Spring form validation #1

2023-09-10

이승진

**학습목표**

spring form validation 구현

모델 클래스, DTO 클래스 개념

**목차**

[1. 배경지식 2](#_Toc145435560)

[1) spring form validation 2](#_Toc145435561)

[2) 모델(model) 클래스 5](#_Toc145435562)

[3) 서비스 클래스 6](#_Toc145435563)

[2. mybatis2 프로젝트 7](#_Toc145435564)

[1) 프로젝트 생성 7](#_Toc145435565)

[2) application.properties 9](#_Toc145435566)

[3) Department DTO 클래스 9](#_Toc145435567)

[4) DepartmentMapper.java 9](#_Toc145435568)

[5) DepartmentService.java 10](#_Toc145435569)

[6) Student DTO 클래스 11](#_Toc145435570)

[7) StudentEdit 모델 클래스 12](#_Toc145435571)

[8) Student Mapper 구현 13](#_Toc145435572)

[9) StudentService 클래스 14](#_Toc145435573)

[10) Student Controller 구현 16](#_Toc145435574)

[11) 정적 컨텐츠 18](#_Toc145435575)

[12) student/list 뷰 구현 19](#_Toc145435576)

[13) student/edit 뷰 구현 20](#_Toc145435577)

[14) 실행 22](#_Toc145435578)

[3. 연습 문제 23](#_Toc145435579)

# 배경지식

## spring form validation

입력 폼에 입력된 내용의 오류를 spring 이 자동으로 검사해 주는 기능이

spring form validation 이다.

### 입력 폼 submit 과정 #1

spring web mvc로 구현한 입력 폼의 submit 과정은 다음과 같다.

(1) 사용자가 웹브라우저에서 입력 폼에 데이터를 입력하고, submit 버튼을 누른다.

|  |
| --- |
| <form method="post">  <input type="text" name="studentNo" />  <input type="text" name="name" />  <button type="submit">저장</button>  </form> |

(2) 서버의 URL 이 요청된다. (http request)

입력 폼에 입력된 데이터도 이 요청에 같이 담겨 전송된다. (request parameter)

|  |
| --- |
| studentNo: "201132055"  name: "홍길동" |

(3) spring web mvc 엔진이 그 요청을 받아서, 요청된 URL과 일치하는 액션 메소드를 찾는다.

|  |
| --- |
| @PostMapping("edit")  public String edit(Model model, Student student) {  . . .  } |

(4) 위 액션 메소드의 파라미터가 객체이기 때문에, spring web mvc 엔진이 아래의 일들이 자동으로 처리한다.

- Student 객체를 생성한다.

Student student = new Student();

- 생성된 객체에 request parameter 데이터를 채운다.

student.setStudentNo("201132055");

student.setName("홍길동");

- 생성된 객체를 모델 객체에 등록한다.

model.addAttribute("student", student);

이때 "student" 이름은 파라미터 변수 이름이 아니고

Student 클래스 이름에서 첫 문자만 소문자로 변경한 것이다.

(5) 액션 메소드를 호출한다.

edit(model, student);

### 입력 폼 submit 과정 #2

spring form validation 기능이 구현되었을 때,

입력 폼의 submit 과정은 다음과 같다.

(1) 사용자가 웹브라우저에서 입력 폼에 데이터를 입력하고, submit 버튼을 누른다.

|  |
| --- |
| <form method="post">  <input type="text" name="studentNo" />  <input type="text" name="name" />  <button type="submit">저장</button>  </form> |

(2) 서버의 URL 이 요청된다. (http request)

입력 폼에 입력된 데이터도 이 요청에 같이 담겨 전송된다. (request parameter)

|  |
| --- |
| studentNo: "201132055"  name: "홍길동" |

(3) spring web mvc 엔진이 그 요청을 받아서, 요청된 URL과 일치하는 액션 메소드를 찾는다.

|  |
| --- |
| @PostMapping("edit")  public String edit(Model model, @Valid Student student, BindingResult bindingResult) {  . . .  } |

spring form validation 기능을 구현하기 위해,

@Valid 어노테이션과 BindingResult bindingResult 객체가 추가되었다.

