Spring Validation

Spring & JSR-303
& Apache validator

대덕인재개발원 : 최희연

❑ Spring validation

1. Spring validator 이용.

```
public class BoardValidator implements Validator {
        @Override
        public boolean supports(Class<?> clazz) {
                return BoardVO.class.isAssignableFrom(clazz);
        }
        @Override
        public void validate(Object target, Errors errors) {
                BoardVO board = (BoardVO)target;
                if(StringUtils.isEmpty(board.getBo_title())){
                        errors.rejectValue("bo_title", "errors.NotEmpty",
                                        new Object[]{"제목"}, "제목은 필수입력사항입니다.");
                }
                 ValidationUtils.rejectIfEmptyOrWhitespace(errors, "bo_writer", "NotEmpty",
                        new Object[]{"작성자"} "작성자는 필수입력사항입니다.");
        }
}
```

1) Validator 인터페이스의 구현체 작성 및 컨테이너에 빈등록.
2) 검증 대상 커맨드 객체가 사용되는 컨트롤러에 상위 Validator 주입.
3) @InitBinder 어노테이션을 이용하여 WebDataBinder에 바인딩 과정에서 사용될 검증객체로 설정.
4) 검증 대상 파라미터(커맨드 객체) 다음에 BindingResult 나 Errors 타입의 파라미터 선언.
5) Errors 의 hasErrors() 메소드를 통해 검증 결과 확인.

```
@Inject
BoardValidator validator;

@InitBinder("board")
public void binderSetting(WebDataBinder binder){
        binder.setValidator(validator);
}

@RequestMapping("boardInsert.do")
public String boardInsert(@ModelAttribute("board")BoardVO board, BindingResult errors,
                        Model model) throws Exception{
        if(errors.hasErrors()){
                return "board/boardForm";
        }
}
```

6) message bundle 작성 및 message resource 등록,
메시지 번들을 작성할 때 에러객체가 생성하는 메시지 코드의 규칙성을 참고하여 작성한다.
Errors/BindingResult는 에러코드에 기반하여 에러 메시지 코드를 생성하게 되는데, 다음과 같은 우선순위에 따라 메시지 코드를 생성한다.
- 에러코드.커맨드객체명.필드명
- 에러코드.필드명
- 에러코드.필드타입명
- 에러코드

```
#에러코드.커맨드객체.필드  or 에러코드.필드  or  에러코드.필드타입  or 에러코드
errors.NotEmpty.board.bo_title=제목 필수
errors.NotEmpty.bo_writer=작성자 필수
errors.Length.bo_passwd=최소3문자, 최대10문자
errors.Email.bo_email=이메일 주소 확인
errors.NotEmpty = {0} 는 필수 입력사항입니다.
errors.Length={0}는 최소 {1}문자,  최대 {2}문자로 입력하십시요.
```

```
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
                <property name="basenames" value="error_message"></property>
</bean>
```

7) 폼에서의 에러 메시지 출력, 뷰에서는 Spring 의 <form:errors path="커맨드객체명.필드명" />으로 검증 메시지 출력 : DefaultMessageCodesResolver에 의해 생성된 메시지 코드에 따라 messageSource로부터 메시지를 찾되 찾지 못한 경우 기본 메시지를 사용한다.

```
<spring:hasBindErrors name="board" />
<form:errors path="board.*" />
<form:form commandName="board" ..... >
        <form:input path="bo_title"> <!– FormattingConversionService 를 이용할수 있음-->
        <form:errors path="bo_title" cssStyle="color:red;" />
        .....
</form:form>
```

❑ 참고 : <mvc:annotation-driven /> 설정.

