mybatis resultMap

2020-10-17

이승진

**학습목표**

mybatis resultmap 기능을 학습한다

lombok을 사용한다

**목차**

[1. 개요 2](#_Toc84755515)

[1) mybatis resultMap 기능 2](#_Toc84755516)

[2) mybatis mapper 구현 방법 2](#_Toc84755517)

[3) 수강신청(register) 테이블 구조 3](#_Toc84755518)

[4) SQL 명령 4](#_Toc84755519)

[2. 수강신청 목록 구현 #1 5](#_Toc84755520)

[1) 프로젝트 생성 5](#_Toc84755521)

[2) Register.java 구현 8](#_Toc84755522)

[3) RegisterMapper.java 구현 10](#_Toc84755523)

[4) RegisterController.java 구현 11](#_Toc84755524)

[5) commons.css 구현 12](#_Toc84755525)

[6) register/list.jsp 구현 13](#_Toc84755526)

[7) 실행 14](#_Toc84755527)

[8) 평가 14](#_Toc84755528)

[3. lombok 15](#_Toc84755529)

[1) lombok이란? 15](#_Toc84755530)

[2) lombok 설치 16](#_Toc84755531)

[4. 수강신청 목록 구현 #2 19](#_Toc84755532)

[1) 프로젝트 생성 19](#_Toc84755533)

[2) Student DTO 클래스 구현 21](#_Toc84755534)

[3) Course DTO 클래스 구현 22](#_Toc84755535)

[4) Register DTO 클래스 구현 22](#_Toc84755536)

[5) RegisterMapper.xml 구현 23](#_Toc84755537)

[6) RegisterMapper.java 구현 25](#_Toc84755538)

[7) RegisterController.java 구현 26](#_Toc84755539)

[8) commons.css 구현 27](#_Toc84755540)

[9) register/list.jsp 28](#_Toc84755541)

[10) 실행 29](#_Toc84755542)

[11) 평가 29](#_Toc84755543)

[5. 연습문제 30](#_Toc84755544)

[1) student/list 학생 목록 구현 30](#_Toc84755545)

# 개요

## mybatis resultMap 기능

mybatis는 ORM 기술 중의 하나이다.

ORM 이란 객체지향 언어의 객체와 관계형 데이터를 서로 변환해 준다는 것이다.

DB 조회 결과를 복잡한 객체 구조로 변환해야 할 때 mybatis의 resultMap 기능을 사용한다.

이 기능은 복잡한 보고서(report)를 출력해야 할 때, 특히 유용하다.

여러 테이블의 조인 결과를 여러 자바 객체에 담을 때 resultMap 기능이 유용하다.

## mybatis mapper 구현 방법

### annotation으로 구현하기

지난 시간에 구현한 StudentMapper는 annotation을 사용하여 mapper를 구현하였다.

간단한 SQL을 구현할 때, annotation을 이용하여 구현하는 것이 편하다.

### mapper XML 파일에 구현하기

복잡한 SQL을 구현하거나, mybatis resultMap 기능을 구현하려면

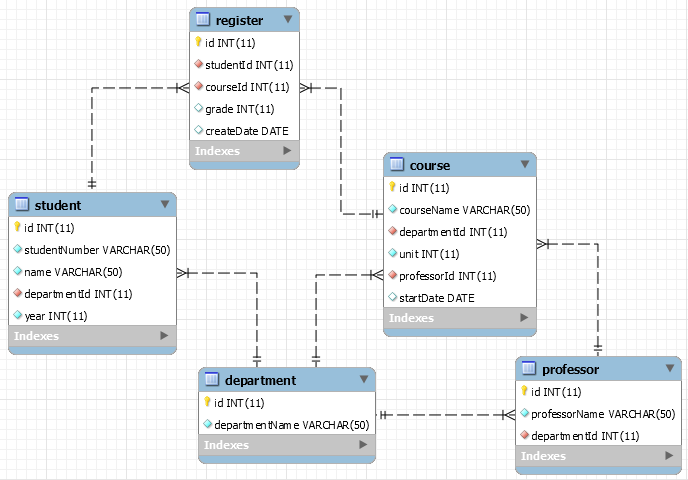
mapper XML 파일에 구현하는 것이 편하다.

위 두 방법을 혼합해도 된다.

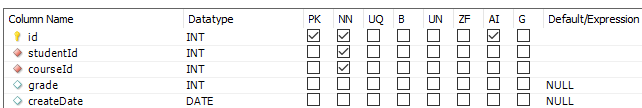
간단한 SQL의 메소드는 annotation으로 구현하고

복잡한 SQL의 메소드는 mapper XML 파일에 구현할 수 있다.

