Advertiser Disclosure

developer.com

Custom Search

April 10, 2019
Hot Topics:
prev

Advertisement

Managing Transactions with EJB

- April 17, 2017
- By Manoj Debnath
- Send Email »
- More Articles »

Tweet

The infrastructural support for managing transactions is one of the major services provided by the EJB container. It has been an emphatic effort on the part of the EJB framework to provide a convenient way to manage transactions and access control since its inception in the Java language. The built-in mechanism has the capability to roll each method with a declarative metadata to manipulate its behavior in the transactional process. It is equally possible to adhere to the automatic function of the underlying container. This article delves into the concept of transaction management with EJB, highlighting the key points in a terse manner.

Transactions in an Enterprise Arena

Transaction refers to a group activity either performed as a unit or not at all. Because the individual activities are so crucial and related that they must be accomplished as a whole; otherwise, it may lead to a "sorry" situation.

An unit of activity, such as persisting data objects, validating security information of credit cards, sending email, and so forth, is actually a collection of several individual tasks. Moreover, the enterprise arena is a multitasking environment where two or more transactions of the same interweaving tasks may result in an inconsistent state of operation. For example, one transaction has updated data in a file and is about to commit; in the meantime, another transaction intervenes and picks up the old/wrong data from the file. This type of problem is very common in transactions and is generally solved by implementing a different locking mechanism.

The Challenges of Cloud Integration Download

Preparing for the Internet of Things: What You Need to Know Download

The guidelines to maintaining the sanity of transaction processing is given by the ACID (atomicity, consistency, isolation, durability) properties. Enterprise application generally works in a multi-tier framework. Any reliable database is capable of maintaining these (ACID) properties at the low-level, no doubt. But, what about similar problem arising in the middle tier, or even in the client tier as well? The possible solution is to handle them with special application logic. EJB here provides the impetus, with its services to build the application logic. Another situation may be that a database may use built-in concurrency control through pessimistic locking to manage the transaction. But, what if the application developer at another tier chooses to use a different locking strategy to optimize the performance? This is where EJB play its role. The transaction management APIs or the JTA focuses in providing enterprise-wide services for transaction management, giving a finer control in the hands of the programmer.

Ways to Manage EJB Transactions

- Post a comment
- Email Article
- Print Article

EJB transaction is built on the JTA model. The transactional context is typically provided by the session bean and application clients. Transactional services are performed within this context. The services are like creating, updating, retrieving, and deleting entities; invoking Web Services; executing JDBC operations; and so on. A transaction manager is provided by the EJB container, yet the real power lies in the declarative services in the form of metadata provided by the EJB. This declarative metadata provides the opportunity to participate actively in the transaction process. Programmers can manipulate the transaction process by using these metadata tags instead of reinventing complicated program logic. The metadata tags control the transaction behavior of the business methods in the enterprise framework.

EJB provides extensive support to both JTA and non-JTA transactions. Non-JTA (resource-local) transactions are generally chosen on the basis of transaction type restricted to a single resource manager, something that we typically see while creating a database connection. This particularly optimizes performance because the overhead of maintaining a distributed transaction monitor is clearly absent in this case.

Code Snippet of a JTA-based Transaction Configuration with JPA from persistence.xml

```
<persistence-unit name="LibraryPU" transaction-type="JTA">
   <jta-data-source>java:app/library_jndi</jta-data-source>
   <exclude-unlisted-classes>false</exclude-unlisted-classes>
   <properties>
      <property
         name="javax.persistence.schema-generation.database.action"
         value="create"/>
   </properties>
</persistence-unit>
```

Code Snippet of a Non–JTA (local resource)-based Transaction Configuration with JPA from persistence.xml

```
<persistence-unit name="LibraryPU-JSE" transaction-type="RESOURCE_LOCAL">
   <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
   <exclude-unlisted-classes>false</exclude-unlisted-classes>
   <properties>
      <property
         name="javax.persistence.schema-generation.database.action"
         value="create"/>
      <property name="javax.persistence.jdbc.driver"
         value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.url"
         value="jdbc:mysql://localhost:3306/testdb?zeroDateTimeBehavior=
                convertToNull"/>
      <property name="javax.persistence.jdbc.user"
                value="root"/>
      <property name="javax.persistence.jdbc.password"
         value="secret"/>
      <property name="eclipselink.logging.level"
         value="FINER"/>
   </properties>
</persistence-unit>
```

The developer may choose to participate more passively in the transaction process by leveraging a *container-managed* transaction. The *container-managed* transaction takes control of the transaction process, by default, relieving the developer from writing complex code.