- handlerMapping, handlerAdapter, conversionService등 MVC 구조에 따른 웹서비스에 필요한 빈 설정정보를 자동 등록해줌(바인딩 절차 (타입변환->바인딩->검증)에 관계된 설정이 주).
- <mvc:annotation-driven /> 으로 등록되는 빈 종류. (3.1.2 ver.)

```
<bean id="handlerMapping"
        class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping“  p:order=“0”
/>
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"/>
<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean"/>
<bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"/>
<util:list id="messageConverters">
        <bean class="org.springframework.http.converter.ByteArrayHttpMessageConverter" />
        <bean class="org.springframework.http.converter.StringHttpMessageConverter"
                p:writeAcceptCharset="false" />
        <bean class="org.springframework.http.converter.ResourceHttpMessageConverter" />
        <bean class="org.springframework.http.converter.xml.SourceHttpMessageConverter" />
        <bean class="org.springframework.http.converter.xml.XmlAwareFormHttpMessageConverter" />
        <bean class="org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter" />
        <bean class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter" />
        <bean class="org.springframework.http.converter.feed.AtomFeedHttpMessageConverter" />
        <bean class="org.springframework.http.converter.feed.RssChannelHttpMessageConverter" />
</util:list>
<bean id="handlerAdapter"
                class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
        <property name="webBindingInitializer">
                <bean class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer"
                        p:validator-ref="validator" p:conversionService-ref="conversionService"/>
        </property>
        <property name="messageConverters" ref="messageConverters" />
        <!-- <property name="customArgumentResolvers"> </property> -->
        <!-- <property name="customReturnValueHandlers"> </property> -->
</bean>
<bean class="org.springframework.web.servlet.handler.MappedInterceptor">
        <constructor-arg index="0">
                <null />
        </constructor-arg>
        <constructor-arg index="1">
                <bean class="org.springframework.web.servlet.handler.ConversionServiceExposingInterceptor">
                        <constructor-arg index="0" ref="conversionService" />
                </bean>
        </constructor-arg>
</bean>
<bean class="org.springframework.web.servlet.mvc.method.annotation.ExceptionHandlerExceptionResolver">
        <property name="messageConverters" ref="messageConverters" />
        <property name="order" value="0" />
</bean>
<bean class="org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver"
                p:order="1"/>
<bean class="org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver"
                p:order="2"/>
```

2. JSR-303 방식의 객체 검증

스프링은 객체 검증을 수행할 때 다음과 같은 순서로 객체 검증자를 검색하게 된다. 먼저 특정 커맨드 객체 하나를 검증할 수 있는 validator를 찾고, 없을 경우 글로벌 validator를 찾게 되는데, 특별한 글로벌 validator가 등록되지 않은 경우, JSR-303 어노테이션을 지원하는 validator를 글로벌 validator로 등록한다.

<mvc:annotation-driven >태그의 설정에 따라 명시적인 글로벌 validator가 등록되지 않는다면, JSR-303 어노테이션을 지원하는 LocalValidatorFactoryBean 이 글로벌 validator로 등록되는 것이다.

1) 도메인 레이어에 직접 검증 규칙을 적용함. : Javax.validation 과 Hibernate Validator 를 사용.

```
public class MemberVO {
	@NotEmpty(message="아이디 필수")
	private String mem_id;
	@Length(min=4, message="암호 4자리 이상")
	private String mem_pass;
	@NotEmpty(message="이름 필수", groups=Insert.class)
	private String mem_name;
	@Length(min=6, max=6, message="주민번호 앞자리 6자리 고정.")
	private String mem_regno1;
	@Length(min=7, max=7, message="주민번호 뒷자리 7자리 고정.")
	private String mem_regno2;
	private String mem_bir;
	private String mem_zip;
	private String mem_add1;
	private String mem_add2;
	private String mem_hometel;
	// 서로 다른 검증조건을 써야하는 경우에 groups를 활용.
	@NotEmpty(message="회사번호 필수", groups=Insert.class)
	private String mem_comtel;
	@NotEmpty(message="휴대폰 번호 필수", groups=Update.class)
	private String mem_hp;
}
```

2) 검증 대상 객체의 앞에 @Valid 어노테이션 선언.