## 수강신청(register) 테이블 구조



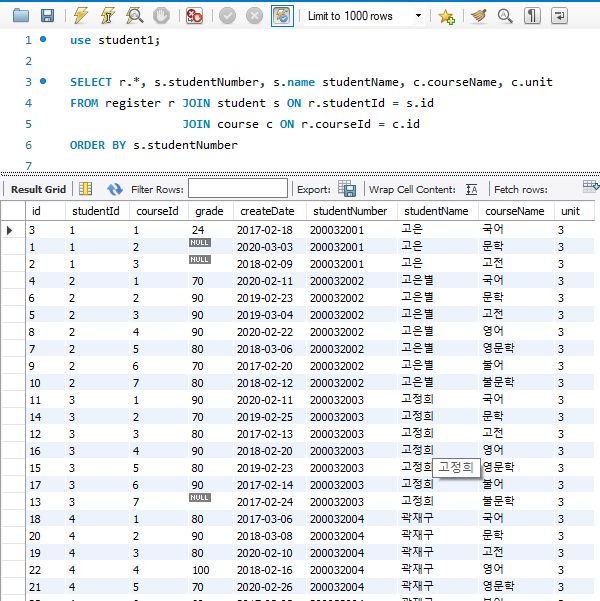
register 테이블에 수강신청 데이터가 들어있다.



|  |  |
| --- | --- |
| 필드명 | 설명 |
| id | 자동으로 일련번호가 부여되는 (auto increment) 기본키(primary key) |
| studentId | 학생 id 외래키 |
| courseId | 강좌 id 외래키 |
| grade | 점수 |
| createDate | 수강신청 등록일 |

## SQL 명령

|  |
| --- |
| SELECT r.\*, s.studentNumber, s.name studentName, c.courseName, c.unit  FROM register r JOIN student s ON r.studentId = s.id  JOIN course c ON r.courseId = c.id  ORDER BY s.studentNumber |



### 조회 결과 컬럼

register 테이블 (id, studentId, courseId, grade, createDate)

student 테이블 (studentNumber, studentName)

course 테이블 (courseName, unit)

student 테이블의 name 필드가 조회 결과에서는 studentName 컬럼으로 출력된 것에 주의하자.

컬럼명을 구별하기 쉽도록 studentName 컬럼명으로 바꿨다.

위 SQL 명령을 mybatis mapper 메소드에 구현하자.

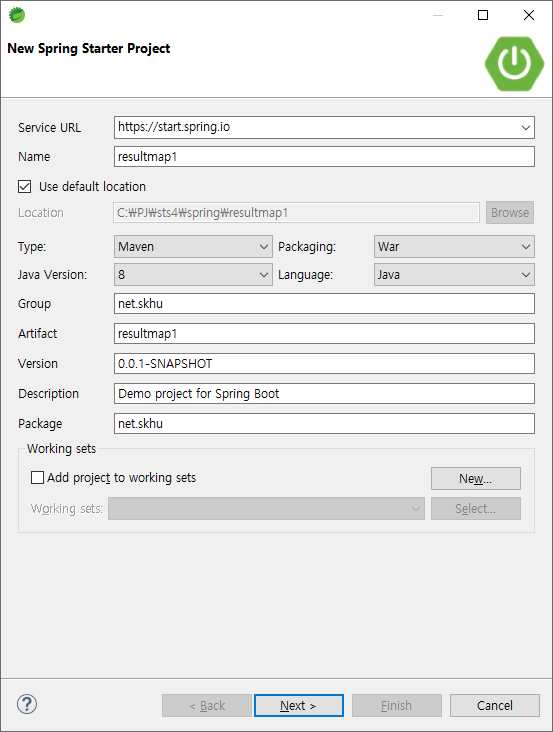
위 조회 결과 데이터를 채울 DTO 클래스를 구현하자.

# 수강신청 목록 구현 #1

전체 수강신청 목록을 조회하는 기능을 mybatis resultMap 기능을 사용하지 않고 구현하자.

## 프로젝트 생성

File - New - Spring Starter Project



|  |  |
| --- | --- |
| Name | resultmap1 |
| Package | net.skhu |

Graphical user interface, text, application

Description automatically generated

Developer Tools 항목 아래에서

Spring Boot DevTools 체크

SQL 항목 아래에서

MyBatis Framework 체크

MySQL Driver 체크

Web 항목 아래에서

Spring Web 체크

### pom.xml 수정

|  |
| --- |
| <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency> |

위 항목들을 maven dependency 라고 부른다.

pom.xml 파일에서 <dependency> 태그들이 있는 곳에 추가하자.

pom.xml은 maven 설정 파일이다.