파라미터 순서도 중요하다. BindingResult bindingResult 파라미터가 뒤쪽에 있어야 한다.

(4) 위 액션 메소드의 파라미터가 객체이기 때문에, spring web mvc 엔진이 아래의 일들이 자동으로 처리한다.

- Student 객체를 생성한다.

Student student = new Student();

- 생성된 객체에 request parameter 데이터를 채운다.

student.setStudentNo("201132055");

student.setName("홍길동");

- 생성된 객체를 모델 객체에 등록한다.

model.addAttribute("student", student);

(5) Student 객체에 채워진 데이터에 문제가 없는지 검사한다.

데이터를 검사할 규칙이, Student 클래스에 등록되어 있어야 한다.

|  |
| --- |
| public class Student {  @NotEmpty  @Size(min=9, max=12)  String studentNo;  @NotEmpty  @Size(min=2)  String name;  } |

spring form validation 기능을 구현하기 위해, @NotEmpty, @Size 어노테이션이 추가되었다.

@NotEmpty 어노테이션은, 이 멤버 변수 값이 꼭 있어야 한다는 뜻이다.

@Size 어노테이션은, 문자열의 최소 길이와 최대 길이를 지정한다.

Student 객체에 채워진 데이터가, 이 어노테이션의 규칙에 부합하는지를,

spring web mvc 엔진이 검사한다.

검사 결과가 BindingResult 객체에 등록된다.

(6) 액션 메소드가 호출된다.

edit(model, student, bindingResult)

## 모델(model) 클래스

### request parameter 데이터를 채우기 위한 객체

앞에서 설명한 입력 폼 submit 과정에서 사용된 Student 클래스처럼,

request parameter 데이터를 채우기 위한 클래스를 모델 클래스라고 부른다.

|  |
| --- |
| @PostMapping("edit")  public String edit(Model model, Student student) {  . . .  } |

### model 객체에 채워져 뷰에 전달되기 위한 객체

|  |
| --- |
| @GetMapping("edit")  public String edit(Model model) {  Student student = new Student();  model.addAttribute("student", student);  return "edit";  } |

위의 코드에서 student 객체는 model 객체에 채워져서

edit 뷰에 전달된다.

이런 용도로 사용되는 클래스를 모델 클래스라고 부른다.

spring form validation 규칙을 설정하기 위한 어노테이션은

모델 클래스에 추가해야 한다. (@NotEmpty, @Size 등)

|  |
| --- |
| public class Student {  @NotEmpty  @Size(min=9, max=12)  String studentNo;  @NotEmpty  @Size(min=2)  String name; |

## 서비스 클래스

컨트롤러 클래스의 역할은 지휘 통제만 하는 것이다.

컨트롤러 클래스에 구체적인 작업을 구현하는 것은 바람직하지 않다.

컨트롤러 클래스는 서비스 클래스만 참조하고 서비스 클래스의 메소드를 호출해야 한다.

구체적인 작업은 서비스 클래스에 구현되어야 한다.

Student 클래스 객체에 대해서 무엇을 검사하거나 계산하거나 변경하는 작업은,

StudentService 클래스에 구현해야 한다.

강의노트에서는 예제 코드를 간결하게 구현하기 위해서,

컨트롤러 클래스의 코드에서 직접 StudentMapper 메소드를 호출하였다.

이 구조는 별로 바람직하지 않다.

StudentController 클래스에서는 StudentService 클래스의 메소드를 호출하고 (사장 -> 부장)

StudentService 클래스에서는 StudentDAO 클래스의 메소드를 호출하고 (부장 -> 과장)

StudentDAO 클래스에서는 StudentMapper 메소드를 호출하는 형태가 바람직하다 (과장 -> 사원)

이 강의에서는 예제를 간결하게 하기 위해서, DAO 클래스를 구현하지 않는다.

DAO 클래스를 구현하는 것은 나중에 회사가서 배워도 된다.

그렇지만 서비스 클래스를 구현하는 것은 학생 때부터 해야 한다.

객체지향적으로 바람직한 구조를 만들기 위한 설계 원칙 중 하나는,

클래스들의 역할을 분명히 구분하는 것이다.

