
JavaOne™

JavaOne™

What's New in Java Transaction Processing

Paul Parkinson, Oracle
Monica Riccelli, Oracle

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

- JTA 1.2
- JTA 1.3
- Beyond
- Discussion

Java Transaction API 1.2

Overview

- First release of JTA in over 10 years
- Complete backward compatibility (essential)
- Two major new features **`@Transactional and @TransactionScoped`** annotations
- Clarification on delistment of resources before prepare.

Java Transaction API 1.2

@Transactional

- Annotation provides the application the ability to control transaction boundaries on CDI managed beans
- Defined in terms of a system interceptor
- Option to specify exception handling behavior
  - Declare which exceptions should cause the transaction to be marked for rollback

Java Transaction API 1.2

Some @Transactional Details

- Can be placed on class or method level
  - Overriding rules apply
- Business methods, not lifecycle methods
- Can invoke **`TransactionSynchronizationRegistry`**, but not **`UserTransaction`**
- Disallowed on EJB components
- Allowable transaction types match existing CMT tx types

Java Transaction API 1.2

@Transactional TxType

- **`TxType.REQUIRED`** – start new tx if none already in progress
- **`TxType.REQUIRES_NEW`** – start new tx, suspend if already in progress
- **`TxType.MANDATORY`** – tx must already be in progress
- **`TxType.SUPPORTS`** – use existing tx if in progress, or none if it isn't
- **`TxType.NOT_SUPPORTED`** – suspend tx if in progress
- **`TxType.NEVER`** – tx must not be in progress

Java Transaction API 1.2

Exception Handling

- Default behavior
  - Runtime exceptions - transaction will be marked for rollback
  - Checked exceptions - transaction will not be marked for rollback
- Can override using **`rollbackOn`** and **`dontRollbackOn`** elements
- Overriding behavior will be applied to the specified exception class and its subclasses
- Rollback behavior applies to the current transaction (regardless of whether it was actually started by the @Transactional method)

Java Transaction API 1.2

@Transactional Example

```
@Transactional(value=Transactional.TxType.REQUIRED,
               rollbackOn={SQLException.class},
               dontRollbackOn={SQLWarning.class})
public class MyBean {
    //…
    @Transactional(value=Transactional.TxType.SUPPORTS)
    private void myMethod(String param) { … }
    //…
}
```

Java Transaction API 1.2

@TransactionScoped

- Declaratively scope a CDI bean to the current transaction
- Independent of the way the transaction was started
  - E.g. **`@Transactional`**, **`UserTransaction`**, or EJB CMT
- Context is maintained across transaction suspend, resume, etc.
- Context lasts until transaction is committed/rolledback.
- Invoking a contextual instance after its enclosing transaction completes results in **`ContextNotActiveException`** being thrown

Java Transaction API 1.2

@TransactionScoped Example Bean

```
@TransactionScoped
private class ValueBean {

    private static int Value = 0;

    private int myValue;

    public ValueBean() { myValue = ++Value; }

    public int getValue() { return myValue; }
}
```

Java Transaction API 1.2

@TransactionScoped Example Injection

```
private class Invoker {

    @Transactional
    public String method1(@Inject ValueBean bean) {
        System.out.println("Value=" + bean.getValue());
        method2(bean);
    }
    @Transactional(TxType.REQUIRES_NEW)
    public String method2(ValueBean bean) {
        System.out.println("Value=" + bean.getValue());
    }
}
```

Java Transaction API 1.2

Minor Clarifications

- A container only needs to call delistResource to explicitly dissociate a resource from a transaction and it is **not a mandatory** container requirement to do so as a precondition to transaction completion.

- A transaction manager **is required** to implicitly delist the resource before tx completion; that is before prepare (or commit/rollback in the onephase-optimized case).

JTA 1.2 Summary

- More flexible container-managed transactional demarcation
  - Any CDI managed bean
  - All CMT tx options
  - Control over rollback behavior
- Transactional scope for CDI managed beans
  - Simpler model for programming using transactions
- Full backward compatibility to all existing JTA providers
- Good foundation for any necessary features going forward

Java Transaction API 1.3

Committed

- JTA_SPEC-4/ordered commit
- JTA_SPEC-9/setRollbackOnly(Exception) API addition

Java Transaction API 1.3

JTA_SPEC-4/support explicit ordering of commits for XAResources enlisted in a transaction

- Ordering issues exist in the case where a message and database row update exist in the same transaction. If the message is committed and processed by another transaction before the database update is committed, the uncommitted row can then be processed prematurely.
- Provide a standard for the ordering of the XAResource commits.
- In addition correct ordering of commits can allow readonly one-phase commit.

Java Transaction API 1.3

JTA_SPEC-4/support explicit ordering of commits for XAResources enlisted in a transaction - continued

- Should it be only first and last or weighted.
- Should this be part of resource-ref configuration?
- Should there also be an interface defined for this?

Java Transaction API 1.3

JTA_SPEC-9/new api for setRollbackOnly so that it can take a nested exception

- Add a new setRollbackOnly(Exception exception) API method to Transaction, TransactionManager, and UserTransaction where the resultant RollbackException later thrown has the exception parameter as a nested exception that can be inspected by the application.

Java Transaction API 1.3 or Beyond

Ideas

- JTA_SPEC-2/standard way to represent XAResources to restore XAResources
- JTA_SPEC-8/specify XAResource(s) transaction timeout value be set to the value of the encompassing JTA transaction's timeout value by default for portability
- No JIRA currently/add a new begin(properties) API to provide information such as expected resource count or read/write characteristics. For example the container could enlist a local-tx mode db connection rather than an XA one if it is know that only one resource will be used in the transaction.

Java Transaction API 1.3 or Beyond

Smaller items and clarifications

- JTA_SPEC-11/elaborate on ordering semantics for Synchronization calls in a distributed transaction
- JTA_SPEC-6/Clarify transaction interactions/restrictions in TX Synchronization callbacks

Resources

- JSR 907 page:
  http://jcp.org/en/jsr/detail?id=907
- JTA java.net project page:
  http://java.net/projects/jta-spec/
- JTA Jira:
  http://java.net/jira/browse/JTA_SPEC
- Distribution list:
  users@jta-spec.java.net

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

MAKE THE FUTURE JAVA

JavaOne™

ORACLE®


JavaOne™