### appliation.properties 수정

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/student1?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=skhuA+4.5  server.port=8088 |

application.properties는 spring Boot 설정 파일이다.

## Register.java 구현

### src/main/java/net/skhu/dto/Register.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | package net.skhu.dto;  import java.util.Date;  public class Register {  int id;  int studentId;  String StudentNumber;  String studentName;  int courseId;  String courseName;  int unit;  int grade;  Date createDate;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public int getStudentId() {  return studentId;  }  public void setStudentId(int studentId) {  this.studentId = studentId;  }  public String getStudentNumber() {  return StudentNumber;  }  public void setStudentNumber(String studentNumber) {  StudentNumber = studentNumber;  }  public String getStudentName() {  return studentName;  }  public void setStudentName(String studentName) {  this.studentName = studentName;  }  public int getCourseId() {  return courseId;  }  public void setCourseId(int courseId) {  this.courseId = courseId;  }  public String getCourseName() {  return courseName;  }  public void setCourseName(String courseName) {  this.courseName = courseName;  }  public int getUnit() {  return unit;  }  public void setUnit(int unit) {  this.unit = unit;  }  public int getGrade() {  return grade;  }  public void setGrade(int grade) {  this.grade = grade;  }  public Date getCreateDate() {  return createDate;  }  public void setCreateDate(Date createDate) {  this.createDate = createDate;  }  } |

## RegisterMapper.java 구현

### src/main/java/net/skhu/mapper/RegisterMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Select;  import net.skhu.dto.Register;  @Mapper  public interface RegisterMapper {  @Select("SELECT r.\*, s.studentNumber, s.name studentName, c.courseName, c.unit " +  " FROM register r JOIN student s ON r.studentId = s.id " +  " JOIN course c ON r.courseId = c.id " +  " ORDER BY s.studentNumber ")  List<Register> findAll();  } |

## RegisterController.java 구현

### src/main/java/net/skhu/controller/RegisterController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import net.skhu.mapper.RegisterMapper;  @Controller  @RequestMapping("register")  public class RegisterController {  @Autowired RegisterMapper registerMapper;  @RequestMapping("list")  public String list(Model model) {  model.addAttribute("registers", registerMapper.findAll());  return "register/list";  }  } |

## commons.css 구현

### src/main/resources/static/common.css

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | div.container { width: 800px; margin: 10px auto; font-size: 10pt; }  .btn { padding: 0.4em 1em; border: 1px solid gray;  border-radius: 0.5em; background: linear-gradient(#fff, #ddd);  text-decoration: none; color: black;  display: inline-block; }  .btn:active {  -ms-transform: translateY(2px);  -webkit-transform: translateY(2px);  transform: translateY(2px);  background: #ccc; }  table.list { border-collapse: collapse; width: 100%; }  table.list td { padding: 4px; border: 1px solid gray; }  table.list th { padding: 4px; border: 1px solid gray; background-color: #eee; }  input { padding: 4px; }  select { padding: 4px; }  tr[data-url]:hover { cursor: pointer; background-color: #ffe; } |