### 컨트롤러의 역할

컨트롤러의 역할은, 이름 그대로 지휘 통제하는 것이다.

지휘 통제 역할만 해야 한다. 구체적인 작업까지 하는 것은 바람직하지 않다.

지휘 통제만 잘 하기도 어렵기 때문이다.

### DAO(Data Access Object) 클래스의 역할

DAO의 역할은, 데이터베이스 테이블에 직업 SELECT/INSERT/UPDATE/DELETE 하는 작업을 하는 것이다.

SELECT/INSERT/UPDATE/DELETE 작업을 간단히 CRUD 작업이라고 부른다.

(Create, Retrieve, Update, Delete)

DAO 클래스 내부에서 mybatis mapper나 JPA repository를 사용해야 한다.

### 서비스 클래스의 역할

DAO 작업을 제외한 나머지 작업들은 서비스 클래스에 구현되어야 한다.

컨트롤러는 서비스 클래스에게 작업을 지시하기만 해야 한다.

서비스 클래스의 메소드에 작업이 구현되어야 한다.



# mybatis2 프로젝트

## 프로젝트 생성

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Name | mybatis2 |
| Java Version | 17 |
| Package | net.skhu |

A screenshot of a computer

Description automatically generated

I/O 그룹 아래에서 Validation 항목을 체크한다.

## application.properties

|  |
| --- |
| spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/student2?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=skhuA+4.5  server.port=8088 |

## Department DTO 클래스

src/main/java/net/skhu/dto/Department.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | package net.skhu.dto;  import lombok.Data;  @Data  public class Department {  int id;  String name;  String shortName;  String phone;  } |

## DepartmentMapper.java

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Select;  import net.skhu.dto.Department;  @Mapper  public interface DepartmentMapper {  @Select("SELECT \* FROM department")  List<Department> findAll();  } |

## DepartmentService.java

src/main/java/net/skhu/service/DepartmentService.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package net.skhu.service;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import net.skhu.dto.Department;  import net.skhu.mapper.DepartmentMapper;  @Service  public class DepartmentService {  @Autowired  public DepartmentMapper departmentMapper;  public List<Department> findAll() {  return departmentMapper.findAll();  }  } |

서비스 클래스에는 @Service 어노테이션을 붙여야 한다.

DepartmentService 클래스는 DepartmentMapper의 메소드를 호출만 할 뿐 별로 하는 일이 없다.

그렇지만 컨트롤러가 DepartmentMapper의 메소드를 직접 호출하는 것은 바람직하지 않기 때문에

DepartmentService 클래스가 필요하다.

이 강의 예제에서 서비스 클래스를 구현하지 않는 경우가 많지만,

실무에서는 별로 구현할 코드가 없더라도 서비스 클래스를 구현하는 것이 좋다.

A black background with a black square

Description automatically generated with medium confidence

## Student DTO 클래스

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package net.skhu.dto;  import lombok.Data;  @Data  public class Student {  int id;  String studentNo;  String name;  int departmentId;  String phone;  String sex;  String email;  String departmentName;  } |

## StudentEdit 모델 클래스

src/main/java/net/skhu/model/StudentEdit.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 | package net.skhu.model;  import jakarta.validation.constraints.Email;  import jakarta.validation.constraints.NotBlank;  import jakarta.validation.constraints.NotEmpty;  import jakarta.validation.constraints.Pattern;  import jakarta.validation.constraints.Size;  import lombok.Data;  @Data  public class StudentEdit {  int id;  @NotEmpty @NotBlank  @Size(min=8, max=12)  String studentNo;  @NotEmpty @NotBlank  @Size(min=2, max=20)  String name;  @NotEmpty @NotBlank  @Pattern(regexp="010-[0-9]{3,4}-[0-9]{4}")  String phone;  @NotEmpty @Email  String email;  @NotEmpty @NotBlank  @Pattern(regexp="남|여")  String sex;  int departmentId;  } |

학생 등록/수정 폼에서 입력한 데이터를 채우기 위한 모델 클래스이다.

