Spring + Mybatis

mvc 패턴 활용 CRUD 기능 구현 가이드

01. MVC 패턴 웹 개발

웹 개발의 코드 구현 방식을 말하며 모델, 뷰, 컨트롤러의 역할을 나누어 코드를 작성하는 방법론이다.

- C (Controller) – 페이지(html) 간의 이동을 결정하는 클래스로 요청에 따른 응답 페이지를 정의하는 관제탑 역할의 클래스.
- M (Model) – 여기서 말하는 모델은 기능을 어떻게 제공할 것인지에 대한 형태를 일컫는 말로써, 목표를 위한 핵심 기능을 제공하는 클래스를 말한다. 웹 개발에서의 핵심기능이란 데이터베이스와 연동한 데이터 조작 기능을 말함으로 결론적으로 데이터베이스 기능을 모아놓은 클래스(수업에서는 이런 클래스명에 ServiceImpl을 사용)를 의미한다.
- V (View) – 기능에 대한 결과를 시각적으로 화면에 어떻게 보여줄지를 결정하는 역할. 쉽게 말해 html이라 생각하면 된다.

다음 페이지부터는 아래와 같이 크게 세가지 분류로 Spring 웹 개발 가이드를 제시하려 한다.

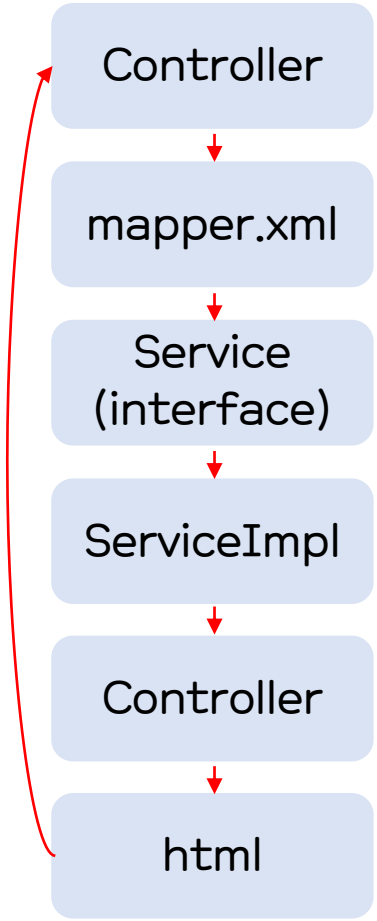
본문 : Spring 웹 개발 코드의 전반적인 코드 작성 흐름과 구체적 코드 작성 가이드

부록1: Spring과 Mybatis, Maria DB 를 연동한 프로젝트 생성방법과 초기 설정 가이드

이 가이드는 웹 개발에서 가장 기초 기능인 CRUD 개발에 초점이 맞추어져 있다. 웹 개발의 모든 내용을 담고 있는 것이 아니다. 하지만 이 가이드에서 소개하는 내용이 기둥이 될 것이며, 기둥이 탄탄해야 덕지덕지 다른 추가 코드를 적용할 수 있다. 중요함을 100번 강조해도 지나치지 않는 내용이다. 어떻게 해서든 반드시 숙지하길 바란다. 다만, 아직은 자바, 데이터베이스, Spring 지식이 부족하기 때문에 가이드를 읽어도 이해가 어려운 부분이 있을 것이다. 그럴 때엔 제발!! 반드시!! 질문을 해서 이해하려고 노력해야 한다.

P.S) 어렵죠? 하지만 전 여러분들이 이까짓 코드 작성 따위에 질 거 라고 생각하지 않아요. 어렵다고 포기하면 언제나 포기만 할 겁니다. 스스로를 이겨내 보세요! 응원합니다!

02. MVC 패턴 웹 개발의 전반적인 코드 작성 흐름



01. html 요청을 받을 메서드 정의 및 응답 페이지(html) 지정
02. 응답 html로 이동 시 쿼리 작업이 필요한지 판단(쿼리 작업이 필요하면 03번부터 수행, 아니면 08번 부터 수행)

03. 필요한 쿼리 작성(이때 쿼리 실행을 위해 어떤 데이터가 필요한지 판단)

04. 작성한 쿼리를 실행할 메서드 정의

05. interface에서 정의한 메서드 구현

06. ServiceImpl 클래스에서 정의한 메서드를 호출
07. 필요에 따라 메서드 호출 결과 데이터를 html로 전달

08. html 작성
09. 01번부터 반복 수행

위의 방식이 100% 맞는 것은 아니다. 상황에 따라 코드 작성 흐름이 달라질 수 있다. 그렇게 때문에 위의 가이드는 학습 초반 무엇을 할지 몰라 코드 작성에 어려움을 겪을 때 많은 도움이 될 것이다. 위 순서의 코드 흐름이 이해되면 그때부터는 여러분들이 응용하여 유연하게 코드를 작성하길 바란다. 다음 페이지부터는 위에서 언급한 01 ~ 05번 까지의 상세 코드 작성 가이드를 안내하겠다.

03. 개발 흐름 상세 | 01. html 요청을 받을 메소드 정의 및 응답(html) 페이지 지정-1

1. 컨트롤러에서 요청에 대한 응답을 처리할 메소드를 정의한다.

```
public String boardList(){

}
```

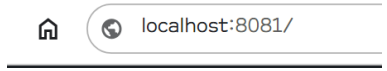
1). 접근 제한자는 public, 메소드명은 자유다.

2). 이러한 메소드는 return에 이동할 html명을 문자열로 작성하기에 return 타입은 무조건 String이다. (나중에 배울 비동기 통신에서는 리턴 타입이 바뀔 수 있음)

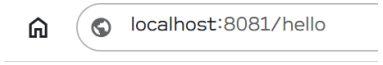
2. 정의한 메소드를 요청된 url과 매핑(연결)한다.

url을 일단 인터넷 주소라고 생각하자. url이 요청되는 방법은 크게 아래와 같이 존재한다.

