**[ 12 ] 폼 데이터 값 검증**

1. Validator(Interface)를 이용한 검증
   * 폼에서 전달 되는 데이터를 커맨드 객체에 담아 컨트롤 객체에 전달 한다고 하였습니다. 이때 커맨드 객체의 유효성 검사를 할 수 있습니다.
   * 참고로 javascript을 이용하는 것은 클라이언트에서 검사하는 방법이고, 지금 하는 Validator 인터페이스를 이용하는 방법은 서버에 검사하는 방법
   * 다음의 과정을 통해 검증한다.

다음단계로

(Service나

View)

클라이언트

<form>

컨트롤러

커맨드객체 검증결과 yes

Validator 객체

검증결과 No

* + 1. 클라이언트 측에서 form에 데이터를 넣고, submit에 의해 전송된다.
    2. 이때 데이터가 많을 경우 커맨트 객체에 담아서 컨트롤러로 온다.
    3. 컨트롤러에서 처리해서 뷰에 응답을 하기 전에
    4. 유효성 검증을 한다.
    5. 그러기 위해 Validator 인터페이스를 구현한 객체를 만들어 유효성 검사를 한다.
    6. 검증결과는 뷰 쪽으로 응답해준다.

(예제)

**index.jsp**

<body>

<jsp:forward page=*"inputForm.do"*></jsp:forward>

</body></html>

**MyController.java**

@Controller

**public** **class** MyController {

@RequestMapping(value = "inputForm", method = RequestMethod.***GET***)

**public** String memberInput() {

**return** "inputForm";

}

@RequestMapping(value = "input", method = RequestMethod.***POST***)

**public** String input(@ModelAttribute("student") Student student,

// 유효성 검증을 하고 나서 에러 결과를 담아주는 객체

BindingResult errors, Model model) {

String page = "result";

StudentValidator validator = **new** StudentValidator();

validator.validate(student, errors);

**if** (errors.hasErrors()) {

page = "inputForm";

**if**(errors.getFieldError("name")!=**null**) {

model.addAttribute("nameError", "name 입력 오류");

}

**if**(errors.getFieldError("id")!=**null**) {

model.addAttribute("idError", "id 입력 오류");

}

}

**return** page;

}

}

**inputForm.jsp**

<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<%@taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"*%>

<c:set var=*"conPath"* value=*"*${pageContext.request.contextPath }*"*/>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<form action=*"*${conPath }*/input.do"* method=*"post"*>

<p>이름 <input type=*"text"* name=*"name"* value=*"*${student.name }*"*></p>

<p>아이디 <input type=*"number"* name=*"id"* value=

*'*<c:if test="${student.id!=0 }">${student.id }</c:if>*'*></p>

<input type=*"submit"*>

</form>

<h6>${nameError }</h6>

<h6>${idError }</h6>

</body>

</html>

**Student.java**

**public** **class** Student {

**private** String name;

**private** **int** id;

**public** Student() {}

**public** Student(String name, **int** id) {

**this**.name = name; **this**.id = id;

}

**public** String getName() {**return** name;}

**public** **void** setName(String name) {**this**.name = name;}

**public** **int** getId() {**return** id;}

**public** **void** setId(**int** id) {**this**.id = id;}

}

**StudentValidator.java**

**import** org.springframework.validation.Errors;

**import** org.springframework.validation.Validator;

**public** **class** StudentValidator **implements** Validator {

@Override

**public** **boolean** supports(Class<?> arg0) {

// 검증할 객체의 클래스 타입 정보를 명시

**return** Student.**class**.isAssignableFrom(arg0);

}

@Override

**public** **void** validate(Object obj, Errors errors) {

System.***out***.println("유효성 검사한다");

Student student = (Student)obj;

String studentName = student.getName();

**int** stuentId = student.getId();

**if**(studentName==**null** || studentName.trim().isEmpty()) {

System.***out***.println("name이 null이거나 비었어요");

errors.rejectValue("name", "trouble");

}

**if**(stuentId==0) {

System.***out***.println("ID를 0으로 했거나 안 넣었거나");

errors.rejectValue("id", "trouble");

}

}

}

result.jsp

<body>

<h1>검증이 완료된 결과 페이지입니다</h1>

<h2>이름 : ${student.name }</h2>

<h2>이름 : ${student.id }</h2>

</body>

1. ValidationUtils 클래스(validate()메소드를 좀더 편리하게 사용 할 수 있도록 고안된 클래스)
   * 데이터 검증을 위해서 Validator 인터페이스의 validate() 메소드를 사용하였습니다.