By active participation in the transaction process, we mean the *bean-managed* transaction, where one takes control of the transactional boundaries by explicitly handling the activity such as begin transaction, commit, and rollback events.

However, it also is possible to leverage both models within a single application. In such a case, the EJB definition takes the decisive stance of transaction demarcation whether to use a *container-managed* or *bean-managed* transaction model. For example, the decision of the transaction model to use with JPA entities is determined by the configuration supplied in the *persistence.xml* file.

Container-managed Transactions

A *container-managed* transaction provides a built-in mechanism to deal with transaction services by default. These services are availed by the session beans and message-driven beans. Transaction directives are given either using annotations or an XML configuration to designate a transaction-aware method. The EJB container performs transactional actions according to the directives provided. These directives define a transactional demarcation, such as beginning a transaction, suspending, reusing an existing transaction, or committing when the method is invoked.

When a call to a transaction method is made, the EJB container intervenes and applies the rule in the session bean's method boundaries. By default, the container checks if the current transaction context is associated with any thread currently running. If found, the transaction control is given to the method invoked; otherwise, a new transaction is begun prior to executing the method.

However, the default behavior of transaction demarcation by the *container-managed* transaction can be overridden by decorating the method with a *@TransactionAttribute* annotation. The attributes *value* is defined in the *javax.ejb.TransactionAttributeType* enum.

Enum Constants	**Description (*)**
MANDATORY	If a client invokes the enterprise bean's method while the client is associated with a transaction context, the container invokes the enterprise bean's method in the client's transaction context.
NEVER	The client is required to call without a transaction context; otherwise, an exception is thrown.
NOT_SUPPORTED	The container invokes an enterprise bean method whose transaction attributes are NOT_SUPPORTED with an unspecified transaction context.
REQUIRED	If a client invokes the enterprise bean's method while the client is associated with a transaction context, the container invokes the enterprise bean's method in the client's transaction context.
REQUIRES_NEW	The container must invoke an enterprise bean method whose transaction attribute is set to REQUIRES_NEW with a new transaction context.
SUPPORTS	If the client calls with a transaction context, the container performs the same steps as described in the REQUIRED case.

(*) Excerpt from Java EE 7 API docs (https://docs.oracle.com/javaee/7/api/)

Here is an example of how to specify transaction behavior through annotation on a session bean method. This overrides the default behavior specified by the *container-managed* transaction model.

```
@TransactionAttribute(value =
   TransactionAttributeType.SUPPORTS)
public BookOrder createBookOrder()
```

```
      throws Exception {

   // ...transaction critical code

}
```

Bean-managed Transactions

As we can see, the container-managed transaction model automatically handles the intricacies of transaction management. Developers need not much bother about them in most cases. But, it has its own limitations. The demarcation granularity achieved by this model is insufficient in some cases. The major flaw is that a method can be wrapped in a single transaction only. Suppose a client wishes to call multiple methods on a session bean where each method finishes execution without committing. The client has two options to control the demarcation of transaction. Firstly, by instantiating its own transaction through customized container-managed transaction or, secondly, by using transaction resources available through the EJB context. A *bean-managed* transaction exactly does the second.

A *bean-managed* transaction provides the capability to handle demarcation of transaction events explicitly. We can turn off the default behavior of *container-managed* transaction services by the annotation *@TransactionManagement (TransactionManagementType.BEAN)*, or we can specify equivalent metadata in the file called *ejb-jar.xml*. However, using the *bean-managed* transaction model does not mean that the container will stop its support. Rather, support is provided through the *UserTransaction* object available through the bean's *EJBContext* object (such as *SessionContext*).

Therefore, grossly, the basic difference between the two models is that in the *container*-managed transaction model, transaction demarcation is established implicitly, whereas the *bean-managed* transaction model is explicit.

Here is a code example (snippet) to give an idea how it may look in code.

```
@Stateless
@TransactionManagement(value=TransactionManagementType.BEAN)
public class BookOrderDaoBean {

   @Resource
   SessionContext sessionContext;

   @PersistenceContext
   private EntityManager entityManager;

   private final UserTransaction userTransaction=
      sessionContext.getUserTransaction();

   public void saveMultipleOrder(List<BookOrder> list){

      // ...
      for(BookOrder b: list){
         try{
            userTransaction.begin();
            entityManager.persist(b);
            userTransaction.commit();
         }catch(Exception e){
            try {
               userTransaction.rollback();
            } catch (Exception ex) {
               Logger.getLogger(BookOrderDaoBean.class
               .getName())
               .log(Level.SEVERE, null, ex);
            }
         }
      }
   }

   // ...

}
```

Conclusion

A *container-managed* transaction model enforces implicit commit behavior by establishing the initiation of a transaction with the method invocation and concludes at the end of its execution. The developer is freed from the burden of handling the intricacies of it. However, there is a chance that the container may lose track of transaction control as it pops through the process stack. *Bean-managed* transaction is more flexible, with the explicit commit model. The developer controls the transaction demarcation stages. As a result, one must ensure that there is no dangling transaction. This is the price of flexibility.

