**Spring Transaction Management**

**Spring Transaction Management** is one of the most widely used and important feature of [spring framework](https://www.journaldev.com/16922/spring-framework). Transaction Management is a trivial task in any enterprise application. Sequence of action that will be performed to complete database operation and its management is known as Transaction Management. All these action in combination will be treated as ONE action only. So that DB doesn’t fall in inconsistent mode ever. Comprehensive transaction support is among the most compelling reasons to use the Spring Framework. The Spring Framework provides a consistent abstraction for transaction management. The concept of transactions can be described with the following four key properties described as **ACID** −

* **Atomicity** − A transaction should be treated as a single unit of operation, which means either the entire sequence of operations is successful or unsuccessful.
* **Consistency** −This represents the consistency of the referential integrity of the database, unique primary keys in tables, etc.
* **Isolation** − There may be many transaction processing with the same data set at the same time. Each transaction should be isolated from others to prevent data corruption.
* **Durability** − Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

**Type of Transaction Management**

In J2EE, Transaction Management can be divided in two types.

1. Global Transaction
2. Local Transaction

Global Transaction

* Use to work with multiple transaction resources like RDBMS or Message Queue (Pros)
* Managed by Application Server (WebSphere, WebLogic) using JTA (Cons)
* JNDI is required to use JTA
* Code cannot be reused as JTA is available at server level(Cons)
* Example of Global Transaction : EJB CMT

### Local Transaction

* Use to work with specific resource(transaction associated with JDBC)
* Cannot work across multiple transaction resource opposite to Global transaction (cons)
* Most of web application uses only single resources hence it is best option to use in normal app.

## Different Approach for transaction management

Spring supports two different approach for transaction management.

Programmatic Transaction Management

Here you will write code for transaction management. Spring API dependency. Not good for maintenance. Good for development. Flexibity.

### Declarative Transaction Management

Here you will use XML or annotation for transaction management. Less flexible but preferable over programmatic approach. In normal case no code is required for transaction management. Declarative transaction management is preferable over programmatic transaction management though it is less flexible than programmatic transaction management, which allows you to control transactions through your code. But as a kind of crosscutting concern, declarative transaction management can be modularized with the AOP approach. Spring supports declarative transaction management through the Spring AOP framework.

## Spring transaction management abstraction

To understand transaction management you should understand abstraction (Transaction strategy) in spring. Which is defined in spring using PlatformTransactionManager Interface.

|  |
| --- |
| public interface PlatformTransactionManager {  TransactionStatus getTransaction(TransactionDefinition definition) throws TransactionException;  void commit(TransactionStatus status) throws TransactionException;  void rollback(TransactionStatus status) throws TransactionException;  } |

As you can see in **PlatformTransactionManager** Interface all methods throw **TransactionException.**This Exception itself is **UncheckedException** means developer is not forced to handle these exceptions.

|  |  |
| --- | --- |
| **Sr.No** | **Method & Description** |
| 1 | **TransactionStatus getTransaction(TransactionDefinition definition)**  This method returns a currently active transaction or creates a new one, according to the specified propagation behavior. |
| 2 | **void commit(TransactionStatus status)**  This method commits the given transaction, with regard to its status. |
| 3 | **void rollback(TransactionStatus status)**  This method performs a rollback of the given transaction. |

The TransactionDefinition is the core interface of the transaction support in Spring and it is defined as follows –

public interface TransactionDefinition {

int getPropagationBehavior();

int getIsolationLevel();

String getName();

int getTimeout();

boolean isReadOnly();

}

## Transaction Definition

TransactionDefinition is an Interface which specifies below 4 points.

1. Isolation
2. Propagation
3. Timeout
4. Read-Only status

### Isolation

Degree to which particular transaction is isolated from other Transaction. Below is the list of Isolation level and their details

#### DEFAULT

Default isolation level. It uses the isolation of underlying data source.

#### READ\_COMMITTED

Dirty reads NOT supported; Non-repeatable reads and Phantom reads can occur.

#### READ\_UNCOMMITTED

Dirty reads / Non-repeatable reads / Phantom reads all can occur.

#### REPEATABLE\_READ

Dirty reads and non-repeatable reads are prevented; phantom reads can occur.

#### SERIALIZABLE

Dirty reads, non-repeatable reads and phantom reads are prevented.

### Propagation

#### MANDATORY

Support a current transaction

#### NESTED

Execute within a nested transaction if a current transaction exists

#### NEVER

Execute non-transactionally

#### NOT\_SUPPORTED

Execute non-transactionally

#### REQUIRED

Support a current transaction

#### REQUIRES\_NEW

Create a new transaction, suspend the current transaction if one exists.

#### SUPPORTS

Support a current transaction, execute non-transaction if none exists.

### Timeout

This setting is used to define how long this transaction may run before timing out (candidate for rolled back by the underlying transaction infrastructure).

#### Read-Only Status

This setting is used to specify the read-only transaction. As read only transactions does not modify any data. Hence it is optimized in some case.