3) 컨트롤러레이어의 검증 대상 파라미터(커맨드 객체) 다음에 BindingResult 나 Errors 타입의 파라미터 선언.

4) Errors 의 hasErrors() 메소드를 통해 검증 결과 확인.

5) 뷰에서는 Spring 의 <form:errors path="커맨드객체명.필드명" />으로 검증 메시지 출력.

```
@RequestMapping("memberInsert.do")
public String insert(@Valid @ModelAttribute("member")MemberVO member, BindingResult errors,
		Model model)
		throws Exception {
		if(errors.hasErrors()){
			return "member/memberForm";
		}
}
```

6) <mvc:annotation-driven /> 설정에 의해 LocalValidatorFactoryBean 등록된 경우, @Valid 어노테이션을 비롯한 JSR-303 방식의 객체 검증을 활용할 수 있음.

- Spring 3.0까지 DefaultAnnotaionHandlerMapping 을 사용하는 경우, ConfigurableWebBindingInitialize 를 핸들러 매핑에 DI 한 후 validator 구현체를 webBindingInitialize 에 DI 해줘야 Validator 구현체를 사용.

```
<bean id="handlerAdapter"
        class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
    <property name="webBindingInitializer">
        <bean class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer">
            <property name="validator" ref="validator" />
        </bean>
    </property>
</bean>
<bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean" >
    <property name="validationMessageSource" ref="messageSource" />
</bean>
```

- JSR-303 검증시, 커맨드에 따라 constraint 를 다르게 적용해야 하는 경우, 비어있는 interface 를 만들고 이를 기준으로 그룹핑.

```
public interface Insert {
}

public interface Update{
}

//.....
// 서로 다른 검증조건을 써야하는 경우에 groups를 활용.
@NotEmpty(message="회사번호 필수", groups=Insert.class)
private String mem_comtel;
@NotEmpty(message="휴대폰 번호 필수", groups=Update.class)
```

7) 컨테이너에 등록된 LocalValidatorFactoryBean으로 Validator 구현체를 주입받은 후, validate 메소드를 호출시, 그룹 interface 의 Class 를 인자로 넘겨, Constraint 적용 그룹을 지정할수 있으며, 일반적으로 등록과 수정시의 constraint 적용 기준이 다른 경우에 이 방식을 사용함. Validation hint 는 여러 개를 넘길수 있고, default group 과 특정 그룹 검증을 같이 하고 싶은 경우, Javax.validator.groups.Default.class 를 사용.
(3.1 버전부터 @Validated({Insert.class, Default.class}) 를 기준으로 group constranits 적용 가능)

```
@Inject
LocalValidatorFactoryBean validator;

@RequestMapping(value="memberUpdate.do", method=RequestMethod.GET)
public ModelAndView memberUpdate(@ModelAttribute("member") MemberVO member,
                                 BindingResult error,
                                 ModelMap model, HttpServletResponse response
        ) throws IOException{

        String viewName = null;
        validator.validate(member, error, Update.class, Default.class);

        if(error.hasErrors()){
                viewName= "member/memberView";
        }else{
//....
}
```

8) View 에서 랜더링할 에러 메시지 작성. Spring 은 DefaultMessageCodeResolver 에 따라 아래 예제의 주석과 같은 방식으로 랜더링할 에러 메시지를 결정한다.

```
#에러코드.커맨드객체.필드  or 에러코드.필드  or  에러코드.필드타입  or 에러코드
errors.NotEmpty.board.bo_title=제목 필수
errors.NotEmpty.bo_writer=작성자 필수
errors.Length.bo_passwd=최소3문자, 최대10문자
errors.Email.bo_email=이메일 주소 확인
errors.NotEmpty = {0} 는 필수 입력사항입니다.
errors.Length={0}는 최소 {1}문자,  최대 {2}문자로 입력하십시요.
```

9) 뷰에서는 Spring 의 <form:errors path="커맨드객체명.필드명" />으로 검증 메시지 출력.