IT Solutions Builder TOP IT RESOURCES TO MOVE YOUR BUSINESS FORWARD

Which topic are you interested in?

M urity Networks/IoT Cloud Data Storage Applications Development IT Management Other

Modernizing Enterprise IT: Motivators and Challenges

hite Papers and Webcasts

Sponsored content

Modernizing Enterprise IT: Motivators and Challenges

By Okta

April 10, 2019

Many organizations are embracing an approach to modernizing IT. And the benefits of replacing legacy infrastructure are clear, including reduced costs and increased business agility. But modernizing IT can mean different things to different IT leaders, and the reasons behind it can vary from business to business. ***Continue reading...***

Many organizations are embracing an approach to modernizing IT. And the benefits of replacing legacy infrastructure are clear, including reduced costs and increased business agility. ***Continue reading...***

Sponsored by Okta

0 Comments **(click to add your comment)**

Comment and Contribute

Your name/nickname

Your email

Subject

(Maximum characters: 1200). You have 1200 characters left.

I'm not a robot

reCAPTCHA
Privacy - Terms

Submit Your Comment

Enterprise Development Update

Don't miss an article. Subscribe to our newsletter below.

Enter Email Address

SIGN UP

Most Popular Developer Stories

- Today
- This Week
- All-Time

- 1 Using JDBC with MySQL, Getting Started
- 2 An Introduction to Java Annotations
- 3 MIDP Programming with J2ME
- 4 An Introduction to JSP Standard Template Library (JSTL)
- 5 Debugging a Java Program with Eclipse

- 1 Using JDBC with MySQL, Getting Started
- 2 An Introduction to Java Annotations
- 3 An Introduction to JSP Standard Template Library (JSTL)
- 4 MIDP Programming with J2ME

- 5 Debugging a Java Program with Eclipse

- 1 Using JDBC with MySQL, Getting Started
- 2 An Introduction to Java Annotations
- 3 An Introduction to JSP Standard Template Library (JSTL)
- 4 MIDP Programming with J2ME
- 5 Debugging a Java Program with Eclipse

Most Commented On

- This Week
- This Month
- All-Time

- 1 10 Experimental PHP Projects Pushing the Envelope
- 2 Day 1: Learning the Basics of PL/SQL
- 3 C# Tip: Placing Your C# Application in the System Tray
- 4 Logical Versus Physical Database Modeling
- 5 Is Ubuntu Contributing as Much as It Should to Free Software Projects?

- 1 Day 1: Learning the Basics of PL/SQL
- 2 The 5 Developer Certifications You'll Wish You Had in 2015
- 3 10 Experimental PHP Projects Pushing the Envelope
- 4 An Introduction to Struts
- 5 Inside Facebook's Open Source Infrastructure

- 1 Creating Use Case Diagrams
- 2 Day 1: Learning the Basics of PL/SQL
- 3 C# Tip: Placing Your C# Application in the System Tray
- 4 Using ASP.NET To Send Email
- 5 Using JDBC with MySQL, Getting Started

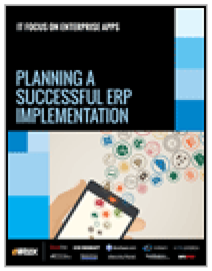
Top White Papers and Webcasts

It's Time to Rethink CRM

Today's CRM systems have the ability to deliver more than just lead generation. If leveraged correctly, they can be major drivers in loyalty and relationship management. But many companies have yet to unlock their true potential...

Ready Your Enterprise for the API Revolution

APIs are changing more than just software architectures. From planning through implementation and beyond, an API-driven business model brings a host of...



Planning a Successful ERP Implementation

Changes in the ERP landscape have made the deployment process increasingly complex. Post-deployment, implementations often fail to deliver on their...

A Developer.com Property

Terms of Service | Licensing & Reprints | About Us | Privacy Policy | Contact Us | Advertise | Sitemap

Copyright 2019 QuinStreet Inc. All Rights Reserved.

Advertiser Disclosure: Some of the products that appear on this site are from companies from which QuinStreet receives compensation. This compensation may impact how and where products appear on this site including, for example, the order in which they appear. QuinStreet does not include all companies or all types of products available in the marketplace.

Thanks for your registration, follow us on our social networks to keep up-to-date