ABOUT JGAURAVGUPTA

Gaurav is a senior software engineer with a passion for learning. He is an evangelist of netbeans & new technologies and author of JPA Modeler , jBatch Suite etc . He loves to go beyond the same old day to day work and find new and innovative ways to do the same things more effectively.

EJB Transaction Management Example

Posted by: jGauravGupta in Transactions April 22nd, 2015 0 4469 Views

1. Introduction

Transactions free the application programmer from dealing with the complex issues of failure recovery and multi-user programming.

The transactional system ensures that a unit of work either fully completes, or the work is fully rolled back.

2. Transaction Management Type in

specified. The Bean Provider of a session bean or a message-driven bean can use the

```
TransactionManagement
```

annotation to declare transaction type . The value of the

```
TransactionManagement
```

annotation is either

```
CONTAINER
```

or

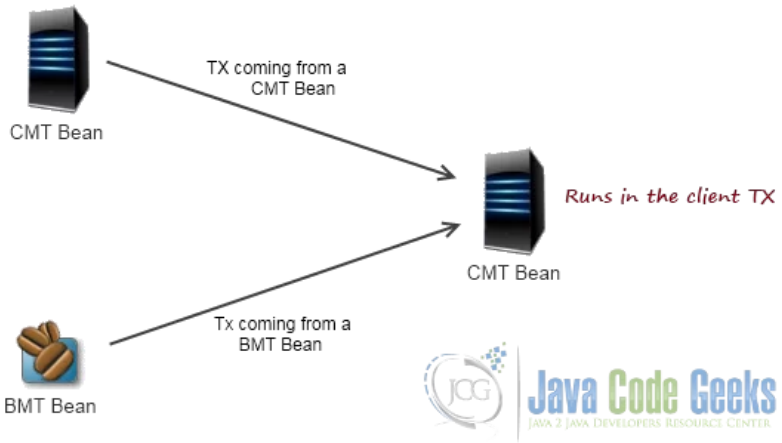
```
BEAN
```

.

3. Container Managed Transaction (CMT)

With container-managed transaction demarcation, the container demarcates transactions per instructions provided by the developer in metadata annotations or in the deployment descriptor.

With CMT, transactions are started and completed (with either a commit or rollback) by the container .



CMT Bean

- **SUPPORTS**
- **NEVER**

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Mandatory	None	TransactionRequiredException
	T1	T1
NotSupported	None	None
	T1	None
Supports	None	None
	T1	T1
Never	None	None
	T1	RemoteException

Transaction Attributes and Scope

A T1 transaction is associated with the client that calls a method in the enterprise bean and T2 transaction is started by the container just before the method executes. The word "None" means that the business method does not execute within a transaction controlled by the container.

3.1 Setting Transaction Attributes

Transaction attributes are specified by decorating the enterprise bean class or method with a

```
javax.ejb.TransactionAttribute
```

annotation and setting it to one of the

```
javax.ejb.TransactionAttributeType
```

constants.
By default, if a

```
TransactionAttribute
```

annotation is not specified for a method of an enterprise bean with container-managed transaction demarcation, the value of the transaction attribute for the method is defined to be

```
REQUIRED
```

is applied to all the business methods in the class. Decorating a business method with

`@TransactionAttribute`

applies the

`TransactionAttributeType`

only to that method. If a

`@TransactionAttribute`

annotation decorates both the class and the method, the method

`TransactionAttributeType`

overrides the class

`TransactionAttributeType`

.

The following code snippet demonstrates how to use the

`@TransactionAttribute`

annotation:

```
01  package com.javacodegeeks.example.beans;
02
03  import javax.ejb.Stateless;
04  import javax.ejb.TransactionAttribute;
05  import javax.ejb.TransactionAttributeType;
06
07  @Stateless
08  @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
09  public class SampleBean {
10      ...
11      @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
12      public void firstMethod() {...}
13
14      @TransactionAttribute(TransactionAttributeType.MANDATORY)
15      public void secondMethod() {...}
16
17      public void thirdMethod() {...}
18
19  }
```

In this example, the SampleBean class's transaction attribute has been set to

`NotSupported`

```
@TransactionAttribute
```

, calls to firstMethod will create a new transaction, and calls to secondMethod must use the transaction of the client. Calls to thirdMethod do not take place within a transaction.