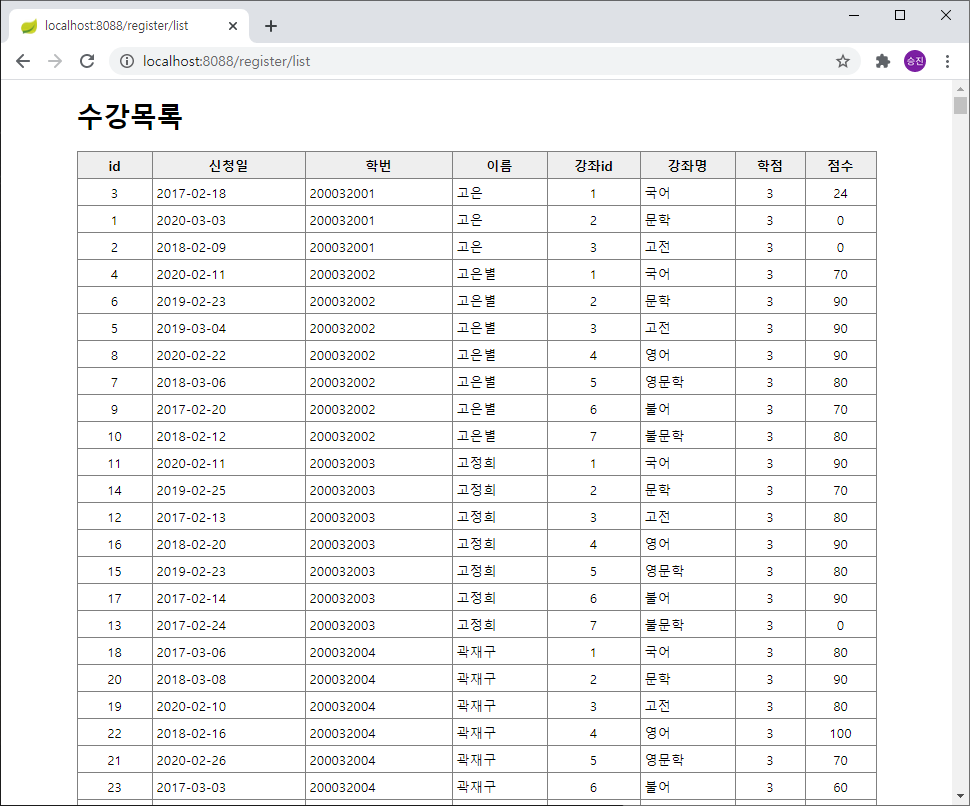
## register/list.jsp 구현

### src/main/webapp/WEB-INF/views/register/list.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  td:nth-child(1), td:nth-child(5), td:nth-child(7), td:nth-child(8) { text-align: center; }  </style>  </head>  <body>  <div class="container">  <h1>수강목록</h1>  <table class="list">  <thead>  <tr>  <th>id</th>  <th>신청일</th>  <th>학번</th>  <th>이름</th>  <th>강좌id</th>  <th>강좌명</th>  <th>학점</th>  <th>점수</th>  </tr>  </thead>  <tbody>  <c:forEach var="register" items="${ registers }">  <tr>  <td>${ register.id }</td>  <td><fmt:formatDate pattern="yyyy-MM-dd" value="${ register.createDate }" /></td>  <td>${ register.studentNumber }</td>  <td>${ register.studentName }</td>  <td>${ register.courseId }</td>  <td>${ register.courseName }</td>  <td>${ register.unit }</td>  <td>${ register.grade }</td>  </tr>  </c:forEach>  </tbody>  </table>  </div>  </body>  </html> |

## 실행

http://localhost:8088/register/list



## 평가

위 구현 방법의 단점은 Register 클래스의 구현이다.

Register 클래스의 속성은 다음과 같다.

register 테이블 (id, studentId, courseId, grade, createDate)

student 테이블 (studentNumber, studentName)

course 테이블 (courseName, unit)

register 테이블의 필드에 해당하는 속성이 5개이고,

다른 테이블의 필드에 해당하는 속성이 4개이다.

나중에 프로젝트 할 때, 이런 방식으로 DTO 클래스를 구현하면,

DTO 클래스에 속성을 몇 십개 정도 만들게 될 수도 있다.

너무 복작하다.

# lombok

## lombok이란?

lombok 은 getter, setter 메소드를 자동으로 생성해 주는 도구이다.

lombok 을 사용하면, getter, setter 메소드를 구현하지 않아도 되니 편하다.

equals 메소드, hashCode 메소드, toString 메소드도 자동으로 생성해 준다.

클래스에 @Data 어노테이션을 붙여주기만 하면 된다.

getter, setter, equals, hashCode, toString 메소드들이 자동 생성된다.

### 바이트코드 파일에 생성됨 (\*.class)

그런데 소스 코드 파일이 아니고, 컴파일되어 생성되는 \*.class 파일에,

위 메소드들이 자동 생성된다.

### @Data 어노테이션

이 어노테이션을 클래스 앞에 붙이면,

getter, setter, equals, hashCode, toString 메소드가 자동 구현된다.

예:

|  |
| --- |
| @Data  public class Student {  int id;  String name;  } |

- 모든 멤버 변수에 대해서 getter setter 가 자동으로 구현된다.

- 모든 멤버 변수 값이 같은지 비교하는 equals 메소드가 자동으로 구현된다.

- 모든 멤버 변수 값을 바탕으로 해시 값을 계산하는 hashCode 메소드가 자동으로 구현된다.

- 모든 멤버 변수 값을 문자열로 표현하는 toString 메소드가 자동으로 구현된다.

만약 자동으로 구현되는 메소드에서 제외되어야 할 멤버 변수가 있다면 다음과 같이 구현한다.