1). 웹 브라우저에 직접 주소를 입력하여 url을 요청한 경우



요청 url이 '/' 인 경우



요청 url이 '/hello'인 경우

2). html의 a 태그로 url을 요청한 경우

```
<a href="/boardList">게시글 목록 페이지</a>
```

요청 url이 '/boardList'인 경우

3). html의 form 태그로 url을 요청한 경우

```
<form action="/boardDetail" method="post">
```

요청 url이 '/boardDetail'인 경우

4). 자바스크립트의 location.href 명령으로 url을 요청한 경우

```
<script>
    location.href = '/updateBoard';
</script>
```

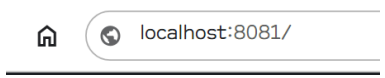
요청 url이 '/boardDetail'인 경우

위와 같은 url 요청이 들어오면 해당 요청에 대한 응답할 메소드를 매핑해야 한다. 컨트롤러에서 정의한 메소드와 요청 url을 매핑하기 위해서는 메소드 위의 다음의 어노테이션 중 하나를 사용한다.

1). @GetMapping("/요청url")

요청 방식이 웹 브라우저에 주소 직접 입력, html의 a태그 사용,

자바스크립트의 location.href 명령어로 인해 요청될 때 사용.



→

```
@GetMapping("/")
public String boardList(){

}
```

```
<a href="/boardList">게시글 목록 페이지</a>
```

→

```
@GetMapping("/boardList")
public String boardList(){

}
```

```
<script>
    location.href = '/updateBoard';
</script>
```

→

```
@GetMapping("/updateBoard")
public String boardList(){

}
```

2). @PostMapping("/요청url")

html에서 form태그를 사용하며 form태그의 action속성이 "post"로 요청될 때 사용

```
<form action="/boardDetail" method="post">
```

→

```
@PostMapping("/updateBoard")
public String boardList(){

}
```

03. 개발 흐름 상세 | 01. html 요청을 받을 메소드 정의 및 응답(html) 페이지 지정-2

3. 메서드 실행 후 응답 할 html 지정

컨트롤러에서 정의한 메서드의 마지막은 항상 응답(이동)할 html 파일명을 문자열로 리턴하는 것으로 마무리한다.

```
@GetMapping("/")
public String boardList(){
    return "board_list";
}
```

위 코드의 전체 해석은 ‘ / 요청이 들어오면 boardList 메서드를 실행하고, 메서드 실행 후 board_list.html 파일로 응답(이동)하라’이다.

이때 코드에서 보듯이 응답(이동)할 파일확장자(.html)는 생략하며, 생략할 경우 자동으로 리턴하는 문자열 뒤에 ‘.html’이 붙는다.

4. 추가 사항

컨트롤러의 메서드 마지막은 대부분 응답(이동)할 html 파일명을 문자열로 리턴하지만 메서드 실행 후 html로 응답하는 것이 아니라 컨트롤러의 다른 메서드를 실행시키고 싶을 때가 생각보다 많이 발생한다. 이럴 때는 return 구문에 “redirect:/요청경로”로 작성.

ex› return “boardList” -› boardList.html 파일로 이동하라는 명령

return “redirect:/boardList” -› 컨트롤러의 url 매핑이 “/boardList”인 메서드를 실행하라는 명령

주의사항!

컨트롤러에서 html로 이동하며 데이터를 전달할 때는 model.addAtrribute() 메서드를 사용하지만 redirect를 사용하여 컨트롤러의 다른 메서드를 호출할 때에는 model.addAtrribute() 메서드로 데이터를 전달할 수 없다.

여기까지 했다면 다음에 할 것은 ’02. 응답 html로 이동 시 쿼리 작업이 필요한지 판단’이다. 이는 spring에서 기능을 제공하거나, 문법을 익히는 것이 아니며 상식?에 가까운 부분이기 때문에 설명은 생략한다. 앞으로 웹 개발을 공부해나가면 자연스레 판단할 수 있을 것이다.

03. 개발 흐름 상세 | 03. 필요한 쿼리 작성-1

1. 쿼리는 mapper.xml 파일에 작성한다.

해당 파일은 class 및 html 파일처럼 프로젝트에서 새로 생성하지 말고, 공유폴더에 올려놓은 mapper.xml 파일을 copy해서 사용하도록 한다.

해당 파일에는 <mapper></mapper> 태그가 있고 해당 mapper 태그 안에 모든 내용을 작성한다. mapper 태그 안에는 크게 resultMap 태그와 쿼리 작성에 필요한 태그(select, insert, update, delete)가 존재한다.

전체적인 형태는 다음과 같다.

```
<mapper namespace="boardMapper">
    <insert id="insertBoard">
        INSERT 쿼리 작성
    </insert>

    <delete id="deleteBoard">
        DELETE 쿼리 작성
    </delete>

    <update id="updateBoard">
        UPDATE 쿼리 작성
    </update>

    <select id="selectBoard" resultMap="">
        SELECT 쿼리 작성
    </select>
</mapper>
```

해당 mapper 파일을 지칭하는 이름으로 원하는 대로 작성할 수 있다.
ServiceImpl 클래스에서 쿼리 실행 시 사용한다.
해당 내용은 ServiceImple 클래스 구현 방법에서 다루겠다.

쿼리 작성은 쿼리를 감싸는 태그 안에 작성한다. 각 쿼리에 대응하는 태그를 사용하면 된다.

각 쿼리문을 감싸는 태그에는 반드시 id 속성 값을 지정해야 한다. 기본적으로 id 속성값은 절대 중복 될 수 없다. 다만 mapper 파일이 여러 개라면 각각의 mapper 파일 안에서만 id가 중복 불가이기 때문에 서로 다른 mapper 파일에는 id가 같은 태그가 존재할 수 있다.

select 태그에만 유일하게 resultMap 혹은 resultType 속성을 작성해야 한다.
resultMap과 resultType 속성은 쿼리 실행결과 조회된 데이터를 자바로 가져올 때 어떤 자료형에 담아 가져올지를 정해주는 속성이다. 자세한 것은 Service 인터페이스 작성에서 알아보겠다.