3.2 Container-Managed Transaction Demarcation

Scope	Stateless	Stateful	Singleton	MessageDriven
Constructer,DI	UT	UT	UT	UT
PostConstruct	UT	TX[RN,NS]	TX[RE,RN,NS]	UT
PreDestroy	UT	TX[RN,NS]	TX[RE,RN,NS]	UT
PrePassivate	XX	TX[RN,NS]	XX	XX
PostActivate	XX	TX[RN,NS]	XX	XX
AfterBegin	XX	TX[RE,RN,MN]	XX	XX
BeforeCompletion	XX	TX[RE,RN,MN]	XX	XX
AfterCompletion	XX	UT	XX	XX
Bussiness Method	TX	TX	TX	TX[RE,NS]
Timeout Method	TX[RE,RN,NS]	XX	TX[RE,RN,NS]	TX[RE,RN,NS]
Asynchronous Method	TX[RE,RN,NS]	TX[RE,RN,NS]	TX[RE,RN,NS]	XX

UT	unspecified transaction context
TX	transaction context

RN	REQUIRES_NEW
NS	NOT_SUPPORTED
RE	REQUIRED
MN	MANDATORY

Transaction Context Scope

3.3 Rolling Back a Container-Managed Transaction

There are two ways to roll back a container-managed transaction. First, if a system exception is thrown, the container will automatically roll back the transaction. Second, by invoking the

```
setRollbackOnly
```

method of the

```
EJBContext
```

interface, the bean method instructs the container to roll back the transaction. If the bean throws an application exception, the rollback is not automatic but can be initiated by a call to

```
setRollbackOnly
```

.

3.4 Sample Scenario for Transaction Attribute to be In Action

is a good choice.

Never – A method should be annotated with

```
TransactionAttributeType.NEVER
```

if it only consist of logics that "NEVER" touches the database or any invocation of other methods which are transactional.

Not Supported – Better suited for methods that query objects which carries static data that won't be expected to be changed or to be transactionally involved with other business transaction. It can be querying methods for statically permanent data like country list, region list, gender list, etc. Methods that query data to especially establish drop-down list options in the selection box of web-forms are very well suited to be annotated with

```
NOT_SUPPORTED
```

. Annotating

```
NOT_SUPPORTED
```

in methods like these will greatly save applications from transaction overhead(s) .

3.5 Session Synchronization (Stateful Session bean transaction)

In the case of a

```
stateful
```

session bean, it is possible that the business method or interceptor method that started a transaction completes without committing or rolling back the transaction. In such a case, the container must retain the association between the transaction and the instance across multiple client calls until the instance commits or rolls back the transaction. When the client invokes the next business method, the container must invoke the business method in this transaction context.

If a session bean class implements the

```
javax.ejb.SessionSynchronization
```

interface or uses the session synchronization annotations, the container must invoke the

```
afterBegin
```

,

```
beforeCompletion
```

, and

```
afterCompletion
```

method to give the enterprise bean instance the last chance to cause the transaction to rollback. The instance may cause the transaction to roll back by invoking the

```
EJBContext.setRollbackOnly
```

method.

- The container invokes the

```
afterCompletion(boolean committed)
```

method after the completion of the transaction commit protocol to notify the enterprise bean instance of the transaction outcome.

CartBean.java

```
01 package com.javacodegeeks.example.beans;
02
03 import java.util.ArrayList;
04 import javax.annotation.PostConstruct;
05 import javax.annotation.PreDestroy;
06 import javax.ejb.AfterBegin;
07 import javax.ejb.AfterCompletion;
08 import javax.ejb.BeforeCompletion;
09 import javax.ejb.Remove;
10 import javax.ejb.Stateful;
11 import javax.ejb.TransactionAttribute;
12 import javax.ejb.TransactionAttributeType;
13 import javax.ejb.TransactionManagement;
14 import javax.ejb.TransactionManagementType;
15
16 /**
17  *
18  * @author jGauravGupta
19  */
20
21 @Stateful
22 @TransactionManagement(value=TransactionManagementType.CONTAINER)
23 public class CartBean  {
24     private ArrayList items;
25
26     @PostConstruct
27     public void init() {
28         items = new ArrayList();
29         System.out.println("CartBean: init");
30     }
31
32     @PreDestroy
33     public void destroy() {
34         System.out.println("CartBean: destroy");
35     }
36
37     @Remove
38     public void checkOut() {
39         // Release any resources.
40         System.out.println("Cart checkout...");
```

```
54          return items;
55      }
56
57      @AfterBegin
58      private void afterBegin(){
59          System.out.println("A new transaction has started.");
60      }
61
62      @BeforeCompletion
63      private void beforeCompletion(){
64          System.out.println("A transaction is about to be committed.");
65      }
66
67      @AfterCompletion
68      private void afterCompletion(boolean committed) {
69          System.out.println("a transaction commit protocol has completed, and tells the instance whether the
70      }
71
72  }
```