예:

|  |
| --- |
| @Data  @EqualsAndHashCode(exclude="temp")  @ToString(exclude="temp")  public class Student {  int id;  String name;  int temp;  } |

equals, hashCode, toString 메소드가 자동으로 구현될 때, temp 멤버 변수는 무시된다.

예:

|  |
| --- |
| @Data  @EqualsAndHashCode(exclude={"temp1","temp2"})  @ToString(exclude={"temp1","temp2"})  public class Student {  int id;  String name;  int temp1;  String temp2;  } |

equals, hashCode, toString 메소드가 자동으로 구현될 때, temp1, temp2 멤버 변수는 무시된다.

## lombok 설치

### 다운로드

https://projectlombok.org/download.html

위 웹페이지에서 lombok.jar 다운로드

Logo

Description automatically generated

### lombok.jar 실행

먼저 Spring tool Suite를 종료하고 설치해야 한다.

명령 프롬프트 창에서, 다운로드된 lombok.jar 파일이 있는 디렉토리로 이동하여, 다음 명령을 실행한다.

|  |
| --- |
| java -jar lombok.jar |

Text

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated

만약 위 대화 상자가 나타나면 '확인' 버튼을 클릭하여 닫는다.

위 대화 상자가 나타나지 않을 수도 있다.

Graphical user interface, text, application, email

Description automatically generated

만약 eclipse.exe 파일이 목록에 보인다면, 체크를 끈다.

SpringToolSuite4.exe 파일이 목록에 보이지 않는다면

'Specify location...' 버튼을 클릭하여 SpringToolSuite4.exe 파일을 찾는다.

Graphical user interface, text, application, email

Description automatically generated

SpringToolSuite4.exe (Spring tool suite) 실행 파일을 찾아서 선택한다.

Graphical user interface, text, application, email

Description automatically generated

SpringToolSuite4.exe 파일이 목록에 보인다.

SpringToolSuite4.exe 파일을 체크하고, 'Install / Update' 버튼을 클릭해서 설치하자.

Graphical user interface, text, application, email

Description automatically generated

Install successful 문구를 확인하고,

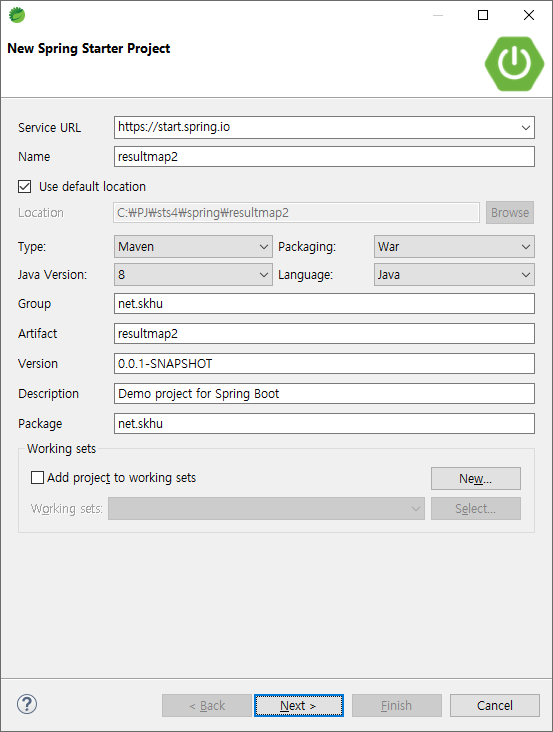
Quit Installer 클릭

# 수강신청 목록 구현 #2

전체 수강신청 목록을 조회하는 기능을 mybatis resultMap 기능으로 구현하자.

## 프로젝트 생성

File - New - Spring Starter Project



|  |  |
| --- | --- |
| Name | resultmap2 |
| Package | net.skhu |

Graphical user interface, text, application

Description automatically generated

Developer Tools 항목 아래에서

Spring Boot DevTools 체크

Lombok 체크

SQL 항목 아래에서

MyBatis Framework 체크

MySQL Driver 체크

Web 항목 아래에서

Spring Web 체크

### pom.xml 수정

|  |
| --- |
| <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency> |

위 항목들을 maven dependency 라고 부른다.

pom.xml 파일에서 <dependency> 태그들이 있는 곳에 추가하자.

pom.xml은 maven 설정 파일이다.

### appliation.properties 수정

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/student1?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=skhuA+4.5  server.port=8088 |

application.properties는 spring Boot 설정 파일이다.