입력된 데이터를 검증하기 위한 spring form validation annotation을 멤버 변수에 붙였다.

|  |  |
| --- | --- |
| annotation | 설명 |
| @NotEmpty | 값이 입력되었는지 검사한다 |
| @NotBlank | 공백만 입력된 것은 아닌지 검사한다 |
| @Size(min=2, max=20) | 입력된 문자열의 최소 크기 최대 크기를 검사한다 |
| @Pattern(regexp="남|여") | 입력된 문자열이 정규식에 일치하는지 검사한다 |
| @Email | 이메일 주소 문자열인지 검사한다 |

@Pattern(regexp="010-[0-9]{3,4}-[0-9]{4}")

노란색으로 칠한 부분은 정규식에서 평범한 문자열이다.

정규식에서 평범한 문자열은 찾을 문자열과 그대로 일치해야 한다.

하늘색으로 칠한 부분은 정규식이다.

[0-9] 0 부터 9 사이의 아무 문자 하나

{3,4} 바로 앞의 내용일 3개 이상 4개 이하 반복된다

[0-9]{3,4} 0 부터 9 사이의 아무 문자가 3개 이상 4개 이하 반복된다

[0-9]{4} 0 부터 9 사이의 아무 문자가 4개 반복된다

@Pattern(regexp="남|여")

"남" 이거나 "여"

## Student Mapper 구현

src/main/java/net/skhu/mapper/StudentMapper.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 | package net.skhu.mapper;  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.Select;  import org.apache.ibatis.annotations.Update;  import net.skhu.dto.Student;  @Mapper  public interface StudentMapper {  @Select("SELECT \* FROM student WHERE id = #{id}")  Student findOne(int id);  @Select("SELECT \* FROM student WHERE studentNo = #{studentNo}")  Student findByStudentNo(String studentNo);  @Select("""  SELECT s.\*, d.name departmentName  FROM student s LEFT JOIN department d ON s.departmentId = d.id """)  List<Student> findAll();  @Insert("""  INSERT student (studentNo, name, departmentId, phone, sex, email)  VALUES (#{studentNo}, #{name}, #{departmentId}, #{phone}, #{sex}, #{email}) """)  @Options(useGeneratedKeys=true, keyProperty="id")  void insert(Student student);  @Update("""  UPDATE student SET  studentNo= #{studentNo},  name = #{name},  departmentId = #{departmentId},  phone = #{phone},  sex = #{sex},  email = #{email}  WHERE id = #{id} """)  void update(Student student);  @Delete("DELETE FROM student WHERE id = #{id}")  void delete(int id);  } |

mapper와 서비스 클래스 사이에는 DTO 객체만 사용해야 한다.

모델 객체는 뷰와 컨트롤러 사이에서만 사용되어야 한다. (Model, View, Controller)

## StudentService 클래스

src/main/java/net/skhu/service/StudentService.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 | package net.skhu.service;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import net.skhu.dto.Student;  import net.skhu.mapper.StudentMapper;  import net.skhu.model.StudentEdit;  @Service  public class StudentService {  @Autowired  StudentMapper studentMapper;  public StudentEdit findOne(int id) {  Student studentDto = studentMapper.findOne(id);  return toEditModel(studentDto);  }  public Student findByStudentNo(String studentNo) {  return studentMapper.findByStudentNo(studentNo);  }  public List<Student> findAll() {  return studentMapper.findAll();  }  public void insert(StudentEdit studentEdit) {  Student student = toDto(studentEdit);  studentMapper.insert(student);  }  public void update(StudentEdit studentEdit) {  Student student = toDto(studentEdit);  studentMapper.update(student);  }  public void delete(int id) {  studentMapper.delete(id);  }  public Student toDto(StudentEdit studentEdit) {  Student studentDto = new Student();  studentDto.setId(studentEdit.getId());  studentDto.setStudentNo(studentEdit.getStudentNo());  studentDto.setName(studentEdit.getName());  studentDto.setDepartmentId(studentEdit.getDepartmentId());  studentDto.setEmail(studentEdit.getEmail());  studentDto.setPhone(studentEdit.getPhone());  studentDto.setSex(studentEdit.getSex());  return studentDto;  }  public StudentEdit toEditModel(Student studentDto) {  StudentEdit studentEdit = new StudentEdit();  studentEdit.setId(studentDto.getId());  studentEdit.setStudentNo(studentDto.getStudentNo());  studentEdit.setName(studentDto.getName());  studentEdit.setDepartmentId(studentDto.getDepartmentId());  studentEdit.setEmail(studentDto.getEmail());  studentEdit.setPhone(studentDto.getPhone());  studentEdit.setSex(studentDto.getSex());  return studentEdit;  }  } |

