데이터베이스 연동 (1) - Mapper, 단위테스트

#01. POJO, JavaBeans, DTO, VO, DAO ?

1. JavaBeans

빌더 형식의 개발도구에서 가시적으로 조작이 가능하고 또한 재사용이 가능한 소프트웨어 컴포넌트이다.

JavaBeans의 관계 (위키백과)

- 클래스는 기본 생성자를 가지고 있어야 한다.
- 클래스의 속성(멤버변수)들은 get, set 혹은 표준 명명법을 따르는 메서드들을 사용해 접근할 수 있어야 한다.

2. POJO 클래스

자바의 순정 기능만을 사용한 클래스

상속, 재정의 등의 규칙이 강제되지 않는 순수한 형태의 기본 클래스

> ex) 콘솔 프로그램의 Main 클래스, JavaBeans

3. DTO (Data Transfer Object)

계층간 데이터를 교환하기 위한 객체

> "계층간"은 A클래스의 메서드가 동작하는 동안 B 클래스의 메서드를 호출할 때 A와 B가 서로 다른 계층이라고 이해하자.

쉽게 풀이하면 메서드를 호출하면서 전달하는 파라미터용 객체

파라미터로 사용하기 위해 객체는 멤버변수를 포함하고 있으며 Java의 표준 코딩 관례에 따라 getter, setter등을 포함한다.

> 결국은 JavaBeans(=POJO) 클래스라는 의미

데이터의 구조를 정의하는 클래스이므로 Model이라고 부르기도 함

4. VO (Value Object)

읽기 전용 객체

DTO에서 setter를 제외한 형태.

5. DAO (Data Access Object)

데이터에 직접적으로 접근하기 위해 저수준의 SQL을 다루는 객체

SQL문에 사용할 파라미터를 DTO형식으로 전달받는다.

> 이 수업에서는 MyBatis의 Mapper가 DAO역할을 수행

#02. 프로젝트 구성

1. 패키지 정보

`kr.hossam.database`

항목	설정 내용
GroupId	`kr.hossam`
ArtifactId	`database`

2. 의존성 설정

프로젝트 생성 과정에서 `dependencies` 항목에 대해 아래의 항목을 선택

항목 이름	구분
Spring Boot DevTools	기존 사용
Spring Boot Acturator Ops	기존 사용
Spring Web	기존 사용
Thymeleaf Template Engines	기존 사용
Lombok	기존 사용
Java Mail Sender	기존 사용
JDBC API	신규 추가
MySQL Driver	신규 추가
MyBatis Framework	신규 추가

3. 추가 설정

1) logback 설정파일 추가

`/src/main/resources/logback-spring.xml` 파일을 추가

(이전 수업에서 생성한 파일을 재사용)

2) UserAgent 라이브러리 설정

`build.gradle`에 의존성 설정 추가

```
implementation 'com.github.ua-parser:uap-java:1.6.1'
```

3) Lombok에 대한 Test 케이스 의존성 추가

```
// test추가
testAnnotationProcessor 'org.projectlombok:lombok'
testCompileOnly 'org.projectlombok:lombok'
```

4) 기본 패키지 추가

패키지	기능
kr.hossam.database.controllers	컨트롤러 클래스
kr.hossam.database.helpers	헬퍼 클래스
kr.hossam.database.interceptors	인터셉터 클래스
kr.hossam.database.mappers	맵퍼 클래스

4. 데이터베이스 연동을 위한 환경설정 추가 - application.properties 파일

기존의 View 설정 이외에 DATABASE 접속 정보가 추가됨

일단은 기본 구성으로 진행 (logback)은 뒤에서 처리함

```
#---------------------------------------------------
# DATABASE 접속 정보
#---------------------------------------------------
# 기본 구성
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/myschool?characterEncoding=UTF8
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=123qwe!@#
```

