application* chapter.
3	Use Beans created in the *EJB - Message Driven Bean* chapter. Keep rest of the files unchanged.
5	Clean and Build the application to make sure business logic is working as per the requirements.

6	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
7	Now create the ejb client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent *EJBModule*

LibraryMessageBean.java

```
package com.tuturialspoint.messagebean;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import javax.annotation.Resource;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.EJB;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;

@MessageDriven(
   name = "BookMessageHandler",
   activationConfig = {
      @ActivationConfigProperty( propertyName = "destinationType",
                                 propertyValue = "javax.jms.Queue"),
      @ActivationConfigProperty( propertyName = "destination",
                                 propertyValue ="/queue/BookQueue")
   }
)
public class LibraryMessageBean implements MessageListener {

   @Resource
   private MessageDrivenContext mdctx;

   @EJB
   LibraryPersistentBeanRemote libraryBean;

   public LibraryMessageBean(){
   }

   public void onMessage(Message message) {
      ObjectMessage objectMessage = null;
      try {
         objectMessage = (ObjectMessage) message;
         Book book = (Book) objectMessage.getObject();
         libraryBean.addBook(book);

      } catch (JMSException ex) {
         mdctx.setRollbackOnly();
      }
   }
}
```

EJBTester *EJBClient*

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.entity.Book;
import com.tutorialspoint.stateless.LibraryPersistentBeanRemote;
import java.io.BufferedReader;
```

```
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Properties;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testMessageBeanEjb();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testMessageBeanEjb(){

      try {
         int choice = 1;
         Queue queue = (Queue) ctx.lookup("/queue/BookQueue");
         QueueConnectionFactory factory =
         (QueueConnectionFactory) ctx.lookup("ConnectionFactory");
         QueueConnection connection =  factory.createQueueConnection();
         QueueSession session = connection.createQueueSession(
         false, QueueSession.AUTO_ACKNOWLEDGE);
         QueueSender sender = session.createSender(queue);

         while (choice != 2) {
            String bookName;
            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
```

```
                Book book = new Book();
                book.setName(bookName);
                ObjectMessage objectMessage =
                session.createObjectMessage(book);
                sender.send(objectMessage);
            } else if (choice == 2) {
                break;
            }
        }

        LibraryPersistentBeanRemote libraryBean =
        (LibraryPersistentBeanRemote)
   ctx.lookup("LibraryPersistentBean/remote");

        List<Book> booksList = libraryBean.getBooks();

        System.out.println("Book(s) entered so far: "
        + booksList.size());
        int i = 0;
        for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            i++;
        }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testStatefulEjb method, jndi lookup is done with name - "/queue/BookQueue" to obtain treference of queue available in Jboss. Then sender is created using queue session.
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and sender sends the book name to queue. When JBoss container receives this message in queue, it calls our message driven bean's onMessage method. Our message driven bean then saves book using stateful session bean addBook method. Session Bean is persisting the book in database via EntityManager call.
- If user enters 2, then another jndi lookup is done with name - "LibraryStatefulSessionBean/remote" to obtain the remote business object *statefulejb* again and listing of books is done.

Run Client to access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
```

```
2. Exit
Enter Choice: 1
Enter book name: Learn EJB
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 2
1. learn java
1. learn EJB
BUILD SUCCESSFUL (total time: 15 seconds)
```

- Output shown above states that our Message driven bean is receiving the message and storing book in persistent storage and books are retrived from database.
- Our Message driven bean is using LibraryPersistentBean injected into it using @EJB annotation and in case of exception MessageDrivenContext object is used to rollback the transaction