**Spring Form Validation 1**

2021-11-01

이승진

**학습목표**

spring form validation 구현

모델 클래스, DTO 클래스 개념

**목차**

[1. 배경지식 2](#_Toc87253052)

[1) spring form validation 2](#_Toc87253053)

[2) 모델(model) 클래스 4](#_Toc87253054)

[3) 서비스 클래스 5](#_Toc87253055)

[2. form2 프로젝트 6](#_Toc87253056)

[1) bbs1 데이터베이스 6](#_Toc87253057)

[2) 프로젝트 생성 7](#_Toc87253058)

[3) pom.xml 9](#_Toc87253059)

[4) application.properties 9](#_Toc87253060)

[5) Department.java 10](#_Toc87253061)

[6) User.java 10](#_Toc87253062)

[7) DepartmentMapper.java 12](#_Toc87253063)

[8) UserMapper.java 12](#_Toc87253064)

[9) UserRegister.java 13](#_Toc87253065)

[10) DepartmentService.java 14](#_Toc87253066)

[11) UserService.java 15](#_Toc87253067)

[12) UserController.java 16](#_Toc87253068)

[13) common.css 18](#_Toc87253069)

[14) user/list.jsp 19](#_Toc87253070)

[15) user/register.jsp 20](#_Toc87253071)

[16) 실행 21](#_Toc87253072)

[3. 기타 22](#_Toc87253073)

[1) Unique Index 22](#_Toc87253074)

[2) spring form validatoin annotation 23](#_Toc87253075)

# 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.(이것도 자동임)

model.addAttribute("student", student);

(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.jsp 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 (병장 -> 상병)

StudentService 클래스에서는 StudentDAO 클래스의 메소드를 호출하고 (상병 -> 일병)

StudentDAO 클래스에서는 StudentMapper 메소드를 호출하는 형태가 바람직하다 (일병 -> 이병)

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.



# form2 프로젝트

## bbs1 데이터베이스

Diagram

Description automatically generated

## 프로젝트 생성

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## pom.xml

|  |
| --- |
| <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency>  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-validation</artifactId>  </dependency> |

spring form validation 라이브러리를 위한 maven dependency

## application.properties

### src/main/resources/application.properties

|  |
| --- |
| 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/bbs1?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=skhuA+4.5  server.port=8088 |

## Department.java

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

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

## User.java

### src/main/java/net/skhu/entity/User.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package net.skhu.dto;  import lombok.Data;  @Data  public class User {  int id;  String userid;  String password;  String name;  String email;  boolean enabled;  String userType;  Integer departmentId;  } |

User 테이블의 departmentId 속성은 null 값이 가능한 필드이다.

따라서 int 속성에 대입할 수 없고, Integer 타입의 속성에 대입해야 한다.

Table

Description automatically generated

위 화면에 NN 컬럼은 Not Nullable 여부를 체크한다.

이 컬럼이 체크되어 있으면, null 값이 가능하지 않다는 뜻이고,

이 컬람의 체크가 꺼져 있으면, null 값이 가능하다는 뜻이다.

따라서 departmentId는 null값이 가능한 Integer(기본자료형 클래스타입)로 선언해야 함.

Graphical user interface, text, application, email

Description automatically generated

관리자는 소속 학과가 없다.

그래서 departmentId 필드 값이 null 이다.

## DepartmentMapper.java

### src/main/java/net/skhu/mapper/DepartmentMapper.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.Department;  @Mapper  public interface DepartmentMapper {  @Select("SELECT \* FROM department")  List<Department> findAll();  } |

## UserMapper.java

### src/main/java/net/skhu/mapper/UserMapper.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 | package net.skhu.mapper;  import java.util.List;  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 net.skhu.dto.User;  @Mapper  public interface UserMapper {  @Select("SELECT \* FROM user")  List<User> findAll();  @Select("SELECT \* FROM user WHERE userid = #{userid}")  User findByUserid(String userid);  @Insert("INSERT user (userid, password, name, email, enabled, userType, departmentId) " +  "VALUES (#{userid}, #{password}, #{name}, #{email}, #{enabled}, #{userType}, #{departmentId})")  @Options(useGeneratedKeys=true, keyProperty="id")  void insert(User user);  } |