❑ Hibernate Validator Annotation

Annotation	Supported type	Description
@AssertFalse	Boolean	대상의 값이 false 인지 체크
@AssertTrue	Boolean	대상의 값이 true 인지 체크
@DecimalMax	BigDecimal, BigInteger, String, number type	최대값 지정.
@DecimalMin	BigDecimal, BigInteger, String, number type	최소값 지정
@Digits(integer=, fraction=)	BigDecimal, BigInteger, String, number type	정수부와 실수부 지정 체크
@Email	String	이메일 주소 검증
@Future	java.util.Date, java.util.Calendar	현재를 기준으로 미래의 date인지 검증
@Length(min=,max=)	String	문자열의 길이 체크
@Max	BigDecimal, BigInteger, String, number type	대상이 가질수있는 최대 length
@Min	BigDecimal, BigInteger, String, number type	대상이 가질수 있는 최소 length
@NotNull	Object	Not null 체크
@NotEmpty	String	Null 과 공백을 동시에 체크
@Null	Object	Null 인지 체크
@Past	java.util.Date, java.util.Calendar	현재를 기준으로 과거의 date인지 검증.
@Pattern(regex= ,Flag=)	String	정규표현식을 통한 검증
@Range(min= ,max=)	BigDecimal, BigInteger, String, number type	대상체의 최소~최대 범위를 지정.
@Size(min= ,Max	String, Collection	대상체의 size 검증
@Valid	Object	어노테이션 기준 검증을 수행

Spring Validator Apache commons-validator 통합

1. apache.org 에서 commons-validator.jar 다운.
2. **ebr.springsource.com** 에서 spring Modules-Validation Language 0.9버전 다운.
   주의 : 의존관계에 따라 commons-beanutils, commons-digester, commons-logging 필요
   참고 : SpringSource의 Enterprise Bundle Repository의 의존성을 사용하려면, maven 설정에 repository를 추가.
   http://ebr.springsource.com/repository/app/faq -> How do I configure a Maven build to work with the repository? 참고
3. DefaultValidatorFactory 를 빈으로 등록. – validationConfigLocations에 정의된 validation-rule을 기반으로 commons validator 들의 인스턴스 생성한다.
4. DefaultBeanValidator 를 빈으로 등록. – org.springframework.validation.Validator의 구현체로, DefaultValidatorFactory를 통해 구해진 Commons Validator의 인스턴스를 이용해 검증을 수행하게 된다.

```
<bean id="validatorFactory"  class="org.springmodules.validation.commons.DefaultValidatorFactory">
        <property name="validationConfigLocations">
                <list>
                        <value>/WEB-INF/spring/validator/validator-rules.xml</value>
                        <value>/WEB-INF/spring/validator/validator.xml</value>
                </list>
        </property>
        </bean>
<bean id="beanValidator" class="org.springmodules.validation.commons.DefaultBeanValidator"
                                        p:validatorFactory-ref="validatorFactory”
/>
```

5. 범용 검증 룰을 가진 validator-rules.xml 작성.

```
<!DOCTYPE form-validation PUBLIC
       "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.0//EN"
       "http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">
<form-validation>
  <global>
    <validator name="required"
          classname="org.springmodules.validation.commons.FieldChecks"
          method="validateRequired"
          methodParams="java.lang.Object,
                org.apache.commons.validator.ValidatorAction,
                org.apache.commons.validator.Field,
                org.springframework.validation.Errors"
                msg="errors.required">
      <javascript><![CDATA[
        ….
        ]]>
      </javascript>
    </validator>
   ……
 <global>
<form-validation>
```

• Validator 속성의 종류

name	validation rule(required,mask,integer,email...)
classname	validation check를 수행하는 클래스명 (org.springmodules.validation.commons.FieldChecks)
method	validation check를 수행하는 클래스의 메소드명 (validateRequired,validateMask...)
methodParams	validation check를 수행하는 클래스의 메소드의 파라미터
msg	에러 메시지 key
javascript	client-side validation을 위한 자바스크립트 코드

