

Spring中判断是否存在事务的原理

在TransactionSynchronizationManager 中存在isActualTransactionActive 可以判断当前是否存在事务，这个原理是什么？

```
public static boolean isActualTransactionActive() {  
    return (actualTransactionActive.get() != null);  
}  
  
private static final ThreadLocal<Boolean> actualTransactionActive =  
    new NamedThreadLocal<>("Actual transaction active");
```

那么actualTransactionActive 中的值是在哪里什么时候被设置的呢？

```
public static void setActualTransactionActive(boolean active) {  
    actualTransactionActive.set(active ? Boolean.TRUE : null);  
}
```

```
/**  
 * Initialize transaction synchronization as appropriate.  
 */  
protected void prepareSynchronization(DefaultTransactionStatus status, TransactionDefinition definition) {  
    if (status.isNewSynchronization()) {  
        TransactionSynchronizationManager.setActualTransactionActive(status.hasTransaction());  
        TransactionSynchronizationManager.setCurrentTransactionIsolationLevel(  
            definition.getIsolationLevel() != TransactionDefinition.ISOLATION_DEFAULT ?  
                definition.getIsolationLevel() : null);  
        TransactionSynchronizationManager.setCurrentTransactionReadOnly(definition.isReadOnly());  
        TransactionSynchronizationManager.setCurrentTransactionName(definition.getName());  
        TransactionSynchronizationManager.initSynchronization();  
    }  
}
```

从上面的这个图片中我们看到 判断当前是否存在事务的原理是依赖了DefaultTransactionStatus的hasTransaction方法

```
/**  
 * Return whether there is an actual transaction active.  
 */
```

```
public boolean hasTransaction() {  
    return (this.transaction != null);  
}
```

```
*/  
  
public class DefaultTransactionStatus extends AbstractTransactionStatus {
```

```
    @Nullable
```

```
    private TransactionObject transaction;
```

那么问题是：为什么可以依据TransactionStatus中的transaction来 判断是否存在事务呢？

因为我们之前分析过 在Spring的getTransaction方法中，第一步就是调用了doGetTransaction方法 来创建一个DataSourceTransactionObject作为事务对象

```
@Override
protected Object doGetTransaction() {
    DataSourceTransactionObject txObject = new DataSourceTransactionObject();
    txObject.setSavepointAllowed(isNestedTransactionAllowed());
    ConnectionHolder conHolder =
        (ConnectionHolder) TransactionSynchronizationManager.getResource(obtainDataSource());
    txObject.setConnectionHolder(conHolder, newConnectionHolder: false);
    return txObject;
}
```

DataSourceTransactionManager

然后这个事务对象DataSourceTransactionObject又会根据当前事务的级别：是否需要事务，是否需要创建新的事务，最终这个事务对象被交给了DefaultTransactionStatus对象中。但是我们注意到如果事务级别是不需要事务，这个时候我们创建了一个DefaultTransactionStatus对象，但是传递的transaction是null，表示他是一个空事务；当事务的级别是需要事务的时候我们会将doGetTransaction返回的事务对象交给DefaultTransactionStatus对象，因此在TransactionStatus中可以通过TransactionStatus的transaction属性是否为空来判断当前是否存在事务

```
@Override
public final TransactionStatus getTransaction(@Nullable TransactionDefinition definition) throws TransactionException {
    // Cache debug flag to avoid repeated checks.
    boolean debugEnabled = logger.isDebugEnabled();

    if (definition == null) {
        // Use defaults if no transaction definition given.
        definition = new DefaultTransactionDefinition();
    }

    if (isExistingTransaction(transaction)) {
        // Existing transaction found -> check propagation behavior to find out how to behave.
        return handleExistingTransaction(definition, transaction, debugEnabled);
    }

    // Check definition settings for new transaction.
    if (definition.getTimeout() < TransactionDefinition.TIMEOUT_DEFAULT) {
        throw new InvalidTimeoutException("Invalid transaction timeout", definition.getTimeout());
    }

    // No existing transaction found -> check propagation behavior to find out how to proceed.
    if (definition.getPropagationBehavior() == TransactionDefinition.PROPGATION_MANDATORY) {
        throw new IllegalTransactionStateException(
            "No existing transaction found for transaction marked with propagation 'mandatory'");
    }
    else if (definition.getPropagationBehavior() == TransactionDefinition.PROPGATION_REQUIRED ||
        definition.getPropagationBehavior() == TransactionDefinition.PROPGATION_REQUIRES_NEW ||
        definition.getPropagationBehavior() == TransactionDefinition.PROPGATION_NESTED) {
        SuspendedResourcesHolder suspendedResources = suspend(transaction);
        if (debugEnabled) {
            logger.debug("Creating new transaction with name [" + definition.getName() + "]: " + definition);
        }
        try {
            boolean newSynchronization = (getTransactionSynchronization() != SYNCHRONIZATION_NEW_Synchronization);
            DefaultTransactionStatus status = newTransactionStatus(
                definition, transaction, newTransaction: true, newSynchronization, debugEnabled, suspendedResources);
            doBegin(transaction, definition);
            prepareSynchronization(status, definition);
            return status;
        }
        catch (RuntimeException | Error ex) {
            resume(transaction: null, suspendedResources);
            throw ex;
        }
        // Create "empty" transaction: no actual transaction, but potentially synchronized.
        if (definition.getIsolationLevel() != TransactionDefinition.ISOLATION_DEFAULT) {
            logger.warn("Custom isolation level specified but no actual transaction initiated; " +
                "isolation level will effectively be ignored: " + definition);
        }
        // Create "empty" transaction: no actual transaction, but potentially synchronized.
        boolean newSynchronization = (getTransactionSynchronization() != SYNCHRONIZATION_NEW_Synchronization);
        return prepareTransactionStatus(definition, transaction: null, newTransaction: true, newSynchronization, debugEnabled, suspendedResources: null);
    }
}
```