If the client request is not associated with a transaction

NO_TX_Client_Tester.java

```
01  package com.javacodegeeks.example.tester.non_tx;
02
03  import com.javacodegeeks.example.beans.CartBean;
04  import java.io.IOException;
05  import java.io.PrintWriter;
06  import java.util.logging.Level;
07  import java.util.logging.Logger;
08  import javax.naming.Context;
09  import javax.naming.InitialContext;
10  import javax.naming.NamingException;
11  import javax.servlet.ServletException;
12  import javax.servlet.annotation.WebServlet;
13  import javax.servlet.http.HttpServlet;
14  import javax.servlet.http.HttpServletRequest;
15  import javax.servlet.http.HttpServletResponse;
16
17  /**
18   *
19   * @author jGauravGupta
20   */
21  @WebServlet(name = "NO_TX_Client_Tester", urlPatterns = {"/NO_TX_Client_Tester"})
22  public class NO_TX_Client_Tester extends HttpServlet {
23
24      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
25              throws ServletException, IOException {
26
27          try (PrintWriter out = response.getWriter()) {
28
29              CartBean cartBean = lookupCartBeanBean();
30
31              cartBean.addItem("Smart Watch");
```

```
45          processRequest(request, response);
46      }
47
48      private CartBean lookupCartBeanBean() {
49          try {
50              Context c = new InitialContext();
51              return (CartBean) c.lookup("java:global/CMT_Example/CartBean!com.javacodegeeks.example.beans.Car
52          } catch (NamingException ne) {
53              Logger.getLogger(getClass().getName()).log(Level.SEVERE, "exception caught", ne);
54              throw new RuntimeException(ne);
55          }
56      }
57
58  }
```

Output

Verify the following output in NetBeans console :

```
01  Info:   A new transaction has started.
02  Info:   Smart Watch item added to cart
03  Info:   A transaction is about to be committed.
04  Info:   a transaction commit protocol has completed, and tells the instance whether the transaction has been
05  Info:   A new transaction has started.
06  Info:   iPhone item added to cart
07  Info:   A transaction is about to be committed.
08  Info:   a transaction commit protocol has completed, and tells the instance whether the transaction has been
09  Info:   A new transaction has started.
10  Info:   Shoes item added to cart
11  Info:   A transaction is about to be committed.
12  Info:   a transaction commit protocol has completed, and tells the instance whether the transaction has been
13  Info:   A new transaction has started.
14  Info:   Cart checkout...
```

If the client request is associated with a transaction

TX_Client_Tester.java

```
01  package com.javacodegeeks.example.tester.tx;
02
03  import com.javacodegeeks.example.beans.CartBean;
04  import java.util.logging.Level;
05  import java.util.logging.Logger;
06  import javax.annotation.Resource;
07  import javax.ejb.Singleton;
08  import javax.ejb.TransactionManagement;
09  import javax.ejb.TransactionManagementType;
10  import javax.naming.Context;
11  import javax.naming.InitialContext;
12  import javax.naming.NamingException;
13  import javax.transaction.HeuristicMixedException;
14  import javax.transaction.HeuristicRollbackException;
15  import javax.transaction.NotSupportedException;
```

```
    @Resource
    private UserTransaction ut;

    public void executeCartProcess() {
        try {
            Context c = new InitialContext();
            CartBean cartBean = (CartBean) c.lookup("java:global/CMT_Example/CartBean!com.javacodegeeks.exam

            ut.begin();
            cartBean.addItem("Smart Watch");
            cartBean.addItem("iPhone");
            cartBean.addItem("Shoes");

            System.out.println("Cart Item Size : " + cartBean.getItems().size());
            ut.commit();

            cartBean.checkOut();

        } catch (NamingException ex) {
            Logger.getLogger(CartProcess.class.getName()).log(Level.SEVERE, null, ex);
        } catch (RollbackException | HeuristicMixedException | HeuristicRollbackException | SecurityExceptio
            try {
                ut.rollback();
                Logger.getLogger(CartProcess.class.getName()).log(Level.SEVERE, null, ex);
            } catch (IllegalStateException | SecurityException | SystemException ex1) {
                Logger.getLogger(CartProcess.class.getName()).log(Level.SEVERE, null, ex1);
            }
        }
    }

}
```

Output

Verify the following output in NetBeans console :

```
Info:   CartBean: init
Info:   A new transaction has started.
Info:   Smart Watch item added to cart
Info:   iPhone item added to cart
Info:   Shoes item added to cart
Info:   Cart Item Size : 3
Info:   A transaction is about to be committed.
Info:   a transaction commit protocol has completed, and tells the instance whether the transaction has been
Info:   A new transaction has started.
Info:   Cart checkout...
Info:   CartBean: destroy
```

4. Bean Managed Transaction (BMT)

While its true that the ejb container is usually pretty smart about handling transactions, its also not as smart as a real human being and probably isn't able to handle complex database transactions and rollbacks. This is where bean managed transactions come in. By handling your own transactions you can avoid some major pitfalls.