• Spring modules(0.9 기준) 에서 제공하는 validation rule의 종류

Name (validation rule)	FieldCheck 클래스	FieldCheck 클래스의 메소드	기능
required	org.springmodules.validation.commons.FieldChecks	validateRequired	필수값 체크
minlength	org.springmodules.validation.commons.FieldChecks	validateMinLength	최소 길이 체크
maxlength	org.springmodules.validation.commons.FieldChecks	validateMaxLength	최대 길이 체크
mask	org.springmodules.validation.commons.FieldChecks	validateMask	정규식 체크
byte	org.springmodules.validation.commons.FieldChecks	validateByte	Byte형 체크
short	org.springmodules.validation.commons.FieldChecks	validateShort	Short형 체크
integer	org.springmodules.validation.commons.FieldChecks	validateInteger	Integer형 체크
long	org.springmodules.validation.commons.FieldChecks	validateLong	Long형 체크
float	org.springmodules.validation.commons.FieldChecks	validateFloat	Float형 체크
double	org.springmodules.validation.commons.FieldChecks	validateDouble	Double형 체크
date	org.springmodules.validation.commons.FieldChecks	validateDate	Date형 체크
range	org.springmodules.validation.commons.FieldChecks	validateIntRange	범위 체크
intRange	org.springmodules.validation.commons.FieldChecks	validateIntRange	int형 범위 체크
floatRange	org.springmodules.validation.commons.FieldChecks	validateFloatRange	Float형 범위체크
creditCard	org.springmodules.validation.commons.FieldChecks	validateCreditCard	신용카드번호체크
email	org.springmodules.validation.commons.FieldChecks	validateEmail	이메일체크

org.springmodules.validation.commons.FieldChecks 클래스에 존재하는 여러 validation check 메소드들은 실제 검증을 Commons Validator 에게 위임하고 있으며, commons-validator 1.3 버전부터 새로 추가된 validation rule은 FieldChecks 클래스에 검증 메소드가 존재하지 않는다. 따라서, 이런 경우 새로운 FieldCheck 클래스를 생성하고 그 클래스와 검증 메소드를 포함한 custom validation rule을 validator-rules.xml 에 추가해줘야 한다.

6. 폼 데이터가 바인딩될 커맨드 객체 검증에 대한 개별 설정을 지닌 validator.xml 파일의 작성

```
<!DOCTYPE form-validation PUBLIC
   "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1//EN"
   "http://jakarta.apache.org/commons/dtds/validator_1_1.dtd">
<form-validation>
        <formset>
                <form name="boardVO">
                        <field property="bo_title" depends="required">
                                <arg0 key="board.bo_title" />
                        </field>
                        <field property="bo_writer" depends="required">
                                <arg0 key="board.bo_writer" />
                        </field>
                        <field property="bo_passwd" depends="required, minlength, maxlength">
                                <arg0 key="board.bo_passwd" />
                                <arg1 name="maxlength"  key="${var:maxlength}" resource="false"/>
                                <arg1 name="minlength"  key="${var:minlength}" resource="false"/>
                                <var>
                                        <var-name>minlength</var-name>
                                        <var-value>8</var-value>
                                </var>
                                <var>
                                        <var-name>maxlength</var-name>
                                        <var-value>12</var-value>
                                </var>
                        </field>
                        <field property="bo_content" depends="required, minlength">
                                <arg0 key="board.bo_content" />
                                <arg1 name="minlength"  key="${var:minlength}" />
                                <var>
                                        <var-name>minlength</var-name>
                                        <var-value>10</var-value>
                                </var>
                        </field>
                </form>
        </formset>
</form-validation>
```

- 서버사이드 검증의 경우, form name 과 field property 는 validation 할 폼 클래스의 이름, 필드와 각각 매핑된다 (camel 표기에 따름).
- 클라이언트사이드 검증의 경우, form name 은 뷰에서 설정한 validator:javascript 태그의 formName 속성값과 매핑되고, field property는 각 폼필드(input tag)의 name 과 매핑된다.
- depends 는 해당 필드에 적용할 검증 룰(validator-rules.xml 의 rule name)을 의미한다.
- arg 는 에러 메시지에서 사용할 메시지 아규먼트를 의미.