String studentName = student.getName();

**if**(studentName==**null** || studentName.trim().isEmpty()){

System.***out***.println("studentName is null or empty");

errors.rejectValue("name", "trouble");

}

위를 아래와 같이 간략하게 바꿀 수 있다.

ValidationUrils.rejectIfEmptyOrWhitespace(errors, “name”,”trouble”);

**위의 예제에서 StudentValidator.java만 바꿈**

**public** **class** StudentValidator **implements** Validator {

@Override

**public** **boolean** supports(Class<?> arg0) {

// 검증할 객체의 클래스 타입 정보를 명시

**return** Student.**class**.isAssignableFrom(arg0);

}

@Override

**public** **void** validate(Object obj, Errors errors) {

System.***out***.println("유효성 검사한다");

Student student = (Student)obj;

String studentName = student.getName();

**int** stuentId = student.getId();

/\*if(studentName==null || studentName.trim().isEmpty()) {

System.out.println("name이 null이거나 비었어요");

errors.rejectValue("name", "trouble");

}\*/

ValidationUtils.*rejectIfEmptyOrWhitespace*(errors, "name", "name trouble");

**if**(stuentId==0) {

System.***out***.println("ID를 0으로 했거나 안 넣었거나");

errors.rejectValue("id", "id trouble");

}

}

}

1. @Valid와 @InitBinder
   * 데이터 검증을 위해서 Validator 인터페이스를 구현한 클래스를 만들고 validate()메소드를 직접 호출하여 사용하지 않고 스프링 프레임워크에서 호출하도록 어노테이션을 이용한다
   * 의존 추가하는 방법

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>4.2.0.Final</version>

</dependency>

@Controller

**public** **class** MyController {

@RequestMapping(value = "inputForm", method = RequestMethod.***GET***)

**public** String memberInput() {

**return** "inputForm";

}

@RequestMapping(value = "input", method = RequestMethod.***POST***)

**public** String input(@ModelAttribute("student") @Valid Student student,

// 유효성 검증을 하고 나서 에러 결과를 담아주는 객체

BindingResult errors, Model model) {

String page = "result";

//StudentValidator validator = new StudentValidator();

//validator.validate(student, errors);

**if** (errors.hasErrors()) {

page = "inputForm";

**if**(errors.getFieldError("name")!=**null**) {

model.addAttribute("nameError", "name 입력 오류");

}

**if**(errors.getFieldError("id")!=**null**) {

model.addAttribute("idError", "id 입력 오류");

}

}

**return** page;

}

@InitBinder

**protected** **void** initBnder(WebDataBinder binder) {

binder.setValidator(**new** StudentValidator());

}

}

* + 의존 추가하지 않는 방법

@Controller

**public** **class** MyController {

@RequestMapping(value = "inputForm", method = RequestMethod.***GET***)

**public** String memberInput() {

**return** "inputForm";

}

@RequestMapping(value = "input", method = RequestMethod.***POST***)

**public** String input(@ModelAttribute("student") @Validated Student student,

// 유효성 검증을 하고 나서 에러 결과를 담아주는 객체

BindingResult errors, Model model) {

String page = "result";

//StudentValidator validator = new StudentValidator();

//validator.validate(student, errors);

**if** (errors.hasErrors()) {

page = "inputForm";

**if**(errors.getFieldError("name")!=**null**) {

model.addAttribute("nameError", "name 입력 오류");

}

**if**(errors.getFieldError("id")!=**null**) {

model.addAttribute("idError", "id 입력 오류");

}

}

**return** page;

}

@InitBinder

**protected** **void** initBnder(WebDataBinder binder) {

binder.setValidator(**new** StudentValidator());

}

}