## Student DTO 클래스 구현

### src/main/java/net/skhu/dto/Student.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | package net.skhu.dto;  import lombok.Data;  @Data  public class Student {  int id;  String studentNumber;  String name;  int departmentId;  int year;  } |

student 테이블의 필드에 해당하는 속성만 구현된 평범한 DTO 클래스이다.

@Data 어노테이션을 클래스 앞에 붙이면,

lombok에 의해서 setter, setter, equals, hashCode, toString 메소드가 자동 구현된다.

## Course DTO 클래스 구현

### src/main/java/net/skhu/dto/Course.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package net.skhu.dto;  import java.util.Date;  import lombok.Data;  @Data  public class Course {  int id;  String courseName;  int departmentId;  int unit;  int professorId;  Date startDate;  } |

course 테이블의 필드에 해당하는 속성만 구현된 평범한 DTO 클래스이다.

## Register DTO 클래스 구현

### src/main/java/net/skhu/dto/Register.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package net.skhu.dto;  import java.util.Date;  import lombok.Data;  @Data  public class Register {  int id;  int studentId;  int courseId;  int grade;  Date createDate;  Student student;  Course course;  } |

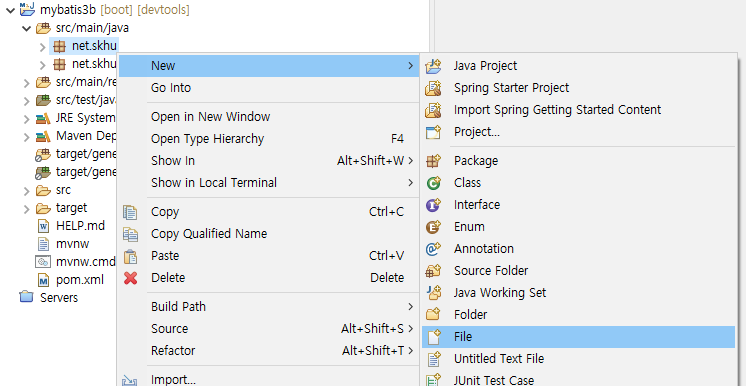
수강신청 데이터를 조회할 때,

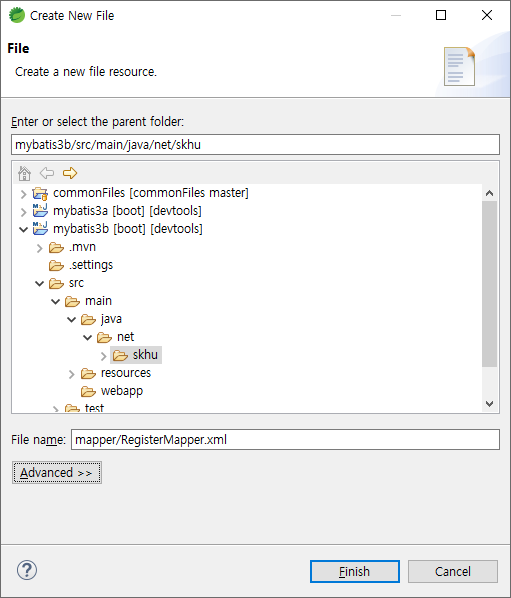
학생(student) 정보와 강좌(course) 정보를 포함해서 조회해야 하기 때문에

student 속성과 course 속성을 구현했다

이 속성의 타입은 각각 Student, Course 클래스 타입이다.

## RegisterMapper.xml 구현





### src/main/java/net/skhu/mapper/RegisterMapper.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <?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="net.skhu.mapper.RegisterMapper">  <resultMap id="RegisterWithStudentAndCourse" type="net.skhu.dto.Register">  <id property="id" column="id" />  <result property="studentId" column="studentId" />  <result property="courseId" column="courseId" />  <result property="grade" column="grade" />  <result property="createDate" column="createDate" />  <association property="student" javaType="net.skhu.dto.Student">  <id property="id" column="studentId" />  <result property="studentNumber" column="studentNumber" />  <result property="name" column="studentName" />  </association>  <association property="course" javaType="net.skhu.dto.Course">  <id property="id" column="courseId" />  <result property="courseName" column="courseName" />  <result property="unit" column="unit" />  </association>  </resultMap>  </mapper> |

<mapper namespace="net.skhu.mapper.RegisterMapper">

RegisterMapper에 대한 SQL 명령이나, resultMap 등을 정의하기 mapper 태그이다.

패키지까지 포함해서 mapper 인터페이스 이름을 적어야 한다. 오타에 주의하자.

<resultMap id="RegisterWithStudentCourse" type="net.skhu.dto.Register">

조회 결과를 Register 객체에 채우는 방법을 정의한다.