**public StudentEdit findOne(int id)**

이 메소드는 학생 수정 화면의 입력 폼에 채울 학생 객체를 조회해서 리턴한다.

학생 수정 화면에서는 StudentEdit 모델 객체를 사용해야 한다.

그런데 StudentMapper 클래스의 findOne 메소드가 리턴한 것은 Student DTO 객체이다.

그래서 toEditModel 메소드를 호출하여, Student DTO 객체를 StudentEdit 모델 객체로 변환해서 리턴한다.

**StudentEdit 모델 클래스**

학생 등록/수정 화면에서는 StudentEdit 모델 클래스를 사용한다.

그래서 insert, update 메소드에 전달되는 객체가 StudentEdit 이다.

그런데 StudentMapper에는 Student DTO 객체를 전달해야 한다.

mapper와 서비스 클래스 사이에는 DTO 객체만 사용해야 하기 때문이다.

그래서 다음과 같이 toDto 메소드를 호출하여, StudentEdit 모델 객체를 Student DTO 객체로 변환한다.

Student student = toDto(studentEdit);

## Student Controller 구현

src/main/java/net/skhu/controller/StudentController.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  6  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91 | package net.skhu.controller;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.validation.BindingResult;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.RequestMapping;  import jakarta.validation.Valid;  import net.skhu.dto.Department;  import net.skhu.dto.Student;  import net.skhu.model.StudentEdit;  import net.skhu.service.DepartmentService;  import net.skhu.service.StudentService;  @Controller  @RequestMapping("student")  public class StudentController {  @Autowired StudentService studentService;  @Autowired DepartmentService departmentService;  @GetMapping("list")  public String list(Model model) {  List<Student> students = studentService.findAll();  model.addAttribute("students", students);  return "student/list";  }  @GetMapping("create")  public String create(Model model) {  StudentEdit studentEdit = new StudentEdit();  List<Department> departments = departmentService.findAll();  model.addAttribute("studentEdit", studentEdit);  model.addAttribute("departments", departments);  return "student/edit";  }  @PostMapping("create")  public String create(Model model,  @Valid StudentEdit studentEdit, BindingResult bindingResult) {  if (bindingResult.hasErrors()) {  model.addAttribute("departments", departmentService.findAll());  return "student/edit";  }  Student student2 = studentService.findByStudentNo(studentEdit.getStudentNo());  if (student2 != null) {  bindingResult.rejectValue("studentNo", null, "학번이 중복됩니다.");  model.addAttribute("departments", departmentService.findAll());  return "student/edit";  }  studentService.insert(studentEdit);  return "redirect:list";  }  @GetMapping("edit")  public String edit(Model model, int id) {  StudentEdit studentEdit = studentService.findOne(id);  List<Department> departments = departmentService.findAll();  model.addAttribute("studentEdit", studentEdit);  model.addAttribute("departments", departments);  return "student/edit";  }  @PostMapping("edit")  public String edit(Model model,  @Valid StudentEdit studentEdit, BindingResult bindingResult) {  if (bindingResult.hasErrors()) {  model.addAttribute("departments", departmentService.findAll());  return "student/edit";  }  Student student2 = studentService.findByStudentNo(studentEdit.getStudentNo());  if (student2 != null && student2.getId() != studentEdit.getId()) {  bindingResult.rejectValue("studentNo", null, "학번이 중복됩니다.");  model.addAttribute("departments", departmentService.findAll());  return "student/edit";  }  studentService.update(studentEdit);  return "redirect:list";  }  @GetMapping("delete")  public String delete(Model model, int id) {  studentService.delete(id);  return "redirect:list";  }  } |

bindingResult.rejectValue("studentNo", null, "학번이 중복됩니다.");

bindingResult 객체에 에러 메시지를 등록한다.