interface or on the

`javax.jms.Session`

interface).

```
01  package com.javacodegeeks.example.beans;
02
03  import javax.annotation.Resource;
04  import javax.ejb.Stateless;
05  import javax.ejb.TransactionManagement;
06  import javax.ejb.TransactionManagementType;
07  import javax.transaction.UserTransaction;
08
09  @Stateless
10  @TransactionManagement(value=TransactionManagementType.BEAN)
11  public class AccountBean {
12
13     @Resource
14     private UserTransaction userTransaction;
15
16     public void withdrawAmount(long accountId , double fund) {
17
18        try{
19           userTransaction.begin();
20
21           // TO DO withdrawAmount ....
22
23           userTransaction.commit();
24        } catch (InsufficientFundException exception){
25           userTransaction.rollback();
26        }
27     }
28
29  }
```

In this example, we made use of

`UserTransaction`

interface to mark beginning of transaction using

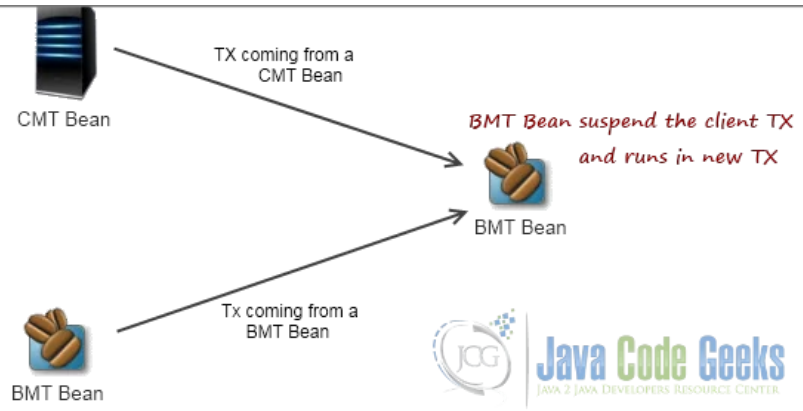
`userTransaction.begin()`

method call. We mark completion of transaction by using

`userTransaction.commit()`

method and if any exception occured during transaction then we rollback the complete transaction using

`userTransaction.rollback()`



BMT bean

The things that happen while the transaction is suspended won't be rolled back if the suspended transaction (after it comes back to life) fails to commit.

5. setRollbackOnly() lives in TWO interfaces

CMT beans can use only the

```
EJBContext.setRollbackOnly()
```

and BMT beans can use only the

```
UserTransaction.setRollbackOnly()
```

.

The CMT bean knows about the transaction's status using

```
EJBContext.getRollbackOnly()
```

method , If transaction marked for rollback then

```
getRollbackOnly()
```

method returns true and otherwise returns false.

The BMT bean knows about the transaction's status using

```
UserTransaction.getStatus()
```

6. Transactions Boundaries

6.1 JMS API

- The Bean Provider should not make use of the JMS request/reply paradigm (sending of a JMS message, followed by the synchronous receipt of a reply to that message) within a single transaction. Because a JMS message is typically not delivered to its final destination until the transaction commits, the receipt of the reply within the same transaction will not take place.
- A transaction starts before the dequeuing of the JMS message and, hence, before the invocation of the message-driven bean's onMessage method. If the onMessage method does not successfully complete or the transaction is rolled back, message redelivery semantics apply.

6.2 Asynchronous Method

The client's transaction context does not propagate with an asynchronous method invocation. The semantics of the

```
REQUIRED
```

transaction attribute for an asynchronous method are the same as

```
REQUIRES_NEW
```

.

6.3 Timing of Return Value Marshalling

When demarcating a container-managed transaction for a business method invocation through a remote view or web service view, the container must complete the commit protocol before marshalling the return value.

7. Download the NetBeans Project

Download the NetBeans project for this tutorial:

Download
You can download the full source code of this example here: **Transaction_Management_Example.zip**

8. Conclusion

BMT bean runs only in the transactions the bean itself creates and starts so that it defeats the whole point of a component model usability. With BMT, you can reduce the scope of a transaction but using CMT, you cannot mark a transaction at anything smaller than a single method.

Leave a Reply

Start the discussion...

Subscribe

KNOWLEDGE BASE

HALL OF FAME

ABOUT JAVA CODE GEEKS

THE CODE GEEKS NETWORK

- .NET Code Geeks
- Java Code Geeks
- System Code Geeks
- Web Code Geeks

- Java write to File Example
- java.io.FileNotFoundException – How to solve File Not Found Exception
- java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception
- java.lang.NoClassDefFoundError – How to solve No Class Def Found Error
- JSON Example With Jersey + Jackson
- Spring JdbcTemplate Example