## UserRegister.java

### src/main/java/net/skhu/model/UserRegister.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  23 | package net.skhu.model;  import javax.validation.constraints.Email;  import javax.validation.constraints.NotBlank;  import javax.validation.constraints.NotEmpty;  import javax.validation.constraints.Size;  import lombok.Data;  @Data  public class UserRegister {  @NotEmpty @NotBlank  @Size(min=3, max=12)  String userid;  @NotEmpty  @Size(min=6, max=12)  String passwd1;  String passwd2;  @NotEmpty @NotBlank  @Size(min=2, max=30)  String name;  @NotEmpty  @Email  String email;  int departmentId;  } |

회원 가입 입력 폼에서는 비밀번호를 두 번 입력해야 한다.

그래서 비밀 번호 멤버 변수가 두 개 이다.

DB 의 user 테이블에는 enabled, userType 필드가 있고,

그래서 User DTO 클래스에도 enabled, userType 멤버 변수가 있지만,

UserRegister 클래스에는 enabled, userType 멤버 변수가 없다.

사용자 활성화 여부와 사용자 권한이, 회원 가입 입력 폼에서 입력되지 않기 때문이다.

즉 입력 폼에서 입력하는 데이터의 구조와 (회원 가입 모델 클래스)

DB의 사용자 테이블의 구조는 서로 다르다.

그렇기 때문에 모델 클래스와 DTO 클래스를 따로 구현하였다.

user 테이블에서 조회한 데이터 -> User DTO 클래스

회원가입 폼에 입력한 데이터 -> UserRegister 모델 클래스

int departmentId;

select 태그에서 선택된 항목의 value 값이 여기에 대입된다.

따라서 여기에 대입되는 값이 null 이 될 수는 없다.

<option value="0">소속 학과를 선택하세요</option>

따라서 여기서는 int 변수를 선언해 대입해도 됨

## 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  20  21 | 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();  }  } |

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

그렇지만 컨트롤러가 DepartmentMapper의 메소드를 직접 호출하는 것은 바람직하지 않기 때문에

DepartmentService 클래스가 필요하다.

## UserService.java

### src/main/java/net/skhu/service/UserService.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 | package net.skhu.service;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import net.skhu.dto.User;  import net.skhu.mapper.UserMapper;  import net.skhu.model.UserRegister;  @Service  public class UserService {  @Autowired  public UserMapper userMapper;  public List<User> findAll() {  return userMapper.findAll();  }  public User findByUserid(String userid) {  return userMapper.findByUserid(userid);  }  public User createDto(UserRegister userRegister) {  User user = new User();  user.setUserid(userRegister.getUserid());  user.setPassword(userRegister.getPasswd1());  user.setName(userRegister.getName());  user.setEmail(userRegister.getEmail());  user.setEnabled(true);  user.setUserType("학생");  if (userRegister.getDepartmentId() != 0)  user.setDepartmentId(userRegister.getDepartmentId());  return user;  }  public void insert(UserRegister userRegister) {  User user = createDto(userRegister);  userMapper.insert(user);  }  } |

### createDto 메소드

UserRegister 모델 객체로부터 User DTO 객체를 생성해서 리턴한다.

회원 가입 입력 폼에 입력된 데이터를 user 테이블에 저장할 때 이 메소드가 호출된다.

## UserController.java

### src/main/java/net/skhu/controller/UserController.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 | package net.skhu.controller;  import javax.validation.Valid;  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 net.skhu.dto.User;  import net.skhu.model.UserRegister;  import net.skhu.service.DepartmentService;  import net.skhu.service.UserService;  @Controller  @RequestMapping("user")  public class UserController {  @Autowired UserService userService;  @Autowired DepartmentService departmentService;  @GetMapping("list")  public String list(Model model) {  model.addAttribute("users", userService.findAll());  return "user/list";  }  @GetMapping("register")  public String register(Model model) {  model.addAttribute("userRegister", new UserRegister());  model.addAttribute("departments", departmentService.findAll());  return "user/register";  }  @PostMapping("register")  public String register(Model model,  @Valid UserRegister userRegister, BindingResult bindingResult)  {  if (bindingResult.hasErrors()) {  model.addAttribute("departments", departmentService.findAll());  return "user/register";  }  User user = userService.findByUserid(userRegister.getUserid());  if (user != null) {  bindingResult.rejectValue("userid", null, "사용자 아이디가 중복됩니다.");  model.addAttribute("departments", departmentService.findAll());  return "user/register";  }  userService.insert(userRegister);  return "redirect:list";  }  } |

if (bindingResult.hasErrors()) {

model.addAttribute("departments", departmentService.findAll());

return "user/register";

}

userRegister 모델 객체에 채워진 데이터에 오류가 있다면, if 문이 true 가 된다. (spring form validation 기능)

회원 가입 입력 항목에 에러가 있기 때문에, 회원 가입 입력 폼이 다시 화면에 나타나야 한다.

그래서 액션 메소드는 "user/register" 뷰 이름을 리턴한다.

user/register.jsp 뷰 파일의 실행 결과가 웹브라우저 창에 나타나게 된다.

이때에도 학과 목록이 출력되어야 하기 때문에, 학과 목록이 모델에 들어있어야 한다.

User user = userService.findByUserid(userRegister.getUserid());

if (user != null) {

bindingResult.rejectValue("userid", null, "사용자 아이디가 중복됩니다.");

model.addAttribute("departments", departmentService.findAll());

return "user/register";

}

회원 가입 입력 폼에 입력된 사용자 아이디가 이미 어떤 사용자에 의해 사용되고 있다면,

위 if 문은 true 가 된다.

그렇다면, bindingResult 객체에 에러 메시지를 등록하고

액션 메소드는 "user/register" 뷰 이름을 리턴한다.

회원 가입 입력 폼이 다시 출력되게 된다.

userService.insert (userRegister);

return "redirect:list";

회원 가입 입력 폼에 입력된 데이터에 오류가 없으면,

그 데이터를 user 테이블에 저장한다.

그리고 사용자 목록 화면으로 리다이렉트한다.

## common.css

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

|  |  |
| --- | --- |
| 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 | 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, .btn-mini:active {  -ms-transform: translateY(2px);  -webkit-transform: translateY(2px);  transform: translateY(2px);  background: #ccc; }  .btn-mini { padding: 0.1em 0.3em; border: 1px solid gray;  border-radius: 0.4em; background: linear-gradient(#fff, #ddd);  text-decoration: none; color: black;  display: inline-block; font-size: 10pt; }  table.list { border-collapse: collapse; width: 100%; }  table.list.condensed { width: auto; }  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; } |

## user/list.jsp

### src/main/webapp/WEB-INF/views/user/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  47 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <c:url var="R" value="/" />  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link rel="stylesheet" href="${R}common.css">  <style>  h1 { display: inline-block; }  a { float: right; margin-top: 30px; }  td:nth-child(1), td:nth-child(5), td:nth-child(7) { text-align: center; }  </style>  </head>  <body>  <div class="container">  <h1>사용자목록</h1>  <a class="btn" href="register">회원등록</a>  <table class="list">  <thead>  <tr>  <th>id</th>  <th>사용자아이디</th>  <th>이름</th>  <th>이메일</th>  <th>활성화</th>  <th>사용자유형</th>  <th>학과ID</th>  </tr>  </thead>  <tbody>  <c:forEach var="user" items="${ users }">  <tr>  <td>${ user.id }</td>  <td>${ user.userid }</td>  <td>${ user.name }</td>  <td>${ user.email }</td>  <td>${ user.enabled }</td>  <td>${ user.userType }</td>  <td>${ user.departmentId }</td>  </tr>  </c:forEach>  </tbody>  </table>  </div>  </body>  </html> |

