@Transcational

> 선언적 트랜잭션
>
> 💡 스프링에서 트랜잭션을 처리하는 방식으로
> 소스코드에 직접 트랜잭션 관련 로직을 넣어두지 않고
> 비지니스 로직에서 완전히 분리 ⇒ AOP를 기반으로 구현

1. 테이블 생성

```
CREATE TABLE aaa (
    num NUMBER
);
```

2. Mapper 생성

```
package com.yedam.app.aop.mapper;

import org.apache.ibatis.annotations.Insert;

public interface AaaMapper {
  // 별도의 xml 파일 없이 해당 메소드가 실행할 SQL문 설정
    @Insert("INSERT INTO aaa VALUES (#{value})")
    public int aaaInsert(String value);
}
```

3. Service 생성

```
// Interface
package com.yedam.app.aop.service;

public interface AaaService {
    public void insert();
}


// Class
package com.yedam.app.aop.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.yedam.app.aop.mapper.AaaMapper;
import com.yedam.app.aop.service.AaaService;

@Service
public class AaaServiceImpl implements AaaService{
    private AaaMapper aaaMapper;
    @Autowired
    AaaServiceImpl(AaaMapper aaaMapper){
        this.aaaMapper = aaaMapper;
    }
```

```
    @Override
    public void insert() {
        aaaMapper.aaaInsert("101");
        aaaMapper.aaaInsert("a101");
    }
}
```

4. 테스트 진행

```
package com.yedam.app;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.yedam.app.aop.service.AaaService;

@SpringBootTest
public class AopTests {
    @Autowired
    AaaService aaaService;

    @Test
    void transcationalTest() {
        aaaService.insert();
    }
}
```

5. @Transcational 적용

```
  //before
    @Override
    public void insert() {
        aaaMapper.aaaInsert("101");
        aaaMapper.aaaInsert("a101");
    }

    //after
    @Transactional
    @Override
    public void insert() {
        aaaMapper.aaaInsert("101");
        aaaMapper.aaaInsert("a101");
    }
```

우선순위	적용 위치	비고
1	클래스 메소드	개별적인 트랜잭션 규칙 구성할때 사용
2	클래스	내부 존재하는 모든 메소드에 적용하므로 공통적인 트랜잭션 규칙 구성할때 사용
3	인터페이스 메소드	
4	인터페이스	

- @Transcational 속성

속성명	**특징**
isolation	일관성없는 데이터 허용 수준을 설정

속성명	특징
noRollbackFor, rollbackFor	특정 예외발생시 rollback 여부를 설정
noRollbackForClassName, rollbackForClassName	특정 클래스이름인 경우 rollback 여부를 설정
propagation(전파속성)	트랜잭션 동작 도중 다른 트랜잭션을 호출할 때, 어떻게 할 것인지 설정
readOnly	트랜잭션을 읽기 전용으로 설정, true면 insert, update, delete 실행 시 예외 발생
timeout, timeoutString	지정한 시간(int)내에 완료되지 않으면 rollback되도록 설정 (단위 : 초,seconds)