리소스 반환과 close()

:Connection이나 PreparedStatement에는 close() 메소드가 있다. 이름으로 보면 열린 것을 닫는다는 의미다. 단순하게 생각하면 만들어진 걸 종료하는 것이라고 볼 수도 있지만 보통 리소스를 반환한다는 의미로 이해하는 것이 좋다. 이들은 보통 풀(pool)방식으로 운영된다.

미리 정해진 풀 안에 제한된 수의 리소스를 만들어 두고 필요할 때 이를 할당하고 반환하면 다시 풀에 넣는 방식으로 운영된다. 요청이 매우 많은 서버환경에서는 매번 새로운 리소스를 생성하는 대신 풀에 미리 만들어 놓은 리소스를 돌려가며 사용하는 편이 훨씬 유리하다. 대신 사용한 리소스는 빠르게 반환해야 한다. 그렇지 않으면 풀에 있는 리소스가 고갈되고 결국 문제가 발생한다.

그래서 이런 JDBC 코드에서는 어떤 상황에서도(예외발생) 가져온 리소스를 반환하도록 try/catch/finally 구문 사용을 권장하고 있다.

하지만 try/catch/finally 문을 사용할 경우 리소스 할당 시에 예외를 처리해야 될 뿐만 아니라 리소스 반환 시에도 예외처리를 해야되며, 각 처리메소드마다 하나하나씩 처리해 줘야 한다는 것이 치명적이다.(중복O)

이를 위한 방식이 템플릿 메소드 패턴을 적용하는 것이다.

템플릿 메소드 패턴이란?

: 상속을 통해 기능을 확장해서 사용하는 부분이다. 변하지 않는 부분은 슈퍼 클래스에 두고 변하는 부분은 추상 메소드로 정의해둬서 서브클래스에서 오버라이드하여 새롭게 정의해 쓰도록 하는 것이다.

템플릿 메소드 패턴의 가장 큰 단점

: 템플릿 메소드는 상위(super) 메소드를 추상화 시켜 하위 메소드에서 공유 메소드를 전달 받고, 추상메소드를 각 자식 메소드 마다 변화시켜 공유 부분은 공유하고 차이가 나는 부분은 개별적으로 만든다. 그러나 상위 메소드를 받기 위해서는 상위 메소드를 abstract화 시켜야 하고, 확장구조가 고정되어 버린다.

전략 패턴의 적용

: 개방 폐쇄 원칙을 잘 지키는 구조이면서도 템플릿 메소드 패턴보다 유연하고 확장성이 뛰어난 것이 오브젝트를 아예 둘로 분리하고 클래스 레벨에서는 인터페이스를 통해서만 의존하도록 만드는 전략패턴이다.

* 일정구조(공유)들은 본래의 DAO를 사용한다는 점에서 공유 부분이 각 클래스마다 존재하는 템플릿 메소드 패턴과의 차이점이 도드라진다.

코드를 이용하는 수동 DI

: JdbcContext를 스프링의 빈으로 등록해서 UserDao에 DI 하는 대신 사용할 수 있는 방법이다.

이 방법을 쓰려면 JdbcContext를 스프링의 빈으로 등록해서 사용했던 첫 번째 이유인 싱글톤으로 만들려는 것은 포기해야 한다. 물론 싱글턴을 포기했다고 해서 JdbcContext 오브젝트를 새로 만드는 무식한 방법을 사용해야 한다는 뜻은 아니다. 조금만 타협해서 DAO마다 하나의 JdbcContext 오브젝트를 갖고 있게 하는 것이다. JdbcContext 내부에는 상태정보가 없기에 수십, 수백개 정도가 만들어진다 해도 메모리에 주는 부담의 거의 없다.

두 번째 이유인 각 클래스들과의 의존성 문제는 직접적으로 선언해줄 수 밖에 없다.

하지만 그냥 dataSource 등을 다양하게 생성하여 사용하는 것이 아닌 JdbcContext에 대한 제어권을 갖고 생성과 관리를 담당하는 UserDao에게 DI까지 맡기는 것이다. 오브젝트를 생성하고 그 의존 오브젝트를 수정자 메소드로 주입해주는 것이 바로 DI의 동작원리이다.

템플릿과 콜백

: UserDao, StatementStrategy, JdbcContext를 이용해 만든 코드는 일종의 전략 패턴이 적용된 것이라고 볼 수 있다. 복잡하지만 바뀌지 않는 일정한 패턴을 갖는 작업 흐름이 존재하고 그 중 일부분만 자주 바꿔서 가용해야 하는 경우에 적합한 구조다. 전략 패턴의 기본 구조에 익명 내부 클래스를 활용한 방식이다. 이런 방식을 스프링에서는 템플릿/콜백 패턴이라고 한다.

전략 패턴의 컨텍스트를 템플릿이라 부르고, 익명 내부 클래스로 만들어지는 오브젝트를 콜백이라고 부른다.

* 템플릿이란?

: 어떤 목적을 위해 미리 만들어둔 모양이 있는 틀을 가르킨다.

* 콜백이란?

: 실행되는 것을 목적으로 다른 오브젝트의 메소드에 전달되는 오브젝트를 말한다.

자바에서는 메소드 자체를 파라미터로 전달할 방법은 없기 때문에 메소드가 담긴 오브젝트를 전달해야 한다. 그래서 펑셔널 오브젝트라고도 한다.

