Transaction in Spring

ZeroPage 30기 정회원 김은솔

Do you know Transaction?

- DBMS에서 수행되는 논리적인 작업의 단위
- 하나 이상의 데이터베이스 조작 (INSERT, UPDATE, DELETE 등)을 묶어 하나의 작업으로 처리함

ACID Properties

- 원자성 Atomicity: 트랜잭션의 모든 연산은 하나의 원자적 작업 단위로 간주되어, 트랜잭션의 연산은 전부 수행되거나 전혀 수행되지 않아야 함
- 일관성 Consistency: 트랜잭션의 수행 여부와 관계 없이 데이터베이스 상태는 일관된 상태를 유지해야 함
- 격리성 Isolation: 각 트랜잭션은 시스템 내에서 독립적으로 실행되는 것처럼 보여야 함
- 지속성 Durability: 트랜잭션이 커밋된 후에 해당 트랜잭션이 갱신한 데이터베이스의 내용은 영구적으로 저장됨

@Transactional

- class level, method level 에 붙일 수 있는 어노테이션
- 인터페이스에도 붙일 수는 있으나, 지양하는 것이 좋음

어떻게 어노테이션으로 트랜잭션을 구현할까?

```
Connection connection = dataSource.getConnection(); // (1)

try (connection) {
    connection.setAutoCommit(false); // (2)
    // execute some SQL statements...
    connection.commit(); // (3)

} catch (SQLException e) {
    connection.rollback(); // (4)
}
```

어떻게 어노테이션으로 트랜잭션을 구현할까?

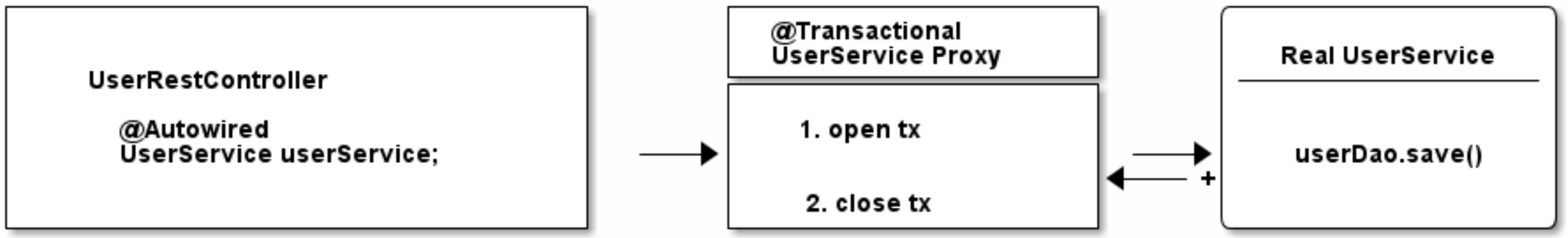
```
public class UserService {

    @Transactional
    public Long registerUser(User user) {
        // execute some SQL that e.g.
        // inserts the user into the db and retrieves the
autogenerated id
        // userDao.save(user);
        return id;
    }
}
```

어떻게 어노테이션으로 트랜잭션을 구현할까?

```
public class UserService {

    public Long registerUser(User user) {
        Connection connection = dataSource.getConnection(); // (1)
        try (connection) {
            connection.setAutoCommit(false); // (1)

            // execute some SQL that e.g.
            // inserts the user into the db and retrieves the autogenerated
id
            // userDao.save(user); <(2)

            connection.commit(); // (1)
        } catch (SQLException e) {
            connection.rollback(); // (1)
        }
    }
}
```

어떻게 어노테이션으로 트랜잭션을 구현할까?

- Spring 이 @Transactional 어노테이션을 감지하면 동적 프록시 생성
- 프록시에서 트랜잭션/연결을 열고 닫도록 요청



Proxy mode

- @Transactional 이 프록시를 통해 들어오는 ‘외부 메서드 호출’만 가로챔
- 객체 내부의 메서드가 동일 객체의 다른 메서드를 호출하는 것은 실제 트랜잭션으로 실행되지 않음
- @PostConstruct 메서드에서 @Transactional 에 의존해서는 안 됨

물리 / 논리 트랜잭션

- 물리 트랜잭션: 실제 JDBC 트랜잭션
- 논리 트랜잭션: 잠재적으로 중첩된 @Transactional 메서드

물리 /

- 물리 트
- 논리 트

```java
@Service
public class UserService {

    @Autowired
    private InvoiceService invoiceService;

    @Transactional
    public void invoice() {
        invoiceService.createPdf();
        // send invoice as email, etc.
    }
}

@Service
public class InvoiceService {

    @Transactional
    public void createPdf() {
        // ...
    }
}
```

트랜잭션 안에 트랜잭션이 있다면?

```java
@Service
public class UserService {

    @Autowired
    private InvoiceService invoiceService;

    @Transactional
    public void invoice() {
        invoiceService.createPdf();
        // send invoice as email, etc.
    }
}

@Service
public class InvoiceService {

    @Transactional
    public void createPdf() {
        // ...
    }
}
```

- 데이터 트랜잭션 관점에서 보면 하나의 Connection

트랜잭션 안에 트랜잭션이 있다면?

- 내부 트랜잭션 속성이 변경된다면?
- 외부 물리 트랜잭션과 별개로 새로운 데이터베이스 Connection 을 사용하는 것으로 간주됨

```java
@Service
public class InvoiceService {

    @Transactional(propagation = Propagation.REQUIRES_NEW)
    public void createPdf() {
        // ...
    }
}
```

Propagation Properties

- REQUIRED
- SUPPORTS
- MANDATORY
- REQUIRES_NEW
- NOT_SUPPORTED
- NEVER
- NESTED

트랜잭션의 격리 레벨

- 동시에 실행중인 여러 트랜잭션 간의 데이터 일관성과 격리 수준을 조절하는데 사용
- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE