Spring form validation #3

2023-09-16

이승진

**학습목표**

spring form validation 메시지 수정

삭제 요청 POST

exception handling

**목차**

[1. 배경지식 2](#_Toc145761720)

[1) 항목별 에러 메시지 등록 2](#_Toc145761721)

[2) select 태그 입력 확인 3](#_Toc145761722)

[3) 삭제 요청 POST 4](#_Toc145761723)

[4) exception handling 6](#_Toc145761724)

[5) 저장할 데이터 검사 6](#_Toc145761725)

[6) 수정 페이지로 넘어가기 7](#_Toc145761726)

[7) 테이블 레이아웃 8](#_Toc145761727)

[2. mybatis4 프로젝트 11](#_Toc145761728)

[1) 프로젝트 생성 11](#_Toc145761729)

[2) pom.xml 12](#_Toc145761730)

[3) application.properties 13](#_Toc145761731)

[4) ValidationMessages.properties 파일 생성 13](#_Toc145761732)

[5) Department DTO 클래스 13](#_Toc145761733)

[6) DepartmentMapper.java 13](#_Toc145761734)

[7) DepartmentService.java 14](#_Toc145761735)

[8) Student DTO 클래스 14](#_Toc145761736)

[9) StudentEdit 모델 클래스 15](#_Toc145761737)

[10) Student Mapper 구현 16](#_Toc145761738)

[11) StudentService 클래스 17](#_Toc145761739)

[12) Student Controller 구현 19](#_Toc145761740)

[13) 정적 컨텐츠 21](#_Toc145761741)

[14) student/list 뷰 구현 22](#_Toc145761742)

[15) student/edit 뷰 구현 23](#_Toc145761743)

[3. 연습 문제 25](#_Toc145761744)

# 배경지식

## 항목별 에러 메시지 등록

지난 강의 예제에서 구현한 spring form validation 메시지는

어노테이션의 에러 메시지를 일괄 수정하는 방식이었다.

A screenshot of a computer

Description automatically generated

위 화면에서 노란색으로 칠한 부분의 에러 메시지는 별로 바람직하지 않다.

위 화면에서 노란색으로 칠한 부분은 아래의 설정에 의한 출력이다.

|  |
| --- |
| jakarta.validation.constraints.Pattern.message={regexp} 패턴과 일치해야 합니다. |

위 화면의 노란색 에러 메시지들을 다음과 같이 수정하자.

"휴대폰 번호를 입력하세요 (예: 010-0000-0000)"

"남, 여 중 하나를 입력하세요."

### 항목별 에러 메시지

|  |
| --- |
| @NotEmpty @NotBlank  @Pattern(regexp="010-[0-9]{3,4}-[0-9]{4}",  message="휴대폰 번호를 입력하세요 예:010-000-0000")  String phone; |

|  |
| --- |
| @NotEmpty @NotBlank  @Pattern(regexp="남|여", message="남, 여 중 하나를 입력하세요.")  String sex; |

## select 태그 입력 확인

|  |
| --- |
| <select th:field="\*{departmentId}">  <option value="0">학과를 선택하세요</option>  <option th:each="dt : ${ departments }"  th:value="${ dt.id }" th:text="${ dt.name }">  </option>  </select> |

"학과를 선택하세요" 항목의 값은 0 이다.

학과를 선택하지 않으면, departmentId 속성값은 0 이 된다.

|  |
| --- |
| @Min(value=1, message="학과를 선택하세요.")  int departmentId; |

위 어노테이션은 departmentId 속성의 값이 1 이상인 것을 확인한다.

만약 departmentId 값이 0 이라면, "학과를 선택하세요." 에러 메시지가 출력된다.

|  |  |
| --- | --- |
| 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 |
| @Min(1) | 입력된 값이 최소 1 이상인지 검사한다 |
| @Max(10) | 입력된 값이 최대 10 이하인지 검사한다 |

## 삭제 요청 POST

지난 강의 예제에서 삭제 기능을 다음과 같이 GET 요청으로 구현했다.

|  |
| --- |
| <a th:if="${ studentEdit.id > 0 }" th:href="${ 'delete?id=' + studentEdit.id }"  class="btn" onclick="return confirm('삭제하시겠습니까?')">삭제</a> |

삭제 버튼이 <a> 태그 이미로, 삭제 버튼을 클릭하면 'delete?id=##' URL을 GET 방식 요청한다.

|  |
| --- |
| @GetMapping("delete")  public String delete(Model model, int id) {  studentService.delete(id);  return "redirect:list";  } |

삭제 버튼을 클릭하면 서버에서 delete 액션 메소드가 호출된다.

데이터 등록/수정/삭제 등의 요청을 GET 방식으로 요청하는 것은 바람직하지 않다.

GET 방식 요청은,

웹 브라우저나 인터넷 중간에 캐시 기능에 의해서 처리 되어 버리고

서버에 전달되지 않을 수 있다.

데이터 조회를 GET 방식 요청했다면,

조회 요청이 서버에 전달되지 않고,

캐시에 저장된 조회 결과만 출력 되어도 별 문제 없다.

그런데 등록/수정/삭제를 GET 방식 요청하면,

그 등록/수정/삭제 요청이 서버에 전달되지 않고,

캐시에 저장된 작업 완료 화면만 웹브라우저에 출력될 수 있다.

서버에서 그 작업이 실행되지 않았지만, 사용자는 작업이 정상 완료된 것으로 착각하게 된다.

### 삭제 버튼

|  |
| --- |
| <button type="submit" class="btn"  name="cmd" value="save">저장</button>  <button type="submit" class="btn"  th:if="${ studentEdit.id > 0 }"  onclick="return confirm('삭제하시겠습니까?')"  name="cmd" value="delete">삭제</button> |

저장 버튼과 마찬가지로 삭제 버튼도 <button type="submit">삭제</button> 태그로 구현하자.

삭제 버튼을 클릭하면, 저장 버튼을 클릭했을 때와 똑같이 form이 submit 되고,

현재 URL(student/edit)을 POST 요청하게 된다.

저장 버튼을 클릭해서 form이 submit 되는 경우에는 서버에 전달된 request parameter에

name="cmd" value="save" 값이 들어있다.

삭제 버튼을 클릭해서 form이 submit 되는 경우에는 서버에 전달된 request parameter에

name="cmd" value="delete" 값이 들어있고,

### 액션 메소드

|  |
| --- |
| @Controller  @RequestMapping("student")  public class StudentController {  ... 생략 ...  @PostMapping(value="edit", params="cmd=save")  public String edit(Model model,  @Valid StudentEdit studentEdit, BindingResult bindingResult) {  ... 생략 ...  @PostMapping(value="edit", params="cmd=delete")  public String delete(Model model,  StudentEdit studentEdit, BindingResult bindingResult) {  ... 생략 ... |

student/edit URL이 POST 요청된 경우에 위 두 액션 메소드들 중 하나가 호출된다.

request parameter에 name="cmd" value="save" 값이 들어있으면 edit 액션 메소드가 호출되고,

request parameter에 name="cmd" value="delete" 값이 들어있으면 delete 액션 메소드가 호출된다.

삭제 요청의 경우에는 입력된 데이터를 검사할 필요 없다.

즉 StudentEdit 모델 객체에 들어있는 데이터를 검사할 필요 없다.

그래서 StudentEdit 모델 객체 앞에 @Valid 어노테이션이 없다.

그렇지만 삭제가 실패했을 때 에러 메시지를 뷰에 전달해야 한다.

그래서 에러 메시지를 전달하기 위해 BindingResult 객체는 여전히 필요하다.

### BindingResult 에러 메시지 등록

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

위 코드는 StudentEdit 객체의 studentNo 속성에 대한 에러 메시지를 등록한다.

|  |
| --- |
| bindingResult.rejectValue("", null, "등록할 수 없습니다.");  bindingResult.rejectValue("", null, "삭제할 수 없습니다."); |

위 코드는 어떤 속성이 아니고 StudentEdit 객체 전체에 대한 에러 메시지를 등록한다.

|  |
| --- |
| <form method="post" th:object="${studentEdit}">  <div class="error" th:errors="\*{studentNo}"></div> |

위 코드는 studentEdit 객체의 studentNo 속성에 대한 에러 메시지를 출력한다.

|  |
| --- |
| <div class="error" th:errors="${studentEdit}"></div> |

위 코드는 studentEdit 객체 전체에 대한 에러 메시지를 출력한다.

## exception handling

DB에 데이터를 등록/수정/삭제 하다가 에러(exception)가 발생할 수 있다.

학생 수정 화면에서 저장 버튼이나 삭제 버튼을 클릭해서 에러가 발생했을 때,

에러 메시지와 함께 학생 수정 화면이 다시 출력되어야 할 것이다.

그러려면, 액션 메소드에 exception handling 기능을 구현해야 한다.

|  |
| --- |
| @PostMapping("create")  public String create(Model model,  @Valid StudentEdit studentEdit, BindingResult bindingResult) {  try {  studentService.insert(studentEdit, bindingResult);  return "redirect:list";  }  catch (Exception e) {  model.addAttribute("departments", departmentService.findAll());  bindingResult.rejectValue("", null, "등록할 수 없습니다.");  return "student/edit";  }  } |

학생 등록 작업이 성공하면,

연두색 코드가 실행되고, 학생 목록 화면으로 리다이렉트 한다.

학생 등록 작업 도중 에러가 발생하면

노란색 코드가 실행되고, "등록할 수 없습니다" 에러 메시지와 함께, 학생 등록 화면이 다시 출력된다.

## 저장할 데이터 검사

|  |
| --- |
| @PostMapping("create")  public String create(Model model,  @Valid StudentEdit studentEdit, BindingResult bindingResult) {  if (studentService.hasErrors(studentEdit, bindingResult)) {  model.addAttribute("departments", departmentService.findAll());  return "student/edit";  }  studentService.insert(studentEdit);  return "redirect:list";  } |

지난 강의 예제에서 등록할 데이터 검사하는 코드를, 컨트롤러 클래스에 위와 같이 구현했다.

컨트롤러 클래스에서 hasErrors 메소드를 호출하여 에러를 검사하고

에러가 없으면 insert 메소드를 호출하였다.

그런데 어떤 작업 코드는 가급적 서비스 클래스에 구현하는 것이 좋다.

이번 강의 예제에서 등록할 데이터 검사하는 코드를, 서비스 클래스에 다음과 같이 구현한다.

|  |
| --- |
| public void insert(StudentEdit studentEdit,  BindingResult bindingResult) throws Exception {  if (hasErrors(studentEdit, bindingResult))  throw new Exception("입력 데이터 오류");  Student student = toDto(studentEdit);  studentMapper.insert(student);  } |

컨트롤러 클래스는 데이터를 검사하지 않는다. 즉 hasErrors 메소드를 호출하지 않고,

그냥 바로 서비스 클래스의 insert 메소드를 호출한다.

서비스 클래스의 insert 메소드에서 hasErrors 메소드를 호출하여 데이터를 검사한다.

데이터에 오류가 있으면 exception을 throw 한다.

## 수정 페이지로 넘어가기

지난 강의 예제에서 수정 페이지로 넘어가기 위한 링크를 다음과 같이 구현했다.

|  |
| --- |
| <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> |

예를 들어 studentNo 값이 201014199 이고, id 값이 3 이라면, 다음과 같은 a 태그가 출력된다.

<a href="edit?id=3">201014199</a>

그런데, 위와 같이 구현하면, 학번을 클릭해야만 수정 페이지로 넘어갈 수 있다.

tr 태그 내부 아무 곳이나 클릭해도 수정 페이지로 넘어갈 수 있도록 구현하자.

|  |
| --- |
| <tr th:each="st : ${ students }"  th:href="${ 'edit?id=' + st.id }"  onclick="onClickHref(event)">  <td th:text="${ st.id }"></td>  <td th:text="${ st.studentNo }"></td> |

예를 들어 studentNo 값이 201014199 이고, id 값이 3 이라면, 다음과 같은 tr 태그가 출력된다.

<tr href="edit?id=3" onclick="onClickHref(event)">

그런데 a 태그를 클릭하면 href 속성의 URL 주소로 자동으로 넘어가지만,

tr 태그에는 그런 기능이 없다.

따라서 javascript 구현이 필요하다.

위 tr 태그를 클릭하면, onClickHref(event) 함수가 호출된다.

여기서 event 파라미터는 클릭 이벤트에 대한 정보가 담겨있는 객체이다.

|  |
| --- |
| **function** onClickHref(event) {  **let** url = event.currentTarget.getAttribute("href");  location.href = url;  } |

**let** url = event.currentTarget.getAttribute("href");

이 코드는 클릭된 태그의 href 속성값을 꺼내서 지역변수 url에 대입한다.

아래와 같은 태그가 클릭되었다면, url 변수에 "edit?id=3" 문자열이 대입된다.

<tr href="edit?id=3" onclick="onClickHref(event)">

location.href = url;

location 객체의 href 속성에 URL 주소를 대입하면,

웹브라우저 창이 그 URL로 넘어간다.

즉 웹브라우저 창이 "edit?id=3" URL로 넘어간다.

이 URL은 상대 URL 이다.

현재 웹브라우저 창의 URL이 "http://localhost:8088/student/list" 이므로

이 현재 URL에서 마지막 이름 부분만 상대 URL로 교체된다.

"http://localhost:8088/student/edit?id=3"

즉 위 주소로 웹브라우저 창이 넘어가게 된다.

## 테이블 레이아웃

지난 강의에서는 입력폼의 레이아웃을 위해 table, tr, td 태그를 사용했다.

그런데 table, tr, td 태그는

표 형태의 데이터를 출력할 때만 사용하는 것이 좋고,

레이아웃을 위한 용도로 사용하는 것은 바람직하지 않다.

테이블 형태의 레이아웃을 구현하려면, 다음과 같이 div 태그를 사용하자.

### table1.html - table 태그로 표 구현

|  |
| --- |
| <!DOCTYPE html>  <html lang="ko">  <head>  <style>  table { border-collapse: collapse; }  td { border: 1px solid gray; padding: 5px; }  </style>  </head>  <body>  <table>  <tr>  <td>1,1</td>  <td>1,2</td>  <td>1,3</td>  </tr>  <tr>  <td>2,1</td>  <td>2,2</td>  <td>2,3</td>  </tr>  <tr>  <td>3,1</td>  <td>3,2</td>  <td>3,3</td>  </tr>  </table>  </body>  </html> |

A grid of numbers in black and white

Description automatically generated

### table2.html - div 태그로 표 구현

|  |
| --- |
| <!DOCTYPE html>  <html lang="ko">  <head>  <style>  div.table { display: table; }  div.row { display: table-row; }  div.cell { display: table-cell; border: 1px solid gray; padding: 5px; }  </style>  </head>  <body>  <div class="table">  <div class="row">  <div class="cell">1,1</div>  <div class="cell">1,2</div>  <div class="cell">1,3</div>  </div>  <div class="row">  <div class="cell">2,1</div>  <div class="cell">2,2</div>  <div class="cell">2,3</div>  </div>  <div class="row">  <div class="cell">3,1</div>  <div class="cell">3,2</div>  <div class="cell">3,3</div>  </div>  </div>  </body>  </html> |

display: table;

display: table-row;

display: table-cell;

위 서식을 사용하면 div 태그가 table, tr, td 태그 처럼 배치된다.

table, tr, td 태그로 표를 구현했을 때, 각 칸의 경계선이 중복되지 않는다.

그런데 div 태그들로 표를 구현하면, 각 칸의 경계선이 중복된다.

즉 첫 칸의 right border와, 그 다음 칸의 left border가 중첩되어 두껍게 그려진다.

A grid of numbers in a white box

Description automatically generated

### table3.html - div 태그로 표 구현

|  |
| --- |
| <!DOCTYPE html>  <html lang="ko">  <head>  <style>  div.table { display: table; }  div.row { display: table-row; }  div.cell { display: table-cell; border: 1px solid gray; padding: 5px; }  div.cell + div.cell { border-left: none; }  div.row + div.row div.cell { border-top: none; }  </style>  </head>  <body>  <div class="table">  <div class="row">  <div class="cell">1,1</div>  <div class="cell">1,2</div>  <div class="cell">1,3</div>  </div>  <div class="row">  <div class="cell">2,1</div>  <div class="cell">2,2</div>  <div class="cell">2,3</div>  </div>  <div class="row">  <div class="cell">3,1</div>  <div class="cell">3,2</div>  <div class="cell">3,3</div>  </div>  </div>  </body>  </html> |

div.cell + div.cell { border-left: none; }

바로 앞에 div.cell 태그가 있는 div.cell 태그는 left border를 그리지 말아라

div.row + div.row div.cell { border-top: none; }

바로 앞에 div.row 태그가 있는 div.row 태그 아래 div.cell 태그들은 top border를 그리지 말아라.

A grid of numbers in black and white

Description automatically generated

# mybatis4 프로젝트

## 프로젝트 생성

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Name | mybatis4 |
| Java Version | 17 |
| Package | net.skhu |

A screenshot of a computer

Description automatically generated

## pom.xml

|  |
| --- |
| <dependency>  <groupId>org.modelmapper</groupId>  <artifactId>modelmapper</artifactId>  <version>3.1.1</version>  </dependency> |

ModelMapper maven dependency를 추가하자.

## application.properties

src/main/resources/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 |

## ValidationMessages.properties 파일 생성

src/main/resources/ValidationMessages.properties

|  |
| --- |
| jakarta.validation.constraints.Size.message=크기가 {min} 이상 {max} 이하이어야 합니다.  jakarta.validation.constraints.Email.message=이메일 주소가 바르지 않습니다.  jakarta.validation.constraints.NotEmpty.message=필수 입력항목입니다.  jakarta.validation.constraints.Max.message={value} 이하이어야 합니다.  jakarta.validation.constraints.Min.message={value} 이상이어야 합니다.  jakarta.validation.constraints.Pattern.message={regexp} 패턴과 일치해야 합니다. |

## 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();  }  } |

## 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  35  36  37 | package net.skhu.model;  import jakarta.validation.constraints.Email;  import jakarta.validation.constraints.Min;  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}",  message="휴대폰 번호를 입력하세요 예:010-000-0000")  String phone;  @NotEmpty @Email  String email;  @NotEmpty @NotBlank  @Pattern(regexp="남|여", message="남, 여 중 하나를 입력하세요.")  String sex;  @Min(value=1, message="학과를 선택하세요.")  int departmentId;  } |

|  |  |
| --- | --- |
| 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 |
| @Min(1) | 입력된 값이 최소 1 이상인지 검사한다 |
| @Max(10) | 입력된 값이 최대 10 이하인지 검사한다 |

## 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.

## 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  67  68 | package net.skhu.service;  import java.util.List;  import org.modelmapper.ModelMapper;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.validation.BindingResult;  import net.skhu.dto.Student;  import net.skhu.mapper.StudentMapper;  import net.skhu.model.StudentEdit;  @Service  public class StudentService {  @Autowired StudentMapper studentMapper;  ModelMapper modelMapper = new ModelMapper();  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,  BindingResult bindingResult) throws Exception {  if (hasErrors(studentEdit, bindingResult))  throw new Exception("입력 데이터 오류");  Student student = toDto(studentEdit);  studentMapper.insert(student);  }  public void update(StudentEdit studentEdit,  BindingResult bindingResult) throws Exception {  if (hasErrors(studentEdit, bindingResult))  throw new Exception("입력 데이터 오류");  Student student = toDto(studentEdit);  studentMapper.update(student);  }  public void delete(int id) {  studentMapper.delete(id);  }  public Student toDto(StudentEdit studentEdit) {  return modelMapper.map(studentEdit, Student.class);  }  public StudentEdit toEditModel(Student studentDto) {  return modelMapper.map(studentDto, StudentEdit.class);  }  public boolean hasErrors(StudentEdit studentEdit, BindingResult bindingResult) {  if (bindingResult.hasErrors()) return true;  Student student2 = findByStudentNo(studentEdit.getStudentNo());  if (student2 != null && student2.getId() != studentEdit.getId()) {  bindingResult.rejectValue("studentNo", null, "학번이 중복됩니다.");  return true;  }  return false;  }  } |

## 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) {  try {  studentService.insert(studentEdit, bindingResult);  return "redirect:list";  }  catch (Exception e) {  model.addAttribute("departments", departmentService.findAll());  bindingResult.rejectValue("", null, "등록할 수 없습니다.");  return "student/edit";  }  }  @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(value="edit", params="cmd=save")  public String edit(Model model,  @Valid StudentEdit studentEdit, BindingResult bindingResult) {  try {  studentService.update(studentEdit, bindingResult);  return "redirect:list";  }  catch (Exception e) {  model.addAttribute("departments", departmentService.findAll());  bindingResult.rejectValue("", null, "수정할 수 없습니다.");  return "student/edit";  }  }  @PostMapping(value="edit", params="cmd=delete")  public String delete(Model model,  StudentEdit studentEdit, BindingResult bindingResult) {  try {  studentService.delete(studentEdit.getId());  return "redirect:list";  }  catch (Exception e) {  model.addAttribute("departments", departmentService.findAll());  bindingResult.rejectValue("", null, "삭제할 수 없습니다.");  return "student/edit";  }  }  } |

## 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  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; }  div.error { color: red; }  tr[href]:hover { background-color: #ffa; cursor: pointer; } |

다음과 같이 href 속성이 있는 tr 태그에 마우스가 위치하면, 노란색 배경이 그려진다.

<tr href="edit?id=3"

src/main/resources/static/common.js

|  |  |
| --- | --- |
| 1  2  3  4 | **function** onClickHref(event) {  **let** url = event.currentTarget.getAttribute("href");  location.href = url;  } |

## 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  41  42  43 | <!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" />  <script type="text/javascript" src="/common.js"></script>  <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 : ${ students }"  th:href="${ 'edit?id=' + st.id }"  onclick="onClickHref(event)">  <td th:text="${ st.id }"></td>  <td th:text="${ st.studentNo }"></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  70  71  72  73  74  75  76  77  78  79  80  81  82 | <!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; }  div.table { display: table; }  div.row { display: table-row; }  div.cell { display: table-cell; padding: 5px; }  div.cell:nth-child(1) { width: 100; text-align: right; vertical-align: top; }  div.buttons { margin: 1em; }  </style>  </head>  <body>  <div class="container">  <form method="post" th:object="${studentEdit}">  <h1 th:text="${studentEdit.id > 0 ? '학생 수정' : '학생 등록'}"></h1>  <div class="table">  <div class="row">  <div class="cell">학번:</div>  <div class="cell">  <input type="text" th:field="\*{studentNo}" />  <div class="error" th:errors="\*{studentNo}"></div>  </div>  </div>  <div class="row">  <div class="cell">이름:</div>  <div class="cell">  <input type="text" th:field="\*{name}" />  <div class="error" th:errors="\*{name}"></div>  </div>  </div>  <div class="row">  <div class="cell">학과:</div>  <div class="cell">  <select th:field="\*{departmentId}">  <option value="0">학과를 선택하세요</option>  <option th:each="dt : ${ departments }"  th:value="${ dt.id }" th:text="${ dt.name }">  </option>  </select>  <div class="error" th:errors="\*{departmentId}"></div>  </div>  </div>  <div class="row">  <div class="cell">전화:</div>  <div class="cell">  <input type="text" th:field="\*{phone}" />  <div class="error" th:errors="\*{phone}"></div>  </div>  </div>  <div class="row">  <div class="cell">성별:</div>  <div class="cell">  <input type="text" th:field="\*{sex}" />  <div class="error" th:errors="\*{sex}"></div>  </div>  </div>  <div class="row">  <div class="cell">이메일:</div>  <div class="cell">  <input type="text" th:field="\*{email}" />  <div class="error" th:errors="\*{email}"></div>  </div>  </div>  </div>  <hr />  <div class="buttons">  <button type="submit" class="btn"  name="cmd" value="save">저장</button>  <button type="submit" class="btn"  th:if="${ studentEdit.id > 0 }"  onclick="return confirm('삭제하시겠습니까?')"  name="cmd" value="delete">삭제</button>  <a href="list" class="btn">목록으로</a>  </div>  <div class="error" th:errors="${studentEdit}"></div>  </form>  </div>  </body>  </html> |

# 연습 문제

professor 테이블에 대해서서

spring form validation을 구현하라