Chapter 5-1 Mybatis 사용해보기

Mybatis 소개

Mybatis로 Database 사용해보기

애플리케이션 설정의 형태

application.xml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/demo_schema
    username: demo_user
    password: ********

mybatis:
  mapper-locations: "classpath:mybatis/mappers/*.xml"
  configration:
    map-underscore-to-camel-case: true
```

`url`: jdbc:mysql://127.0.0.1:3306/`demo_schema` - 어떤 스키마를 사용할 것인지 끝에 작성

`mapper-locations`: "classpath:mybatis/mappers/*.xml"

resources 폴더를 root로 보고 mabatis/mappers에서 xml 파일을 모두 찾을 수 있음.

configration:
  map-underscore-to-camel-case: true

대부분의 데이터베이스는 underscore 즉, `camelCase` 로 !

→ 데이터 자체 내용을 변환하는 것이 아닌, 인자로서 활용되는 칼럼 등을 치환해줌.

`snake_case`

`camelCase`

`PascalCase`

JAVA는 일반적으로 camelCase를 많이 쓰고, MySQL은 snake_case를 많이 사용하므로 간극을 줄이기 위한 설정

post-mapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="dev.hsooovn.mybatis.mapper.PostMapper">
  <insert id="createPost" parameterType="dev.hsooovn.mybatis.dto.PostDto">
        insert into POST(title, content, writer, board)
        values (title, content, writer, board)
    </insert>
</mapper>
```

mapper

어떤 mapper class가 있는지 지정해줌.

`mapper namespace`에 있는 인터페이스와 `insert id=""` 부분에서 어떤 함수의 sql문을 연결하는지 결정

`parameterType`은 createPost 함수가 받아들일 파라미터 타입을 의미, 사용하게될 파라미터를 의미

PosstMapper

```
package dev.hsooovn.mybatis.mapper;

import dev.hsooovn.mybatis.dto.PostDto;

public interface PostMapper {
    int createPost(PostDto dto);
}
```

`int` : insert, update, delete의 경우 int로 ! 결과로서 몇개의 로우가 조작 받았는지 결과로 돌려주기 때문에

PostDto

```
package dev.hsooovn.mybatis.dto;

public class PostDto {
// table의 컬럼 값과 동일한 이름으로
    private int id;
    private String title;
```

```
    private String content;
    private String writer;
    private int board;
}
```

post-mapper.xml 수정

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="dev.hsooovn.mybatis.mapper.PostMapper">
    <insert id="createPost" parameterType="dev.hsooovn.mybatis.dto.PostDto">
        insert into POST(title, content, writer, board)
        values (#{title}, #{content}, #{writer}, ${board})
    </insert>
</mapper>
```

문자열의 경우: #{}

숫자(?): ${}

post-mapper.xml

```
<select id="readPost"
        parameterType="int"
        resultType="dev.hsooovn.mybatis.dto.PostDto">
    select * from post where id = ${id}
</select>
```

`parameterType="int"` : 인자를 하나 가져올 때 프라이머리 키를 가져옴

`resultType="dev.hsooovn.mybatis.dto.PostDto"` : 결과가 테이블 형태로 나올거고 사용하고 싶은 형태의 오브젝트로 만들어줘야함

`select문` : id가 동일한 post를 돌려주게 된다.

PostDao

```
package dev.hsooovn.mybatis.dao;

import org.apache.ibatis.session.SqlSessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
```

```
@Repository
public class PostDao {
    private final SqlSessionFactory sessionFactory;

    public PostDad(
            @Autowired SqlSessionFactory sessionFactory
    ){
        this.sessionFactory = sessionFactory;
    }
}
```

`@Repository`를 붙여줌으로서 데이터를 주고 받기 위해서 존재하는 클래스라고 이해할 수 있음(?)

```
SqlSession session = sessionFactory.openSession();
PostMapper mapper = session.getMapper(PostMapper.class);
int rowAffected = mapper.createPost(dto);
session.close();
return rowAffected;
```

```
try (SqlSession session = sessionFactory.openSession()){
    PostMapper mapper = session.getMapper(PostMapper.class);
    return mapper.createPost(dto);
}
```

하단의 코드로 사용하면 `session close`가 자동으로!

sessionFactory를 왜 열고 닫고 ?

→ thread safe 하지 않음 요청이 빠르게 두 번 연속으로 들어오면 서로에게 영향 있을 수 도

TestComponent

```
package dev.hsooovn.mybatis;

import dev.hsooovn.mybatis.dao.PostDao;
import dev.hsooovn.mybatis.dto.PostDto;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.List;
```

```
@Component
public class TestComponent {
    private final PostDao postDao;
    public TestComponent(
            @Autowired PostDao postDao
    ){
        this.postDao = postDao;
        PostDto newPost = new PostDto();
        newPost.setTitle("From Mybatis");
        newPost.setContent("Hello Database!");
        newPost.setWriter("hsooovn");
        newPost.setBoard(1);
        this.postDao.createPost(newPost);

        List<PostDto> postDtoList = this.postDao.readPostAll();
        System.out.println(postDtoList.get(postDtoList.size() - 1)); //추가가 됐는지 확인

        PostDto firstPost = postDtoList.get(0);
        firstPost.setContent("Update From Mybatis!");
        postDao.updatePost(firstPost);

        System.out.println(this.postDao.readPost(firstPost.getId()));
    }
}
```

dao : data access object

데이터를 주고 받는 기능을 해주기 위한 객체, 클래스들을 불러옴

dto : 실제 데이터를 담기위한 오브젝트들

mapper : xml 설정에 따라 연결됨

Test Component 추가 시 생긴 오류

: Type interface dev.hsooovn.mybatis.mapper.PostMapper is not known to the MapperRegistry.