03. 개발 흐름 상세 | 03. 필요한 쿼리 작성-2

2. 쿼리 작성 및 빈 값 표현하기

쿼리 작성 문법은 본 가이드에서 안내하지 않는다. 여기에서는 쿼리 실행 시 우리가 채워줘야 하는 내용을 어떻게 표현하는지만 알아보겠다.

```
<select id="selectBoard" resultMap="board">
    SELECT BOARD_NUM
        , BOARD_TITLE
        , CONTENT
    FROM BOARD
    WHERE BOARD_NUM = #{boardNum}
</select>
```

빨간 네모칸이 쿼리 실행 시 우리가 값을 채워줘야 하는 부분을 표현한 것이다.

보는 것과 같이 채워줘야하는 부분은 #{} 형태로 작성한다.

(추후 배우겠지만 ${}형태로 사용해야 하는 경우도 존재한다.)

#{} 안의 단어를 작성하는 것은 아무렇게나 작성하는 것이 아니라 문법이 존재한다.

다만, 글을 통해 해당 내용을 전달하기에는 조금 무리가 있으므로 본 내용은 수업시간에 안내하겠다. 일단 채워야하는 컬럼명에 매칭되는 VO클래스의 변수명을 #{} 안에 작성한다고 생각하자. 이렇게 하면 절대 코드 오류가 발생하지는 않는다.

쿼리 문법을 익히지 못해 쿼리를 작성하지 못한다면 어떤 값을 채워야하는지 파악 조차 불가능하다. 현재 우리가 사용하는 쿼리는 아주 쉬운 내용만 하고 있기 때문에 조금만 노력해서 쿼리 작성에 어려움이 없다고 하자.

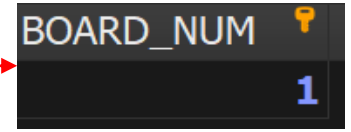
3. <select> 태그의 resultType 속성과 resultMap 속성

다른 쿼리와 다르게 select 쿼리는 조회 결과 데이터가 존재한다. 이렇기 때문에 select 쿼리 결과 데이터를 자바로 가져와야 하고, 조회 결과를 자바의 어떤 자료형에 담아 가져올지 결정하는 것이 resultType과 resultMap 속성이다.

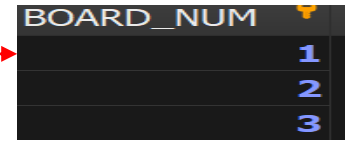
1). resultType 속성

resultType 속성은 자바에서 제공하는 자료형으로 조회 결과 데이터를 가져올 때 사용하는 속성이다. 대표적으로는 String, int, Map 등이 있으며 Map을 사용하여 조회 데이터를 가져오는 것은 기본이 익숙해지면 수업시간에 학습하도록 한다. 여기서는 쿼리 조회 결과 데이터가 어떤 형식일때 resultType 속성에 String과 int를 사용하는지 알아보겠다. 아래의 쿼리문과 쿼리 결과 조회 데이터를 살펴보자.

```
SELECT BOARD_NUM
FROM BOARD
WHERE BOARD_NUM = 1;
```




```
SELECT BOARD_NUM
FROM BOARD;
```



위와 같이 하나의 컬럼만 조회하며 조회한 컬럼이 MariaDB에서 INT형일 때는 조회 데이터를 자바로 가져올 때 자바의 int 자료형으로 가져올 수 있다. 여기서 중요한 것은 INT형 컬럼 하나를 조회 했다는 것이다. 조회 결과 데이터의 행의 개수가 하나이든 여러개든 조회 결과 행의 개수는 상관없다. 이럴 때는 resultType="int" 로 작성한다.

03. 개발 흐름 상세 | 03. 필요한 쿼리 작성-3

다음 쿼리를 보겠다.

```
SELECT BOARD_TITLE
FROM BOARD
WHERE BOARD_NUM = 1;
```

BOARD_TITLE
제목1

```
SELECT BOARD_TITLE
FROM BOARD;
```

BOARD_TITLE
제목1
제목2
제목3

위와 같이 하나의 컬럼만 조회하며 조회한 컬럼이 MariaDB에서 VARCHAR형일 때는 조회 데이터를 자바로 가져올 때 자바의 String 자료형으로 가져올 수 있다. 여기서 중요한 것은 VARCHAR형 컬럼 하나를 조회 했다는 것이다. 조회 결과 데이터의 행의 개수가 하나이든 여러개든 조회 결과 행의 개수는 상관없다. 이럴 때는 resultType="String" 로 작성한다.

2). resultMap 속성

resultType을 사용할 때는 조회 쿼리문에서 조회하는 컬럼이 하나일 때라 위에서 말했다. resultMap은 여러 컬럼을 조회할 때 사용한다. 여러 컬럼을 조회한 조회 결과 행 하나를 자바의 int, String type으로 가져올 수 없기 때문에 조회 결과 하나의 행 데이터를 자바로 가져올 수 있는(저장할 수 있는) class형 자료형을 만들어줘야 한다.

```
SELECT BOARD_NUM
FROM BOARD;
```

BOARD_NUM
1
2
3

위의 쿼리는 조회 결과 하나의 행 데이터를 자바의 int로 가져올 수 있는 경우이다.

```
SELECT BOARD_TITLE
FROM BOARD;
```

BOARD_TITLE
제목1
제목2
제목3

위의 쿼리는 조회 결과 하나의 행 데이터를 자바의 String으로 가져올 수 있는 경우이다.

이런 경우 조회 결과 데이터를 자바로 가져오기 위해 resultType을 사용한다고 했다.

아래의 쿼리를 보겠다.

```
SELECT BOARD_NUM, BOARD_TITLE, READ_CNT
FROM BOARD;
```

BOARD_NUM	BOARD_TITLE	READ_CNT
1	제목1	0
2	제목2	3
3	제목3	0

위와 같은 쿼리(여러 컬럼을 조회한 쿼리)의 결과 데이터 하나의 행 정보를 자바의 int, String으로 가져오는 것은 불가능하다. (*int는 정수 하나를 저장할 수 있는 공간, String은 문자열 하나를 저장할 수 있는 공간인데 위의 예시에서는 하나의 행으로 정수2개, 문자열 1개의 결과가 나왔기 때문*)

따라서 이렇게 여러 컬럼을 조회한 결과를 자바로 가져오기 위해서는 우선적으로 하나의 행 결과를 저장할 수 있는 자료형(class)를 만들어야 한다.