- 상기 설정의 의미

BoardVO 클래스의 각 프로퍼티에 대한 검증 수행시,

1) bo_title, bo_writer 의 경우 필수 값 체크,

2) bo_passwd의 경우 필수값 체크 및 최소(8), 최대 길이(12) 검증.

3) bo_content 의 경우 필수값 체크 및 최소 길이 검증을 수행하게 된다.

7. 컨트롤러단에서의 서버사이드 검증.

```
@Inject
DefaultBeanValidator beanValidator;

@RequestMapping("boardInsert.do")
public String boardInsert(@ModelAttribute("board")BoardVO board, BindingResult errors,
                                        Model model) throws Exception{

                beanValidator.validate(board, errors);

                if(errors.hasErrors()){
                                        return "board/boardForm";
                }
//.....
}
```

8. 메시지 번들의 작성

```
board.bo_title=제목
board.bo_writer=작성자
board.bo_content=내용
board.bo_passwd=비밀번호

## validator errors message –
errors.required={0}는 필수입력사항입니다.
errors.minlength={0}은 {1}자 이상 입력해야 합니다.
errors.maxlength={0}은 {1}자  이상 입력할수 없습니다.
#.....
```

9. 클라이언트사이드 검증

1) 클라이언트 사이드 검증을 위해 validator-rules.xml 에 등록된 javascript 코드를 응답데이터로 생성해줄 서버 모듈 작성.

```
<%-- 자바스크립트 프록시 모듈로 사용할 validatorScript.jsp  --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%@taglib uri="http://www.springmodules.org/tags/commons-validator" prefix="validator" %>
<validator:javascript dynamicJavascript="false" staticJavascript="true" />
```

```
@RequestMapping("/validate.do")
public String validateScript(){
        return "validateScript.jsp";
}
```

2) form view(boardForm.jsp) 작성

- commons-validator taglib 등록

```
<%@taglib uri="http://www.springmodules.org/tags/commons-validator" prefix="validator" %>
```

- validaotrScript 에서 생성된 javascript 코드를 로딩하기 위한 script 태그 작성.

```
<SCRIPT type="text/javascript" src="<c:url value='/validator.do'/>"></SCRIPT>
```

- 로딩된 검증 함수를 사용해 폼 데이터 검증을 수행

```
<validator:javascript formName="BoardVO" staticJavascript="false" xhtml="true" cdata="false"/>
```

  * formName 은 validator.xml 에 등록된 forma name(검증 대상 클래스명)

- form submit 전 검증을 수행하도록 submit 이벤트 핸들러 작성.

```
<form onsubmit="return validateBoardVO(this)">
```

