Spring.transactions

InformAtelier

Strategy

The Spring Framework gives you the choice of when to scale your application to a fully loaded application server.

InformAtelier

PlatformTransactionManager

```
public interface PlatformTransactionManager {

    TransactionStatus getTransaction(TransactionDefinition definition)
                                                        throws TransactionException;

    void commit(TransactionStatus status) throws TransactionException;

    void rollback(TransactionStatus status) throws TransactionException;
}
```

Propagation

- PROPAGATION_REQUIRES_NEW
- PROPAGATION_REQUIRED
- PROPAGATION_NESTED

- a participating transaction joins the characteristics of the outer scope

Isolation

- DEFAULT: use the default isolation level of the underlying datastore.
- READ_COMMITTED: dirty reads are prevented; non-repeatable reads and phantom reads can occur.
- READ_UNCOMMITTED: dirty reads, non-repeatable reads and phantom reads 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.

- Only applicable to propagation settings of REQUIRED or REQUIRES_NEW.

TimeOut

- Only applicable to propagation settings of REQUIRED or REQUIRES_NEW.

Transaction Defaults

- The propagation setting is PROPAGATION_REQUIRED.
- The isolation level is ISOLATION_DEFAULT.
- The transaction is read-write.
- The transaction timeout defaults to the default timeout of the underlying transaction system, or to none if timeouts are not supported.
- Any RuntimeException triggers rollback, and any checked Exception does not.

@Transactional

- @Transactional, meaning that any failure causes the entire operation to roll back to its previous state, and to re-throw the original exception.
- Only to methods with public visibility, else ignored.
- Or use AspectJ

Usage

```
@Component
public class BookingService {
private final JdbcTemplate jdbcTemplate;

public BookingService(JdbcTemplate jdbcTemplate) {
      this.jdbcTemplate = jdbcTemplate;
  }

@Transactional
  public void book(String... persons) {
      for (String person : persons) {
         jdbcTemplate.update("insert into BOOKINGS(FIRST_NAME) values (?)", person);
      }
  }
```

Riferimenti

- https://docs.spring.io/spring/docs/current/spring-framework-reference/data-access.html#transaction
- https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#_transactions_with_code_mongotransactionmanager_code
- https://spring.io/guides/gs/managing-transactions/