#03. SQL문 수행을 위한 구성

1. 데이터 구조 정의 (Model 클래스)

kr.hossam.database.models 패키지에 테이블과 1:1로 매칭되는 클래스를 정의한다.

```
package kr.hossam.database.models;

import lombok.Data;

/**
 * 테이블이름 테이블의 구조를 정의하는 클래스
 */
@Data
public class 테이블이름 {
    private int deptNo;     // <-- 테이블 필드이름과 다름에 주의
    private String dname;
```

```
    private String loc;
}
```

2. 저수준의 SQL문 수행 모듈 작성

1) Mapper 구현

`kr.hossam.mappers` 패키지에 `테이블이름.Mapper.java` 형태의 인터페이스를 추가하고 SQL문을 구현한다.

```
package kr.hossam.database.mappers;

import java.util.List;

import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Options;
import org.apache.ibatis.annotations.Result;
import org.apache.ibatis.annotations.ResultMap;
import org.apache.ibatis.annotations.Results;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;

import kr.hossam.database.models.Model클래스;

@Mapper
public interface 테이블이름Mapper {
    @Insert("...")
    // INSERT문에서 필요한 PK에 대한 옵션 정의
    //  --> useGeneratedKeys: AUTO_INCREMENT가 적용된 테이블인 경우 사용
    //  --> keyProperty: 파라미터로 전달되는 MODEL 객체에서 PK에 대응되는 멤버변수
    //  --> keyColumn: 테이블의 Primary Key 컬럼명
    @Options(useGeneratedKeys = true, keyProperty = "...", keyColumn = "...")
    public int insert(Model클래스 input);

    /**
     * UPDATE문을 수행하는 메서드 정의
     * @param input - 수정할 데이터에 대한 모델 객체
     * @return 수정된 데이터 수
     */
    @Update("...")
    public int update(Model클래스 input);

    /**
     * DELETE문을 수행하는 메서드 정의
     * @param input - 삭제할 데이터에 대한 모델 객체
     * @return 수정된 데이터 수
     */
    @Delete("...")
    public int delete(Model클래스 input);
```

```
    /**
     * 단일행 조회를 수행하는 메서드 정의
     * @param input - 조회할 데이터에 대한 모델 객체
     * @return 조회된 데이터
     */
    @Select("...")
    // 조회 결과와 리턴할 MODEL 객체를 연결하기 위한 규칙 정의
    //  --> property : MODEL 객체의 멤버변수 이름
    //  --> column : SELECT문에 명시된 필드 이름 (AS 옵션을 사용한 경우 별칭으로 명시)
    // --> import org.apache.ibatis.annotations.Result;
    // --> import org.apache.ibatis.annotations.Results;
    @Results(id="결과맵이름(직접정함)", value={
        @Result(property="deptNo", column="deptno"),
        @Result(property="dname", column="dname"),
        @Result(property="loc", column="loc")
    })
    public Model클래스 selectItem(Model클래스 input);


    /**
     * 다중행 조회를 수행하는 메서드 정의
     * @param input - 조회 조건을 담고 있는 객체
     * @return 조회 결과를 담은 컬렉션
     */
    @Select("...")
    // 조회 결과와 MODEL 의 맵핑이 이전 규칙과 동일한 경우 id값으로 이전 규칙을 재사용
    @ResultMap("결과맵이름(앞에서 정의한 값 연결)")
    public List<Model클래스> selectList(Model클래스 input);
}
```

2) Mapper에 대한 단위 테스트 구현

`test` 폴더 안에 임의의 패키지와 클래스를 만들고 단위 테스트를 수행한다.

```
package kr.hossam.database.mappers;

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

import kr.hossam.database.models.테이블이름;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@SpringBootTest
public class 테이블이름MapperTest {
    // 테스트 클래스에서는 객체 주입을 사용해야 함
    @Autowired
    private 테이블이름Mapper 테이블이름Mapper;
```

```
    @Test
    @DisplayName("기능명(직접정함)")
    void 메서드이름은_직접정함() {
        // 단위테스트 코드 수행
    }
}
```