템플릿/콜백의 동작원리

: 템플릿은 고정된 작업 흐름을 가진 코드를 재사용한다는 의미에서 붙인 이름이다. 콜백은 템플릿 안에서 호출되는 것을 목적으로 만들어진 오브젝트를 말한다.

템플릿/콜백의 특징

여러 개의 메소드를 가진 일반적인 인터페이스를 사용할 수 있는 전략 패턴의 전략과 달리 템플릿/콜백 패턴의 콜백은 보통 단일 메소드 인터페이스를 사용한다. 템플릿의 작업 흐름 중 특정 기능을 위해 한 번 호출되는 경우가 일반적 이기 때문이다. 하나의 템플릿에서 여러 가지 종류의 전략을 사용해야 한다면 하나 이상의 콜백 오브젝트를 사용할 수도 있다. 콜백은 일반적으로 하나의 메소드를 가진 인터페이스를 구현한 익명 내부 클래스로 만들어진다고 보면된다.

콜백 인터페이스의 메소드에는 보통 파라미터가 있다. 이 파라미터는 템플릿의 작업 흐름 중에 만들어지는 컨텍스트 정보를 전달받을 때 사용된다.

일반적인 DI라면 템플릿에 인스턴스 변수를 만들어두고 사용할 의존 오브젝트를 수정자 메소드로 받아서 사용할 것이다. 반면에 템플릿/콜백 방식에서는 매번 메소드 단위로 사용할 오브젝트를 새롭게 전달 받는다는 것이 특징이다. 콜백 오브젝트가 내부 클래스로서 자신을 생성한 클라이언트 메소드(UserDao) 내의 정보를 직접 참조한다는 것도 템플릿/콜백의 고유한 특징이다. 클라이언트와 콜백이 강하게 결합된다는 면에서도 일반적인 DI와는 조금 다르다.

템플릿/콜백 설계

1. Test 클래스 생성
2. 기능 클래스 생성(try, catch, finally 연결)
3. 두 클래스 연결
4. 템플릿 설계(try, catch, finally 부분 즉, 예외처리 부분)
5. 콜백 설계(계산 부분을 도출)
6. 기능 클래스에서 콜백 메소드를 오브젝트화 시킨 후,

템플릿 메소드를 콜백 오브젝트를 매개변수로 실어서 호출

1. 템플릿 메소드에서 예외발생 부분은 본인 코드로, 기능 처리 부분은 매개변수인 콜백 오브젝트 내의 메소드를 호출하여 처리 후, 리턴
2. Test 클래스에서 @Before을 사용하여 기능 클래스에 필요한 매개변수 및 초기화 실행
3. Test클래스에서 assertThat으로 호출

본래의 구조 : Test(@Test) -> 기능메서드(구현에 필요한 장치 불러오기(예외처리) -> 기능 실행 -> 리턴)

이후의 구조

: Test(@Before(필요한 초기화 및 경로 구성) @Test(assertThat실행)) -> 기능메서드( 콜백 메소드 오브젝트화 -> 템플릿 메소드 콜백오브젝트를 매개변수로 호출 -> 템플릿 메소드에서 구현에 필요한 장치 불러오기(예외처리) -> 장치를 바탕으로 콜백 메소드 실행 -> 리턴값을 받아서 assertThat에 도로 리턴

템플릿/콜백 패턴만을 활용하면 return Type이 고정된다는 단점이 있다.

이를 해결하기 위해서 우리는 자바5부터 생긴 제네릭스(Generics)를 활용할 수 있다.

1. 인터페이스 메소드에 제네릭 추가

Ex) public interface A<T>{

T 함수명(매개변수(필요하면 T활용가능));

1. 콜백 인터페이스를 상속받는 기능 구현 메소드에서 다채로운 타입 추가

Ex)

A a<String> = new A<String>{

String 메소드명(매개변수 위와 동){

…

}

};

템플릿메소드 호출

1. 템플릿메소드도 제네릭스 처리

Ex)

Public <T> T

스프링이 제공하는 템플릿/콜백 기술

: 스프링이 제공하는 JDBC 코드용 기본 템플릿은 JdbcTemplate이다.

Ex)

Public void deleteAll(){

This.jdbcTemplate.update(

New PreparedStatementCreator(){

Public PreparedStatement createPreparedStatement(Connection c) throws SQLException{

Return c.preparedStatement(“delete from users”);

}

}

}

}

jdbcTemplate.query 및 queryForObject

ex) return this.jdbcTemplate.query(

new PreparedStatementCreator() {

public PreparedStatement createPreparedStatement(Connection c) throws SQLException { // 첫번째 콜백 메서드(쿼리 처리)

return c.prepareStatement("select count(\*) from users");

}

},

new ResultSetExtractor<Integer>() {

public Integer extractData(ResultSet rs) throws SQLException, DataAccessException{ // -> 두번째 콜백 메서드(리턴 값 추출)

rs.next();

return rs.getInt(1);

}

});

ex) return this.jdbcTemplate.queryForObject("select \* from users where name = ?",

new Object[] {name},

new RowMapper<User>() {

public User mapRow(ResultSet rs, int rowNum) throws SQLException {

User user = new User();

user.setId(rs.getInt("id"));

user.setName(rs.getString("name"));

user.setPassword(rs.getString("password"));

return user;

}

});

Query()는 여러 개의 로우가 나오는 일반적인 경우에 쓸 수 있다. Query()의 리턴 타입은 List<T>다., query()는 제네릭 메소드로 타입은 파라미터로 넘기는 RowMapper<T> 콜백 오브젝트에서 결정된다.