03. 개발 흐름 상세 | 03. 필요한 쿼리 작성-4

3. select 조회 결과를 저장할 class 구현

```
SELECT BOARD_NUM, BOARD_TITLE, READ_CNT
FROM BOARD;
```

BOARD_NUM	BOARD_TITLE	READ_CNT
1	제목1	0
2	제목2	3
3	제목3	0

앞서 언급한 쿼리 결과를 보면 BOARD_NUM, BOARD_TITLE, READ_CNT 세 개의 컬럼 데이터를 조회하였다. 여기서 중요한 것은 컬럼명이 아니다. 하나의 행 결과로 정수 데이터 2개, 문자열 데이터 하나를 1개가 조회되었다는 것이 중요하다!

그렇다면 정수 두 개, 문자열 하나를 저장할 수 있는 자료형인 클래스를 만들어주면 된다. 다음과 같이 말이다.

```
public class SelectData {
    private int aa;
    private int bb;
    private String cc;
}
```

위의 SelectData 클래스는 변수(변수 하나는 하나의 데이터를 저장할 수 있는 공간이다)로 정수를 저장할 수 있는 변수 두 개, 문자열을 저장할 수 있는 변수 한 개가 있으므로 위 쿼리를 조회 결과 데이터 하나의 행을 저장할 수 있는 자료형이다.

하지만 클래스를 이렇게 만들면 오류는 나지 않지만 개발자가 신경 쓸 것이 더욱 많아진다.

BOARD_NUM 컬럼의 조회결과는 INT형이기 때문에 SelectData 클래스의 aa 혹은 bb 변수에 저장할 수 있다.

aa 혹은 bb 변수 중 어느 변수에 BOARD_NUM 컬럼의 조회 데이터를 저장할지 개발자가 알아서 결정하고 기억만 하고 있으면 전혀 문제가 없다는 말이다. 하지만 이런식이라면 개발자가 기억하고 있어야 할 것이 얼마나 많아지겠는가. 그렇기에 조회한 컬럼의 데이터를 어떤 변수에 저장할지 구지 기억하고 있지 않아도 되도록 변수명을 잘 만들어주는 것이 중요하다. 이렇게 말이다.

```
public class BoardVO {
    private int boardNum;
    private String boardTitle;
    private int readCnt;
}
```

여기서 선언한 변수 boardNum은 BOARD_NUM이라는 컬럼으로 조회된 데이터를 저장하기 위한 변수이다. boardTitle 변수는 BOARD_TITLE 컬럼으로 조회된 데이터를 저장하기 위한 변수이다. 더 이상 설명이 필요한가?

추가적으로 클래스명(BoardVO) 또한 아무렇게나 지은 것이 아니라 실제 테이블명과 동일하게 작성한 것이다. 테이블명 BOARD에 VO라는 문자만 추가한 것 뿐이다.

여기서 VO는 Value Object의 줄임말로써 '값을 가진 객체'라고 해석한다.

실제로 이 클래스를 만든 이유는 조회 결과 데이터(값)를 가지기 위해 만든 것이 아닌가.

위에서 언급했든 VO클래스는 DB에서 조회된 결과를 자바로 가져오기 위해 구현한다.

그래서 조회되는 컬럼명에 맞춰 변수를 선언하면 되지만 조회되는 컬럼은 언제든 변할 수 있기 때문에 데이터베이스의 테이블에 있는 모든 컬럼과 매칭되는 변수를 다 만들어주는 것이 좋다.

03. 개발 흐름 상세 | 03. 필요한 쿼리 작성-5

위에서 언급했든 VO클래스는 DB에서 조회된 결과를 자바로 가져오기 위해 구현한다. 그래서 조회되는 컬럼명에 맞춰 변수를 선언하면 되지만 조회되는 컬럼은 언제든 변할 수 있기 때문에 데이터베이스의 테이블에 있는 모든 컬럼과 매칭되는 변수를 다 만들어주는 것이 좋다. 아래는 하나의 테이블과 그 테이블에서 조회한 데이터를 자바로 가져오기 위해 만든 class이다.

```
CREATE TABLE BOARD (
   BOARD_NUM INT AUTO_INCREMENT PRIMARY KEY
   , BOARD_TITLE VARCHAR(20) NOT NULL
   , BOARD_CONTENT VARCHAR(50)
   , CREATE_DATE DATETIME DEFAULT CURRENT_TIMESTAMP
   , READ_CNT INT DEFAULT 0
);
```

게시글 정보를 저장하는 테이블

```
@Getter
@Setter
@ToString
public class BoardVO {
    private int boardNum;
    private String boardTitle;
    private String boardContent;
    private String createDate;
    private int readCnt;
}
```

BOARD 테이블에서 조회한 결과를 가져올 VO클래스

VO클래스에는 테이블의 각 컬럼과 연결할 변수를 만들어준다. 이때 Maria DB의 자료형에 맞게 class의 자료형도 정의해준다.

위에서 보듯 DB에는 날짜 데이터에 맞는 자료형 DATETIME이 존재한다. 날짜 데이터를 자바에서 받을 때는 문자열로 받도록 한다.

VO 클래스에서 테이블과 연결되는 각 변수를 선언했다면 반드시 모든 변수에 대한 gettet, setter, toString 메소드를 lombok을 이용해 만들어 준다.

