Transactional Tests


ACID
SUPPORTS
NESTED
REQUIRED
FANTOM_READS
MANDATORY
REQUIRES_NEW
NON_REPEATABLE_READ
READ_UNCOMMITED
NOT_SUPPORTED
REPEATABLE_READ
NEVER
@Transactional
DIRTY_READS
SERIALIZABLE
Transactions
READ_COMMITED
PlatformTransactionManager
@EnableTransactionManagement

Transactions

- Atomicity is the main attribute of a transaction and is the characteristic mentioned earlier, that if an operation in a transaction fails, the entire transaction fails, and the database is left unchanged. When all operations in a transaction succeed, all changes are saved into the database when the transaction is committed. Basically it is “all or nothing.”

- Consistency implies that every transaction should bring the database from one valid state to another.

- Isolation implies that when multiple transactions are executed in parallel, they won’t hinder one another or affect each other in any way. The state of the database after a group of transactions is executed in parallel should be the same as if the transactions in the group had been executed sequentially.

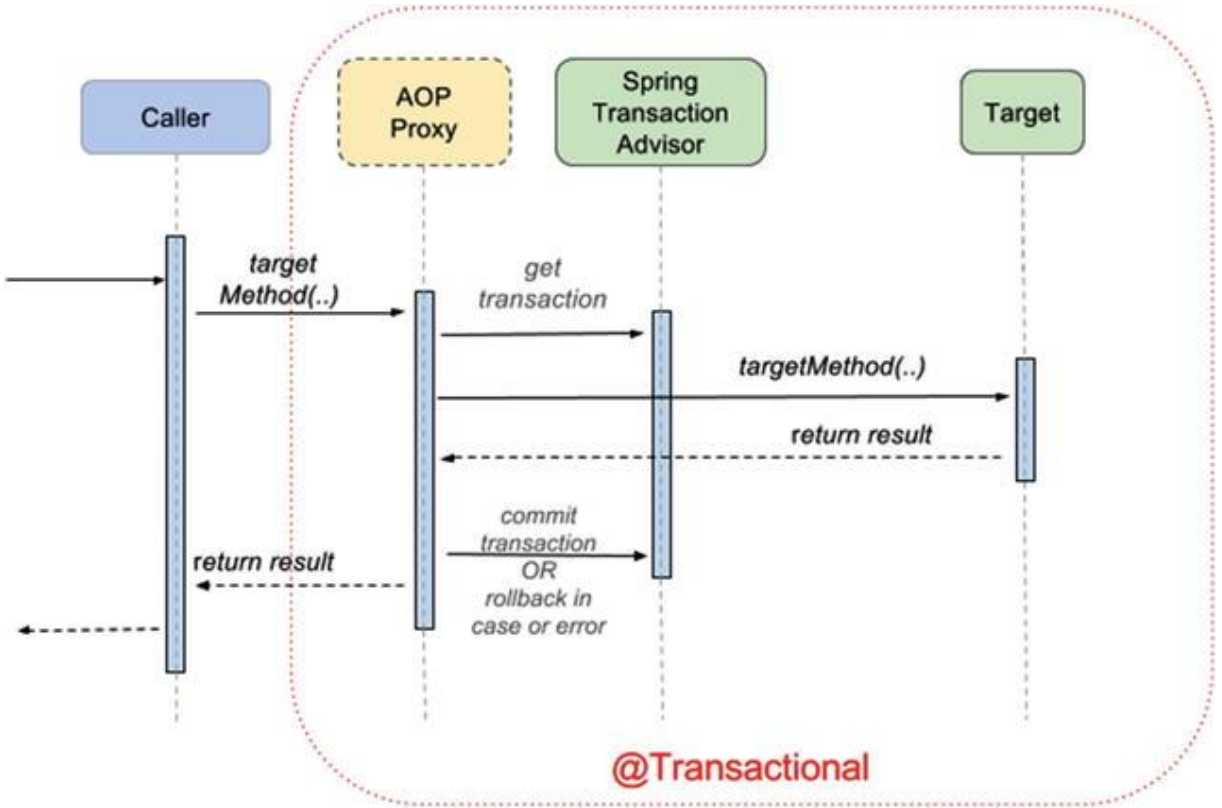
- Durability is the property of a transaction that should persist even in cases of power off, crashes, and other errors on the underlying system.

Enabling Transactions

- @EnableTransactionManagement - enable all infrastructure beans necessary for supporting transactional execution

- PlatformTransactionManager - Defines Spring’s transaction strategy

@Transactional



Isolation Levels

- READ UNCOMMITTED
- READ COMMITTED (protecting against dirty reads)
- REPEATABLE READ (protecting against dirty and non-repeatable reads)
- SERIALIZABLE (protecting against dirty, non-repeatable reads and phantom reads)

Propagation Behaviour

- REQUIRED: an existing transaction will be used or a new one will be created to execute the method annotated with @Transactional(propagation = Propagation.REQUIRED).
- REQUIRES_NEW: a new transaction is created to execute the method annotated with @Transactional(propagation = Propagation.REQUIRES_NEW). If a current transaction exists, it will be suspended.
- NESTED: an existing nested transaction will be used to execute the method annotated with @Transactional(propagation = Propagation.NESTED). If no such transaction exists, it will be created.
- MANDATORY: an existing transaction must be used to execute the method annotated with @Transactional(propagation = MANDATORY). If there is no transaction to be used, an exception will be thrown.
- NEVER: methods annotated with @Transactional(propagation = Propagation.NEVER must not be executed within a transaction. If a transaction exists, an exception will be thrown.
- NOT_SUPPORTED: no transaction is used to execute the method annotated with @Transactional(propagation = Propagation.NOT_SUPPORTED). If a transaction exists, it will be suspended.
- SUPPORTS: an existing transaction will be used to execute the method annotated with @Transactional(propagation = Propagation.SUPPORTS). If no transaction exists, the method will be executed anyway, without a transactional context.


ACID
SUPPORTS
NESTED
REQUIRED
FANTOM_READS
MANDATORY
REQUIRES_NEW
NON_REPEATABLE_READ
READ_UNCOMMITED
NOT_SUPPORTED
REPEATABLE_READ
NEVER
@Transactional
DIRTY_READS
SERIALIZABLE
Transactions
READ_COMMITED
PlatformTransactionManager
@EnableTransactionManagement