## user/register.jsp

### src/main/webapp/WEB-INF/views/user/register.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  47  48  49  50  51  52  53  54  55  56 | <%@ 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://www.springframework.org/tags/form" prefix="form"%>  <c:url var="R" value="/" />  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  <link rel="stylesheet" href="${R}common.css">  <style>  label { display: block; margin-top:10px; }  .error { color: red; display: block; }  button { margin-top: 20px; }  </style>  </head>  <body>  <div class="container">  <h1>회원 가입</h1>  <form:form method="post" modelAttribute="userRegister">  <div>  <label>아이디:</label>  <form:input path="userid" />  <form:errors path="userid" class="error" />  </div>  <div>  <label>비밀번호:</label>  <form:password path="passwd1" />  <form:errors path="passwd1" class="error" />  </div>  <div>  <label>비밀번호 학인:</label>  <form:password path="passwd2" />  <form:errors path="passwd2" class="error" />  </div>  <div>  <label>이름:</label>  <form:input path="name" />  <form:errors path="name" class="error" />  </div>  <div>  <label>이메일:</label>  <form:input path="email" />  <form:errors path="email" class="error" />  </div>  <div>  <label>학과:</label>  <form:select path="departmentId">  <form:option value="0">학과를 선택하세요</form:option>  <form:options itemValue="id" itemLabel="name" items="${ departments }" />  </form:select>  </div>  <button type="submit" class="btn">회원가입</button>  </form:form>  </div>  </body>  </html> |

<form:errors path="userid" class="error" />

userid 데이터 항목과 관련됭 에러 메시지가 자동으로 여기에 표시된다.

<form:errors path="passwd1" class="error" />

passwd1 데이터 항목과 관련됭 에러 메시지가 자동으로 여기에 표시된다.

<form:errors path="passwd2" class="error" />

passwd2 데이터 항목과 관련됭 에러 메시지가 자동으로 여기에 표시된다.

<form:errors path="name" class="error" />

name 데이터 항목과 관련됭 에러 메시지가 자동으로 여기에 표시된다.

## 실행

http://localhost:8088/user/register

Graphical user interface

Description automatically generated

아무 것도 입력하지 않고 저장 버튼을 클릭하면, 위와 같은 에러 화면을 볼 수 있다.

# 기타

## Unique Index

### user 테이블의 userid 필드

이 필드에 unique index가 만들어져 있다.

그렇기 때문에 중복된 userid 값의 레코드를 저장하려면 에러가 발생한다.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

## spring form validatoin annotation

All of the annotations used in the example are standard JSR annotations:

* @NotNull – validates that the annotated property value is not null
* @AssertTrue – validates that the annotated property value is true
* @Size – validates that the annotated property value has a size between the attributes min and max; can be applied to String, Collection, Map, and array properties
* @Min – vValidates that the annotated property has a value no smaller than the value attribute
* @Max – validates that the annotated property has a value no larger than the value attribute
* @Email – validates that the annotated property is a valid email address

Some annotations accept additional attributes, but the message attribute is common to all of them. This is the message that will usually be rendered when the value of the respective property fails validation.

Some additional annotations that can be found in the JSR are:

* @NotEmpty – validates that the property is not null or empty; can be applied to String, Collection, Map or Array values
* @NotBlank – can be applied only to text values and validated that the property is not null or whitespace
* @Positive and @PositiveOrZero – apply to numeric values and validate that they are strictly positive, or positive including 0
* @Negative and @NegativeOrZero – apply to numeric values and validate that they are strictly negative, or negative including 0
* @Past and @PastOrPresent – validate that a date value is in the past or the past including the present; can be applied to date types including those added in Java 8
* @Future and @FutureOrPresent – validates that a date value is in the future, or in the future including the present

<https://www.baeldung.com/javax-validation>