

Spring 5화

Transaction 1

Spring(3화) 목차

- transaction?
- Spring의 transaction 관리
- 선언적 transaction의 설정 및 원리
- Aop를 이용한 설정 및 원리
- Aop의 원리
- Aop의 설정 및 원리
- Proxy Pattern의 적용
- Spring Boot의 트랜잭션 관리

transaction?

또갈수 없는업무 처리단위

- **trans + action**으로 구성되어 있으며, 거래하는 행위라고 함.
- 거래를 하기위해서는 주고 받는 행위가 만들어 지고, 주는 것과 받는 것이 연결되어 처리되어져야하기 때문에 거래 업무에서 많이 사용되어졌다고 함.
- 프로그램적으로는 꼭 같이 처리되어져야 로직 정도 일 것 같음.

transaction?

ACID - Atomicity, Consistency, Isolation, Durability

- **ACID**(원자성, 일관성, 고립성, 지속성)는 데이터베이스 트랜잭션이 안전하게 수행된다는 것을 보장하기 위한 성질을 가리키는 약어
- 짐 그레이는 **1970**년대 말에 정의했다고 함. 알아주면 있어보임.
- 원자성은 쪼갤수 없으니깐
- 일관성은 트랜잭션 시스템이 일관성 있게 유지되어야 한다는 것, 실패했을 때 롤백, 성공했을 때의 전체적인 데이터 처리
- 고립성은 다른 트랜잭션이 끼어들지 못하도록 고립하는 것
- 지속성은 성공된 트랜잭션의 반영은 지속적이어야 한다는 것

Spring의 transaction관리

TransactionManager를 통하여 관리된다.

- `org.springframework.transaction.PlatformTransactionManager`
- JTA(Java Transaction API), JDBC를 활용한 방법은 복잡하고 불편하나 Spring은 위의 일관된 인터페이스를 통해서 지속적이고 편안함을 제공한다고 함.(저 인터페이스만 보면 그렇기는 한데...)

Spring의 transaction관리

PlatformTransactionManager

- **public interface** PlatformTransactionManager {

 TransactionStatus getTransaction(
 TransactionDefinition definition) **throws** TransactionException;

 void commit(TransactionStatus status) **throws** TransactionException;

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

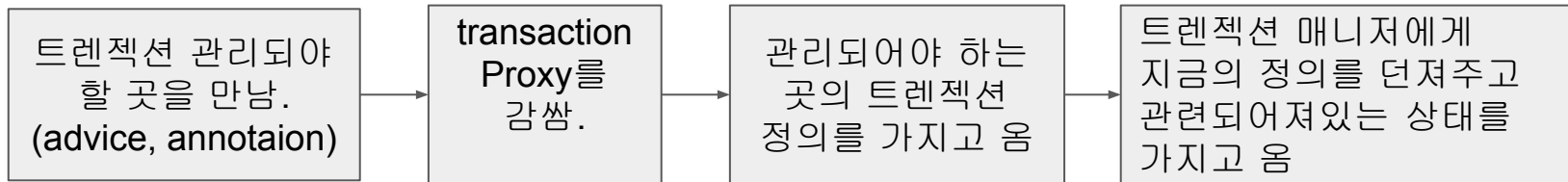
Spring의 transaction관리

- **TransactionDefinition** : 소스에 코딩 되어져있는 트랜잭션 설정 정의
 - 격리(Isolation): 해당 트랜잭션이 다른 트랜잭션의 작업과 격리되는 정도.
 - 전파(Propagation): 전파 정도
 - 시간만료(Timeout): 시간이 만료되기 전에 해당 트랜잭션이 얼마나 오랫동안 실행되고 의존 트랜잭션 인프라스트럭처가 자동으로 롤백하는 지를 나타낸다.
 - 읽기 전용 상태(Read-only status)
 - 이쁜 형태가 없어서 소스로 설명

Spring의 transaction관리

- TransactionStatus

- 트랜잭션의 상태를 나타냄.
- 트랜잭션 정의를 바탕으로 트랜잭션 매니저를 통해서 트랜잭션의 현재 상태를 가지고 옴
- 직접적으로 관리될 때는 어렵지 않음.
- Aop를 사용하여 프록시를 타게 되면 순서와 개념이 흔들림.
- Aop 트랜잭션 관리를 할시 흐름.



Spring의 transaction관리

- TransactionStatus

- **public interface** TransactionStatus **extends** SavepointManager {

- boolean** isNewTransaction();

- boolean** hasSavepoint();

- void** setRollbackOnly();

- boolean** isRollbackOnly();

- void** flush();

- boolean** isCompleted();

- }



Spring의 transaction관리

- 예제(수동관리되는 예제)

```
@Test
public void testTransactionPlatformTransactionManagerBasic(){
    ApplicationContext ctx = new ClassPathXmlApplicationContext("spring/transaction/application-config.xml");

    트랜잭션의 정의를 만들
    DefaultTransactionDefinition def = new DefaultTransactionDefinition();
    정의 이름을 줌
    def.setName("SomeTxName");
    def.setPropagationBehavior(TransactionDefinition.PROPGATION_REQUIRED);

    PlatformTransactionManager txManager = ctx.getBean("txManager", PlatformTransactionManager.class);

    정의를 주면 매니저가 현재 처리되어야 하는 status를 줌.
    TransactionStatus status = txManager.getTransaction(def);

    try {
        // execute your business logic here
    }
    catch (Exception ex) {
        txManager.rollback(status);
    }

    txManager.commit(status);
}
```

Spring의 transaction관리

- 스프링 트랜잭션 설정 시 중요한 점들.
 - 트랜잭션 설정(제일 삽질 많이 함.)
 - 전파 및 고립(두번째로 많이 함.)
 - **Proxy**의 개념 정리 필요(**Aop Proxy**때 같이 설명)
 - 계층간의 개념 확립 필요
 - 다중 **DataSource**일때의 트랜잭션 관리(1부와 2부)
 - 예제로 설명 드림.