SQL 조회 결과를 Java 객체 구조에 채우는 방법을 정의하는 것이 resultMap 이다.

<id property="id" column="id" />

조회 결과의 id 컬럼은 Register 클래스의 id 속성(property)에 채운다.

이 컬럼이 테이블의 기본키(primary key)이기 때문에 <id> 태그를 사용하였다.

<result property="studentId" column="studentId" />

조회 결과의 studentId 컬럼를 Register 클래스의 studentId 속성(property)에 채운다.

이 컬럼이 테이블의 기본키가 아니기 때문에 <result> 태그를 사용하였다.

<association property="student" javaType="net.skhu.dto.Student">

Register 클래스의 student 속성에 Student 객체를 대입한다.

<id property="id" column="studentId" />

조회 결과의 studentId 컬럼은 Student 클래스의 id 속성에 채운다.

이 컬럼이 student 테이블의 기본키이기 때문에 <id> 태그를 사용하였다.

<result property="studentNumber" column="studentNumber"/>

조회 결과의 studentNumber 컬럼은 Student 클래스의 studentNumber 속성(property)에 채운다.

이 컬럼이 student 테이블의 기본키가 아니기 때문에 <result> 태그를 사용하였다.

<result property="name" column="studentName"/>

조회 결과의 studentName 컬럼은 Student 클래스의 name 속성에 채운다.

조회 결과의 컬럼명과 Java 객체의 속성명이 달라도 된다.

## RegisterMapper.java 구현

### src/main/java/net/skhu/mapper/RegisterMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.ResultMap;  import org.apache.ibatis.annotations.Select;  import net.skhu.dto.Register;  @Mapper  public interface RegisterMapper {  @ResultMap("RegisterWithStudentAndCourse")  @Select("SELECT r.\*, s.studentNumber, s.name studentName, c.courseName, c.unit " +  " FROM register r JOIN student s ON r.studentId = s.id " +  " JOIN course c ON r.courseId = c.id " +  " ORDER BY s.studentNumber ")  List<Register> findAll();  } |

@ResultMap("RegisterWithStudentAndCourse")

RegisterMapper.xml 파일의 id="RegisterWithStudentAndCourse" resultMap 방법으로

조회 결과를 Java 객체들에 채워서 리턴한다.

mapper xml 파일과 mapper java 파일에서 노란색과 파란색으로 칠한 부분이 각각 일치해야 한다.

오타를 조심하자.

## RegisterController.java 구현

### src/main/java/net/skhu/controller/RegisterController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import net.skhu.mapper.RegisterMapper;  @Controller  @RequestMapping("register")  public class RegisterController {  @Autowired RegisterMapper registerMapper;  @RequestMapping("list")  public String list(Model model) {  model.addAttribute("registers", registerMapper.findAll());  return "register/list";  }  } |

## commons.css 구현

### src/main/resources/static/common.css

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | div.container { width: 800px; margin: 10px auto; font-size: 10pt; }  .btn { padding: 0.4em 1em; border: 1px solid gray;  border-radius: 0.5em; background: linear-gradient(#fff, #ddd);  text-decoration: none; color: black;  display: inline-block; }  .btn:active {  -ms-transform: translateY(2px);  -webkit-transform: translateY(2px);  transform: translateY(2px);  background: #ccc; }  table.list { border-collapse: collapse; width: 100%; }  table.list td { padding: 4px; border: 1px solid gray; }  table.list th { padding: 4px; border: 1px solid gray; background-color: #eee; }  input { padding: 4px; }  select { padding: 4px; }  tr[data-url]:hover { cursor: pointer; background-color: #ffe; } |

## register/list.jsp

### src/main/webapp/WEB-INF/views/register/list.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  td:nth-child(1), td:nth-child(5), td:nth-child(7), td:nth-child(8) { text-align: center; }  </style>  </head>  <body>  <div class="container">  <h1>수강목록</h1>  <table class="list">  <thead>  <tr>  <th>id</th>  <th>신청일</th>  <th>학번</th>  <th>이름</th>  <th>강좌id</th>  <th>강좌명</th>  <th>학점</th>  <th>점수</th>  </tr>  </thead>  <tbody>  <c:forEach var="register" items="${ registers }">  <tr>  <td>${ register.id }</td>  <td><fmt:formatDate pattern="yyyy-MM-dd" value="${ register.createDate }" /></td>  <td>${ register.student.studentNumber }</td>  <td>${ register.student.name }</td>  <td>${ register.course.id }</td>  <td>${ register.course.courseName }</td>  <td>${ register.course.unit }</td>  <td>${ register.grade }</td>  </tr>  </c:forEach>  </tbody>  </table>  </div>  </body>  </html> |

${ register.student.studentNumber }

register 객체에 student 속성이 있다. student 속성은 Student 클래스이다.