에러가 발생한 속성은 "studentNo" 이고,

에러 메시지는 "학번이 중복됩니다." 이다.

## 정적 컨텐츠

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 | 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; }  div.error { color: red; } |

## student/list 뷰 구현

src/main/resources/templates/student/list.html

|  |  |
| --- | --- |
| 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 | <!DOCTYPE html>  <html lang="ko" xmlns:th="http://www.thymeleaf.org">  <head>  <meta charset="utf-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  a.btn { float: right; margin-top: -40px; }  </style>  </head>  <body>  <div class="container">  <h1>학생목록</h1>  <a href="create" class="btn">학생등록</a>  <table class="list">  <thead>  <tr>  <th>id</th>  <th>학번</th>  <th>이름</th>  <th>학과</th>  <th>전화</th>  <th>성별</th>  <th>이메일</th>  </tr>  </thead>  <tbody>  <tr th:each="st : ${ students }">  <td th:text="${ st.id }"></td>  <td><a th:text="${ st.studentNo }" th:href="${ 'edit?id=' + st.id }"></a></td>  <td th:text="${ st.name }"></td>  <td th:text="${ st.departmentName }"></td>  <td th:text="${ st.phone }"></td>  <td th:text="${ st.sex }"></td>  <td th:text="${ st.email }"></td>  </tr>  </tbody>  </table>  </div>  </body>  </html> |

## student/edit 뷰 구현

src/main/resources/templates/student/edit.html

|  |  |
| --- | --- |
| 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 | <!DOCTYPE html>  <html lang="ko" xmlns:th="http://www.thymeleaf.org">  <head>  <meta charset="utf-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  form { width: 600px; margin: auto; padding: 5px 20px; box-shadow: 2px 2px 5px gray; }  td { min-width: 100; padding: 5px; }  td:nth-child(1) { text-align: right; }  button { margin: 5px 0 20px 20px; }  </style>  </head>  <body>  <div class="container">  <form method="post" th:object="${studentEdit}">  <h1 th:text="${studentEdit.id > 0 ? '학생 수정' : '학생 등록'}"></h1>  <table>  <tr>  <td>학번:</td>  <td><input type="text" th:field="\*{studentNo}" />  <div class="error" th:errors="\*{studentNo}"></div>  </td>  </tr>  <tr>  <td>이름:</td>  <td><input type="text" th:field="\*{name}" />  <div class="error" th:errors="\*{name}"></div>  </td>  </tr>  <tr>  <td>학과:</td>  <td><select th:field="\*{departmentId}">  <option th:each="dt : ${ departments }"  th:value="${ dt.id }" th:text="${ dt.name }">  </option>  </select>  <div class="error" th:errors="\*{departmentId}"></div>  </td>  </tr>  <tr>  <td>전화:</td>  <td><input type="text" th:field="\*{phone}" />  <div class="error" th:errors="\*{phone}"></div>  </td>  </tr>  <tr>  <td>성별:</td>  <td><input type="text" th:field="\*{sex}" />  <div class="error" th:errors="\*{sex}"></div>  </td>  </tr>  <tr>  <td>이메일:</td>  <td><input type="text" th:field="\*{email}" />  <div class="error" th:errors="\*{email}"></div>  </td>  </tr>  </table>  <hr />  <div>  <button type="submit" class="btn">저장</button>  <a th:if="${ studentEdit.id > 0 }" th:href="${ 'delete?id=' + studentEdit.id }"  class="btn" onclick="return confirm('삭제하시겠습니까?')">삭제</a>  <a href="list" class="btn">목록으로</a>  </div>  </form>  </div>  </body>  </html> |

<div class="error" th:errors="\*{studentNo}">

studentNo 속성에 등록된 에러 메시지가 출력된다.

## 실행

A screenshot of a computer

Description automatically generated

아무것도 입력하지 않고 저장

# 연습 문제

professor 테이블에 대해서서

spring form validation을 구현하라