03. 개발 흐름 상세 | 03. 필요한 쿼리 작성-6

4. VO클래스와 테이블 매핑하기

이렇게 VO 클래스를 만들었다면 아래와 같이 여러 컬럼이 조회 결과 중 하나의 행 데이터를 자바로 가져올 수 있는 자료형을 만든 것이다.

```
SELECT BOARD_NUM, BOARD_TITLE, READ_CNT
FROM BOARD;
```

BOARD_NUM	BOARD_TITLE	READ_CNT
1	제목1	0
2	제목2	3
3	제목3	0

```
public class BoardVO {
    private int boardNum;
    private String boardTitle;
    private int readCnt;
}
```

하지만 이렇게 테이블과 매칭되는 클래스를 만들었다고 해서 조회 결과를 우리가 만든 클래스로 가져올 수 있는 것은 아니다. 지금까지 작성한 내용은 개발자가 스스로 정의한 내용이지 컴퓨터에게 '내가 만든 클래스로 조회 결과를 가져올거야' 라는 명령은 하지 않았기 때문이다. 우리가 만든 클래스로 조회 결과를 가져오기 위해서는 mapper.xml 파일에 테이블과 class를 연결시켜주는 작업이 필요하며 이것을 resultMap을 만드는 것으로 구현할 수 있다.

```
<mapper namespace="boardMapper">

    <resultMap id="board" type="com.green.Board.vo.BoardVO">
        <result  column="BOARD_NUM" property="boardNum"/>
        <result  column="BOARD_TITLE" property="boardTitle"/>
        <result  column="BOARD_CONTENT" property="boardContent"/>
        <result  column="CREATE_DATE" property="createDate"/>
        <result  column="READ_CNT" property="readCnt"/>
    </resultMap>

</mapper>
```

resultMap은 위와 같이 mapper.xml 파일의 <mapper> 태그 안에서 작성한다. resultMap 태그 안에서 다시 테이블의 컬럼 개수만큼의 <result> 태그를 선언한다. <result> 태그 안의 column 속성에는 연결할 테이블의 컬럼명을 작성하고, property 속성에는 VO클래스에서 컬럼과의 연결을 위해 정의한 변수명을 작성한다. 또한, property 속성에 작성한 변수가 어떤 class에 정의되어 있는지 알려주기 위해 <resultMap> 태그의 type 속성에 prpperty에서 작성한 변수가 정의된 VO클래스를 패키지명부터 작성하면 된다. id 속성에는 이렇게 테이블과 VO클래스 연결을 위해 만든 resultMap의 이름을 부여하는 것이라 생각하면 되고, 당연히 다른 resultMap과 id는 중복될 수 없다.

03. 개발 흐름 상세 | 03. 필요한 쿼리 작성-7

```
<select id="selectBoard" resultMap="board">
    SELECT BOARD_NUM
        , BOARD_TITLE
        , CONTENT
    FROM BOARD
    WHERE BOARD_NUM = #{boardNum}
</select>
```

그럼 이제 위의 쿼리 결과를 어떻게 가져올 것인지 정의한 내용은 조금은 이해가 되는가.

위의 쿼리는 쿼리 실행 시 여러 컬럼을 조회하고 있으며 이렇게 조회되는 하나의 행 결과를 자바의 int, String을 통해 가져올 수 없었다. 그래서 위의 쿼리의 조회 결과 하나의 행 데이터를 자바로 가져오기 위해 VO 클래스를 만들었고, <resultMap> 작성을 통해 BOARD 테이블에서 조회된 데이터를 BoardVO 클래스를 통해 자바로 가져오겠다고 정의했다. 그리고 이러한 정의를 가진 <resultMap>의 id 속성을 'board'라고 지정했다.

위 쿼리문의 <select> 태그에 resultMap 속성이 'board'인 것이 왜인지 이해가 되길 바란다.

03. 개발 흐름 상세 | 04. 작성한 쿼리를 실행할 메서드 정의-1

mapper.xml 파일에서 작성한 쿼리를 실행할 메서드를 interface에서 정의한다.

우리는 수업 시간에 이러한 ineterface의 이름에 Service라는 이름을 사용하고 있다.

insterface 안에는 메서드의 정의 구문만 있기 때문에 쿼리를 실행할 메서드를 정의하는 방법, 그 중에서도 메서드의 리턴 타입, 매개변수 선언하는 방법만 익히면 된다.

(지금부터 제시하는 방법이 무조건적으로 정답이거나 효율이 좋은 것은 아니다. 학습 초반 아직 이해가 되지 않는 부분이 많아 코드 작성에 어려움을 겪을 테니 그때 참고하길 바란다. 본 가이드를 통해 메서드를 정의하는 것을 어느정도 이해했다면 여러분들이 유동적으로 리턴 타입 및 매개변수를 지정할 수 있을 것이다.)

인터페이스에서 메서드의 리턴 타입, 매개변수를 지정하기 위해선 반드시 쿼리가 작성되어 있어야 한다. 실행할 쿼리를 작성할 수 있다는 전재하에 설명하겠다.

우선 메서드의 리턴 타입, 매개변수가 각각 어떤 역할을 하는지 숙지한다.

1). 리턴 타입 : 쿼리 실행 결과를 자바의 어떤 자료형에 받아 올 것인지 결정하는 역할

2). 매개변수 : 쿼리 실행 시 채워줘야 할 값이 있다면 그 데이터를 쿼리 실행 시 전달하는 역할

그럼 리턴 타입을 지정하는 방법, 매개변수를 지정하는 방법을 차례대로 살펴보겠다.

1. 메서드의 리턴 타입 결정하기

메서드의 리턴 타입은 쿼리의 실행 결과를 자바로 가져올 때 어떤 자료형에 담아 가져올 지 결정하는 것이다. 그래서 insert, update, delete, select 등의 각 쿼리의 실행 결과에 따라 유동적으로 리턴 타입은 변경된다. 다행히 아래 가이드하는 방법대로 리턴 타입을 작성하면 큰 무리가 없을 것이다. 다만, 리턴 타입이 이해가 되었다면 쿼리 결과를 받아올 수 있도록 리턴 타입을 다양하게 사용해보길 바란다.

1). insert, update, delete의 쿼리 실행의 리턴 타입

insert, update, delete 쿼리를 heidiSQL에서 실행하면 쿼리 실행은 되지만 결과 데이터가 나타나지 않는다. 당연하다. 위 세 종류의 쿼리는 어떤 데이터를 조회하는 쿼리가 아니기 때문이다. 그렇다면 리턴 타입은 쿼리 실행 결과를 어떤 자료형에 담아 자바로 가져 올 것인지를 결정하는 것이라 했는데 insert, update, delete의 쿼리 결과는 무엇일까? 바로 영향을 받은 데이터 행의 개수가 쿼리의 결과이다. 다음 장에서 예시를 보자.

