Home

Transaction Annotations

Also see Testing Transactions *for an example of how to use and test EJB transaction attributes*

The *javax.ejb.TransactionAttribute* annotation (@TransactionAttribute) can be applied to a bean class or it's methods.

Usage of the @TransactionAttribute requires you to specify one of six different transaction attribute types defined via the javax.ejb.TransactionAttributeType enum.

- TransactionAttributeType.*MANDATORY*
- TransactionAttributeType.*REQUIRED*
- TransactionAttributeType.*REQUIRES_NEW*
- TransactionAttributeType.*SUPPORTS*
- TransactionAttributeType.*NOT_SUPPORTED*
- TransactionAttributeType.*NEVER*

Per EJB 3.0 the default transaction attribute for all EJB 3.0 applications is *REQUIRED*. The default transaction attribute for EJB 2.1, 2.0 and 1.1 applications is vendor specific. In OpenEJB EJB 2.1, 2.0 and 1.1 applications also use *REQUIRED* as the default.

Attribute Types summary

A simplistic way to visualize the transaction attributes is as follows.

	Failing	Correcting	No Change
Transacted	MANDATORY	REQUIRED, REQUIRES_NEW	SUPPORTS
Not Transacted	NEVER	NOT_SUPPORTED	SUPPORTS

The "*Transacted*" and "*Not Transacted*" categories represent the container guarantee, i.e. if the bean method will or will not be invoked in a transaction. The "*Failing*", "*Correcting*", and "*No Change*" categories represent the action take by the container to achieve that guarantee.

For example, *Never* and *Mandatory* are categorized as "*Failing*" and will cause the container to throw an exception to the caller if there is (Tx Never) or is not (Tx Mandatory) a transaction in progress when the method is called. The attributes *Required*, *RequiresNew*, and *NotSupported* are categorized as "*Correcting*" as they will cause the container to adjust the transactional state automatically as needed to match the desired state, rather than blocking the invocation by throwing an exception.

Detailed description of each Attribute

MANDATORY

A *MANDATORY* method is guaranteed to always be executed in a transaction. However, it's the caller's job to take care of suppling the transaction. If the caller attempts to invoke the method *outside* of a transaction, then the container will block the call and throw them an *exception*.

REQUIRED

A *REQUIRED* method is guaranteed to always be executed in a transaction. If the caller attempts to invoke the method *outside* of a transaction, the container will *start* a transaction, execute the method, then *commit* the transaction.

REQUIRES_NEW

A *REQUIRES*NEW_ method is guaranteed to always be executed in a transaction. If the caller attempts to invoke the method *inside or outside* of a transaction, the container will still *start* a transaction, execute the method, then *commit* the transaction. Any transaction the caller may have in progress will be *suspended* before the method execution then *resumed* afterward.

NEVER

A *NEVER* method is guaranteed to never be executed in a transaction. However, it's the caller's job to ensure there is no transaction. If the caller attempts to invoke the method *inside* of a transaction, then the container will block the call and throw them an *exception*.

NOT_SUPPORTED

A *NOT*SUPPORTED_ method is guaranteed to never be executed in a transaction. If the caller attempts to invoke the method *inside* of a transaction, the container will *suspend* the caller's transaction, execute the method, then *resume* the caller's transaction.

SUPPORTS

A *SUPPORTS* method is guaranteed to adopt the exact transactional state of the caller. These methods can be invoked by caller's *inside or outside* of a transaction. The container will do nothing to change that state.

On Methods

```
 @Stateless
public static class MyBean implements MyBusinessInterface {
    @TransactionAttribute(TransactionAttributeType.MANDATORY)
    public String codeRed(String s) {
    return s;
    }

    public String codeBlue(String s) {
    return s;
    }
}
```

- *codeRed* will be invoked with the attribute of *MANDATORY*
- *codeBlue* will be invoked with the default attribute of *REQUIRED*

On Classes

```
 @Stateless
@TransactionAttribute(TransactionAttributeType.MANDATORY)
public static class MyBean implements MyBusinessInterface {

    public String codeRed(String s) {
    return s;
    }

    public String codeBlue(String s) {
    return s;
    }
}
```

- *codeRed* and *codeBlue* will be invoked with the attribute of *MANDATORY*

Mixed on classes and methods

```
 @Stateless
@TransactionAttribute(TransactionAttributeType.SUPPORTS)
public static class MyBean implements MyBusinessInterface {

    @TransactionAttribute(TransactionAttributeType.NEVER)
    public String codeRed(String s) {
    return s;
    }

    public String codeBlue(String s) {
    return s;
    }

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public String codeGreen(String s) {
    return s;
    }
}
```

- *codeRed* will be invoked with the attribute of *NEVER*
- *codeBlue* will be invoked with the attribute of *SUPPORTS*
- *codeGreen* will be invoked with the attribute of *REQUIRED*

Illegal Usage

Generally, transaction annotationss cannot be made on AroundInvoke methods and most callbacks.

The following usages of @TransactionAttribute have no effect.

```
 @Stateful
public class MyStatefulBean implements  MyBusinessInterface  {

    @PostConstruct
    @TransactionAttribute(TransactionAttributeType.NEVER)
    public void constructed(){

    }

    @PreDestroy
    @TransactionAttribute(TransactionAttributeType.NEVER)
    public void destroy(){

    }

    @AroundInvoke
    @TransactionAttribute(TransactionAttributeType.NEVER)
    public Object invoke(InvocationContext invocationContext) throws Exception {
    return invocationContext.proceed();
    }

    @PostActivate
    @TransactionAttribute(TransactionAttributeType.NEVER)
    public void activated(){

    }

    @PrePassivate
    @TransactionAttribute(TransactionAttributeType.NEVER)
    public void passivate(){

    }
}
```