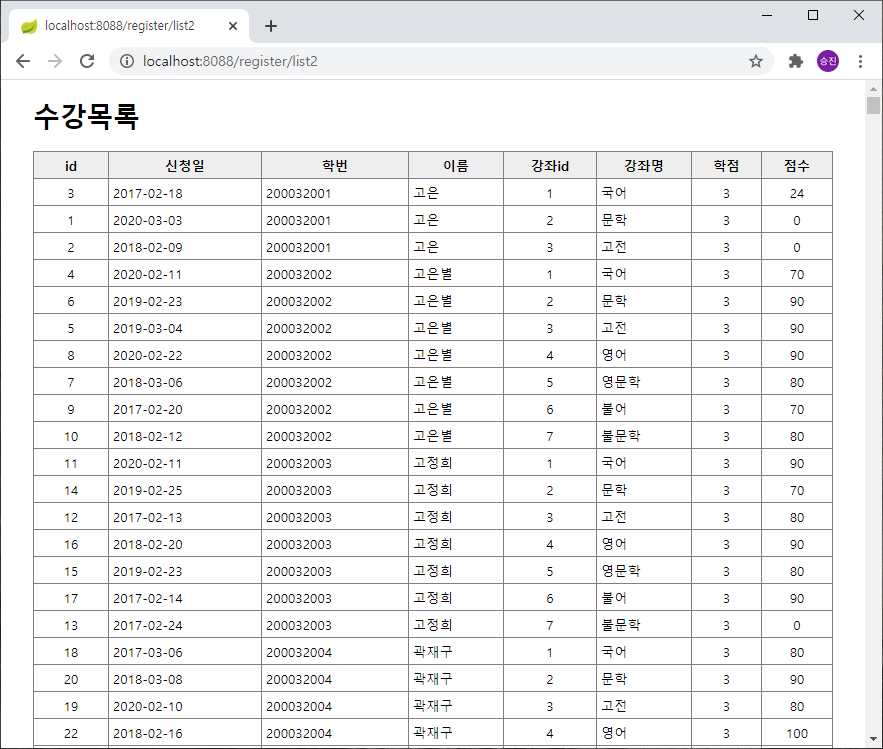
이 student 속성의 studentNumber 속성값을 출력한다.

register.getStudent().getStudentNumber()

위 표현식의 값이 출력된다.

## 실행

http://localhost:8088/register/list2



## 평가

resultMap 구현 방법의 장점은 깔끔한 객체 구조에 조회 결과를 채워서 리턴할 수 있다는 점이다.

Register 클래스의 속성은 다음과 같다.

register 테이블 (id, studentId, courseId, grade, createDate)

Student 객체 속성

Course 속성

student 테이블에 해당하는 컬럼들은 Student 객체에 채우고,

course 테이블에 해당하는 컬럼들은 Course 객체에 채운다.

# 연습문제

## student/list 학생 목록 구현

Student DTO 클래스에 String departmentName 속성 대신 Department department 객체 속성을 만들자.

그리고 학생 목록 조회 결과에서 departmentId, departmentName 컬럼들을

student.department.id, student.department.departmentName 속성에 각각 채우자

## RegisterMapper 수정

### 수정 전

src/main/java/net/skhu/mapper/RegisterMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Select;  import net.skhu.dto.Register;  @Mapper  public interface RegisterMapper {  @Select("SELECT r.\*, s.studentNumber, s.name studentName, c.courseName, c.unit " +  " FROM register r JOIN student s ON r.studentId = s.id " +  " JOIN course c ON r.courseId = c.id " +  " ORDER BY s.studentNumber ")  List<Register> findAll();  } |

### 수정 후

src/main/java/net/skhu/mapper/RegisterMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Select;  import net.skhu.dto.Register;  @Mapper  public interface RegisterMapper {  List<Register> findAll();  } |

src/main/java/net/skhu/mapper/RegisterMapper.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <?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="net.skhu.mapper.RegisterMapper">  <select id="findAll" resultType="net.skhu.dto.Register">  SELECT r.\*, s.studentNumber, s.name studentName, c.courseName, c.unit  FROM register r JOIN student s ON r.studentId = s.id  JOIN course c ON r.courseId = c.id  ORDER BY s.studentNumber  </select>  </mapper> |