03. 개발 흐름 상세 | 04. 작성한 쿼리를 실행할 메서드 정의-2

BOARD_NUM	BOARD_TITLE	WRITER
1	첫번째	김자바
2	두번째	김자바
3	345	김자바

테이블명 : BOARD

위와 같은 데이터가 존재 할 때

DELETE FROM BAORD WHERE BOARD_NUM <= 2;

위의 쿼리가 실행되면 삭제되는 데이터는 2행이다. 따라서 쿼리 결과 2개 행이 쿼리의 영향을 받은 것이고 '2행이 삭제되었다' 가 해당 쿼리의 결과인 것이다. 따라서, 위 쿼리의 실행 결과를 int로 리턴받으면 2가 나온다.

한가지 예를 더 보자.

INSERT INTO BOARD VALUES (4, '네번째', '관리자');

위의 쿼리를 실행하면 1행이 테이블에 삽입된다. 따라서, 위 쿼리의 실행 결과를 int로 받으면 1이 나온다.

결과적으로 insert, update, delete의 쿼리 결과는 쿼리의 영향을 받은 행의 갯수고, 이러한 개수는 정수이기 때문에 자바로 쿼리 결과를 받아오는 메서드의 리턴 타입으로 int를 사용한다. 하지만 int로 리턴 받은 데이터로 특별한 기능을 만드는 것이 아니라면 리턴 데이터가 불필요 때문에 리턴 타입을 void로 선언하여 구지 int형 데이터를 받아오지 않을 때도 많다.

결론 : insert, update, delete 메서드의 리턴 타입은 int 혹은 void를 사용한다.

2). select 쿼리 실행의 리턴 타입

select 쿼리는 당연하게도 조회 결과 데이터가 있다. 이 조회된 결과 데이터를 어떤 자료형을 통해 자바로 가져올 지 결정하는 것이 리턴 타입이다. 아래 예시들을 보겠다.

```
SELECT BOARD_NUM
FROM basic_board
WHERE BOARD_NUM = 1;
```

→

BOARD_NUM
1

위 쿼리의 실행결과 조회 결과를 어떤 자료형에 담아오면 될까? 정답은 int다.

그럼 쿼리 실행을 위해 만든 메서드의 리턴 타입은 int가 된다.

```
SELECT BOARD_NUM
FROM basic_board;
```

→

BOARD_NUM
1
2
3

위 쿼리 실행을 위해 만든 메서드의 리턴 타입은 List<Integer>가 된다.

```
SELECT BOARD_TITLE
FROM basic_board
WHERE BOARD_NUM = 1;
```

→

BOARD_TITLE
첫번째

위는 쿼리 실행 메서드의 리턴 타입이 String이다.

03. 개발 흐름 상세 | 04. 작성한 쿼리를 실행할 메서드 정의-3

```
SELECT BOARD_TITLE
FROM basic_board
WHERE BOARD_NUM >= 1;
```

→

BOARD_TITLE
첫번째
두번째
345

위 쿼리를 실행하는 메서드의 리턴 타입은 List<String>이다.

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board
WHERE BOARD_NUM = 1;
```

→

BOARD_NUM	BOARD_TITLE	READ_CNT
1	첫번째	0

위 쿼리를 실행하기 위한 메서드의 리턴 타입은 Board 이다.

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board;
```

→

BOARD_NUM	BOARD_TITLE	READ_CNT
1	첫번째	0
2	두번째	0
3	345	0

위 쿼리를 실행하기 위한 메서드의 리턴 타입은 List<BoardVO> 이다.

위의 예시들처럼 select 쿼리의 조회 결과를 가져오기 위해 return 타입을 지정할 때는 다양한 방식이 존재한다. 위의 예시 정도면 기초적인 쿼리에 대한 리턴 타입을 대부분 작성할 수 있을 것이다. 다시 말하지만 리턴 타입이 이해가 된다면 여러 가지 리턴 타입으로 데이터를 받아보길 바란다.

2. 메서드의 매개변수 결정하기

쿼리 실행을 위해 정의한 메서드의 매개변수 역할은 쿼리 실행 시 빈 값을 채우는 기능이다. 그렇기 때문에 매개변수를 정의하기 위해서 우선 실행할 쿼리문을 작성할 수 있어야 한다. 아래 예시들을 보며 매개변수를 어떻게 정의할지 알아보겠다.

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board;
```

위의 쿼리는 게시글 목록 조회를 위해 작성한 쿼리이다. 해당 쿼리는 채워져야 하는 값(빈 값)이 존재하지 않는다. 따라서 위 쿼리를 실행하기 위해 만든 메서드의 매개변수는 없다.

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board
WHERE BOARD_NUM = #{boardNum};
```

위의 쿼리는 넘어오는 baordNum값을 갖는 게시글 정보를 조회하는 쿼리이다.
이 쿼리에서 우리가 채워야 하는 데이터는 1개이다. 이렇게 빈 값이 하나 일 때는 받아야 하는 데이터가 정수인지 문자열인지에 따라 매개변수가 달라진다. 위의 쿼리문은 쿼리 실행을 위해 정수 데이터가 1개 필요하다.(필요한 데이터가 정수인지 문자인지 판단은 테이블의 어떤 컬럼의 값을 채워야하는지 보고 판단할 수 있다. 위 쿼리에서 우리는 BOARD_NUM 컬럼의 값이 필요하고 이 컬럼은 INT형 컬럼이기 때문에 정수 데이터가 필요하다 판단할 수 있다)

03. 개발 흐름 상세 | 04. 작성한 쿼리를 실행할 메서드 정의-4

이렇게 채워야 하는 데이터가 정수 데이터 하나 일 때는 매개변수로 정수형 데이터를 선언하면 된다.

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board
WHERE WRITER = #{writer};
```

위 쿼리는 전달받은 작성자가 쓴 게시글 정보를 조회하는 쿼리이다. 이 쿼리는 채워야 할 데이터가 문자열 데이터 하나이다. 따라서, 해당 쿼리의 실행 메서드의 매개변수는 문자열 데이터로 선언한다.

이제 남은 것은 아래 쿼리와 같이 채워야하는 값이 여러 개인 경우이다.

```
<insert id="insertBoard">
    INSERT INTO BASIC_BOARD (
        BOARD_NUM
        , BOARD_TITLE
        , BOARD_CONTENT
        , WRITER
        , CREATE_DATE
    ) VALUES (
        #{boardNum}
        , #{boardTitle}
        , #{boardContent}
        , #{writer}
        , #{createDate}
    )
</insert>
```

이렇게 채워야하는 값이 2개 이상일 때는 매개변수로 채워야하는 모든 데이터를 전달할 수 있는 class 객체를 전달해야 한다. 좌측의 쿼리에서 보듯이 해당 쿼리는 5개의 채울 데이터가 필요하다. boardNum, boardTitle, boardContent, writer, createDate 등의 5개를 모두 저장하고 전달할 수 있는 클래스는 우리는 VO클래스를 만들어놨을 것이다.(예시에서는 BoardVO 클래스). 그렇기 때문에 좌측의 쿼리와 같이 여러 빈 값을 채우기 위해서는 VO클래스 객체를 매개변수로 선언하면 된다.

이렇게 쿼리 실행을 위해 정의한 메서드의 매개변수 및 리턴 타입을 어떻게 결정하는지 알아봤다. 아래는 쿼리에 대한 메소드를 정의한 예시이다.

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board;
```

List<boardVO> select1();

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board
WHERE BOARD_NUM = #{boardNum};
```

BoardVO select2(int boardNum);

```
SELECT BOARD_NUM
    , BOARD_TITLE
    , READ_CNT
FROM basic_board
WHERE WRITER = #{writer};
```

BoardVO select3(Strnig writer);

```
SELECT BOARD_TITLE
FROM basic_board
WHERE BOARD_NUM >= 1;
```

List<String> select4();

03. 개발 흐름 상세 | 04. 작성한 쿼리를 실행할 메서드 정의-5

```xml
<insert id="insertBoard">
    INSERT INTO BASIC_BOARD (
        BOARD_NUM
        , BOARD_TITLE
        , BOARD_CONTENT
        , WRITER
        , CREATE_DATE
    ) VALUES (
        #{boardNum}
        , #{boardTitle}
        , #{boardContent}
        , #{writer}
        , #{createDate}
    )
</insert>
```

int insert1(BoardVO boardVO); , void insert1(BoardVO boardVO);

```xml
UPDATE basic_board
SET
    WRITER = #{writer}
    , BOARD_CONTENT = #{boardContent}
WHERE BOARD_NUM = 1;
```

int update1(BoardVO boardVO);, void update1(BoardVO boardVO);

```xml
DELETE FROM basic_board
WHERE BOARD_NUM = #{boardNum};
```

int delete1(int boardNum);, void delete1(int boardNum);

03. 개발 흐름 상세 | 05. interface에서 정의한 메서드 구현-1

1. SqlSessionTemplate 객체를 활용한 쿼리 실행 메서드 호출

ServiceImpl클래스에는 Mybatis의 쿼리 실행 기능을 제공하는 SqlSessionTemplate 클래스에 대한 객체가 존재한다.

```
@Service("boardService")
public class BoardServiceImpl implements BoardService{
    @Autowired
    private SqlSessionTemplate sqlSession;
```

예시) BoardServiceImpl 클래스에서 정의된 SqlSessionTemplate 클래스의 객체

이렇게 생성된 SqlSessionTemplate 클래스에 대한 객체 sqlSession으로 Mabatis의 쿼리 실행 메서드를 호출할 수 있다. sqlSession 객체에서 쿼리 실행을 위해 호출할 메소드는 크게 아래와 같다.

1) UPDATE 쿼리 실행 메서드 : sqlSesssion.update();
2) INSERT 쿼리 실행 메서드 : sqlSesssion.insert();
3) DELETE 쿼리 실행 메서드 : sqlSesssion.delete();
4) SELECT 쿼리 실행 메서드 : sqlSesssion.selectOne();
   sqlSesssion.selectList();

위에서 언급한 쿼리 실행을 위한 메서드 5개 모두 매개변수는 동일하게 두개를 갖는다.

첫번째 매개변수는

위에서 언급한 쿼리 실행을 위한 메서드 5개 모두 매개변수는 동일하게 두개를 갖는다.

첫번째 매개변수는 실행할 쿼리가 작성된 “mapper.xml 파일의 namespace명.쿼리ID”이다.

```
<mapper namespace="boardMapper">
        <select id="selectBoardList" resultMap="board">
        SELECT BOARD_NUM
             , BOARD_TITLE
             , WRITER
             , CREATE_DATE
             , READ_CNT
        FROM BASIC_BOARD
    </select>
```

위의 같이 mapper.xml에 선언된 쿼리를 실행하기 위해선

첫번째 매개변수로 “boardMapper.selectBoardList” 문자열 값이 들어간다.

두번째 매개변수도 5개의 메서드가 동일하며 쿼리 실행 시 넘겨줘야하는 빈 값이 들어간 데이터를 두번째 매개변수로 전달한다. 이는 메서드 구현 직전 interface에서 정의한 쿼리를 실행하기 위해 만든 메서드의 매개변수를 그대로 전달하면 된다.

03. 개발 흐름 상세 | 05. interface에서 정의한 메서드 구현-2

두번째 매개변수도 5개의 메서드가 동일하며 쿼리 실행 시 빈 값이 채울 데이터가 있는 데이터를 두번째 매개변수로 전달한다. 이는 메서드 구현 직전 interface에서 정의한 쿼리를 실행하기 위해 만든 메서드의 매개변수를 그대로 전달하면 된다. 예를 보자.

```
//게시글 등록
void insertBoard(BoardVO boardVO);
```

BoardService interface에서 정의한 insert 쿼리 실행을 위한 메서드 정의

```
public void insertBoard(BoardVO boardVO) {
    sqlSession.insert("boardMapper.insertBoard", boardVO);
}
```

interface에서 정의한 insertBoard 메서드를 ServiceImpl 클래스에서 구현한 코드

위의 코드에서 알 수 있는 sqlSession 클래스의 메소드의 두번째 매개변수는 인터페이스에서 정의한 메서드의 매개변수를 그대로 전달하면 된다.

여기까지 01~05번까지 순차적으로 어떻게 코드를 작성하는지 가이드를 작성하였다.

06. ServiceImpl 클래스에서 정의한 메서드를 호출

07. 필요에 따라 메서드 호출 결과 데이터를 html로 전달

08. html 작성

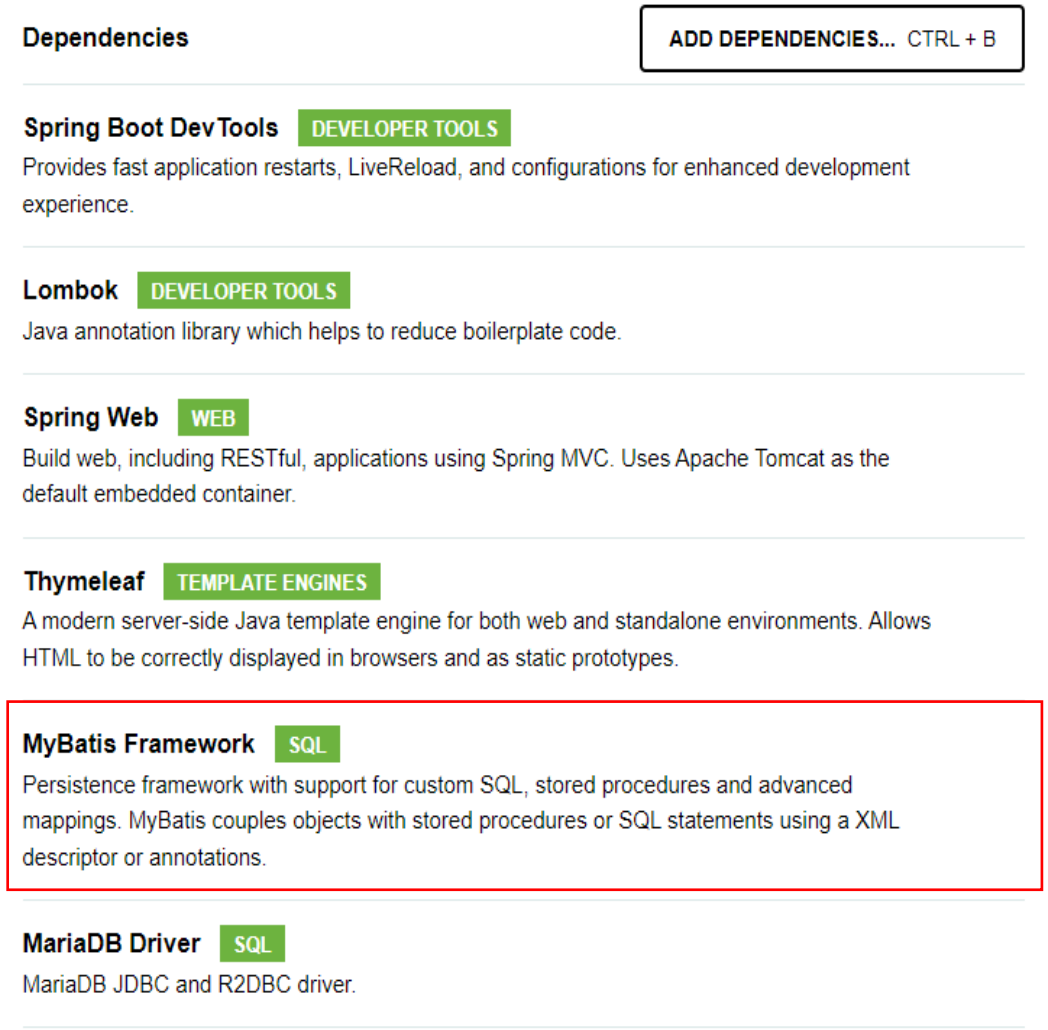
등의 내용은 따로 설명하지 않겠다. 이해가 안되는 부분은 수업 시간이 질문을 부탁한다.

부록 1

Spring boot 프로젝트에 Mybatis 연동하기

부록. Spring boot 프로젝트에 Mybatis 연동하기 – 1

1. spring.io 싸이트에서 프로젝트 생성 시 Mabatis Framework 추가



2. 프로젝트의 application.properties 파일에 DB 접속 정보 및 쿼리를 작성하는 mapper 파일의 위치를 지정

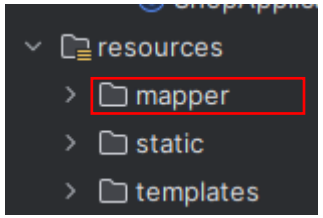
```
#데이터베이스 접속 정보
spring.datasource.driverClassName=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://localhost:3306/study_db
spring.datasource.username=study_user
spring.datasource.password=mariadb

#쿼리가 들어있는 mapper 파일 위치 지정
mybatis.mapper-locations=classpath:mapper/*.xml
```

데이터베이스명

계정의 아이디와 비번

mapper 위치



classpath 는 resources 파일을 지칭함

resources 폴더 밑 mapper 라는 폴더 안의 모든 xml 파일을 쿼리가 작성된 mapper 파일로 인식하는 설정

부록. 콘솔에 실행하는 쿼리 로그가 보일 수 있도록 설정

1. 아래와 같이 DB 접속 정보 변경

```
#데이터베이스 접속 정보
#spring.datasource.driverClassName=org.mariadb.jdbc.Driver
#spring.datasource.url=jdbc:mariadb://localhost:3306/study_db
spring.datasource.driver-class-name=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
spring.datasource.url=jdbc:log4jdbc:mariadb://localhost:3306/study_db
spring.datasource.username=study_user
spring.datasource.password=mariadb
```

기존 내용은 삭제
또는 주석처리

새로 추가한 부분

2. 함께 제공한 log4jdbc.log4j2.properties 파일과 logback.xml 파일을 resources 폴더에 붙여넣기 하여 사용

3. 혹시 몰라 mapper.xml 파일도 함께 제공하니 xml 파일은 resources/mapper 폴더 안에 붙여넣기하여 사용