  * 검증 함수명은 validate+검증대상클래스명(validateBoardVO)

```
<%@taglib uri="http://www.springmodules.org/tags/commons-validator" prefix="validator" %>
<SCRIPT type="text/javascript" src="<c:url value='/validator.do'/>"></SCRIPT>
<validator:javascript formName="BoardVO" staticJavascript="false" xhtml="true" cdata="false"/>
<script type="text/javascript">
      function submitContents(form){
           if(!validateBoardVO(form)){
                return false;
           } else{
                return true;
           }
      }
</script>
<form:form action="${pageContext.request.contextPath }/board/boardInsert.do"
    method="post" enctype="multipart/form-data"
    onsubmit="return submitContents(this)" commandName="board">
    <INPUT type="hidden" name="bo_no" value="${board.bo_no }"/>
    <table>
        <tr>
            <th>제목</th>
            <td>
                <INPUT type="text" name="bo_title" value="${board.bo_title }"/>
                <form:errors path="bo_title" cssClass="fieldErrors" />
            </td>
        </tr>
        <tr>
            <th>작성자</th>
            <td>
                <INPUT type="text" name="bo_writer" value="${board.bo_writer }"/>
                <form:errors path="bo_writer" cssStyle="color: red;"></form:errors>
            </td>
        </tr>
        <tr>
            <th>비밀번호</th>
            <td>
                <INPUT type="password" name="bo_passwd" value="${board.bo_passwd }" size="12"/>
                <form:errors path="bo_passwd" cssClass="fieldErrors"></form:errors>
            </td>
        </tr>
        <tr>
            <th>내용</th>
            <td>
                <TEXTAREA id="ir1" rows="5" cols="40"
                            name="bo_content" >${board.bo_content }</TEXTAREA>
                <form:errors path="bo_content" cssClass="fieldErrors"></form:errors>
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <INPUT type="submit" value="등록하기" />
            </td>
        </tr>
    </table>
</form:form>
```

❖ Validation rule 추가(validator-rules.xml).

어플리케이션에 특화된 검증 룰을 추가해야 하는 경우 범용 검증 룰을 가진 validator-rules.xml 에 검증룰을 추가해야 한다. Validator 엘리먼트가 가진 중 classname 에 실제 검증을 담당할 validation class 를 설정하게 되는데, springmodules 가 가진 FieldChecks 만으로 해결되지 않기 때문에 특화된 검증기능을 담당할 validation class를 작성하는 것부터 시작하게 된다.

1. Custom FieldChecks 작성.

```
public class RegNoChecks extends FieldChecks{

	/**
	 * 주민등록번호 검증 룰에 등록할 메소드
	 */
	public boolean validateRegno(Object bean, ValidatorAction va, Field field, Errors errors){
		String regno = FieldChecks.extractValue(bean, field).trim();
		if(validate(regno)){
			return true;
		}else{
			FieldChecks.rejectValue(errors, field, va);
			return false;
		}
	}

	/**
	 * 주민등록번호 검증 메소드
	 */
	private boolean validate(String regno){
		String regex = "\\d{6}[1-4]\\d{6}";
		String dateStr = null;
		boolean result = false;
		if(regno.matches(regex)){
			String temp = (regno.charAt(6)==0 || regno.charAt(6)==1 ? "19" : "20");
			dateStr = temp+regno.substring(0, 6);
			SimpleDateFormat format = new SimpleDateFormat("yyyyMMdd");
			try {
				Date date = format.parse(dateStr);
				if(format.format(date).equals(dateStr)){
					char[] charArray = regno.toCharArray();
					long sum = 0;
					int[] intArray = {2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4, 5};
					for(int i=0; i<charArray.length-1; i++){
						sum += Integer.parseInt(""+charArray[i]) * intArray[i];
					}
					int checkDigit =((int) (11 - sum % 11)) % 10;
					if(checkDigit == Integer.parseInt(charArray[12]+"")){
						result = true;
					}
				}
			} catch (ParseException e) { }
		}
		return result;
	}
}
```

2. validator-rules.xml 에 검증 룰 추가

```
<validator name="regno"
            classname="kr.or.ddit.web.validator.RegNoChecks"
            method="validateRegno"
            methodParams="java.lang.Object,
                    org.apache.commons.validator.ValidatorAction,
                    org.apache.commons.validator.Field,
                    org.springframework.validation.Errors"
            depends=""
            msg="errors.regno">
        <javascript> <![CDATA[
            function validateRegno(form) {
                    var bValid = true;
                    var focusField = null;
                    var i = 0;
                    var fields = new Array();
                    oRegno = new regno();
                    for (x in oRegno) {
                     var field = form[oRegno[x][0]];
                        if ((field.type == 'text' ||
                            field.type == 'hidden' ||
                            field.type == 'textarea')) {
                                var value = field.value.trim();
                                if (value.length==0||!checkRegno(value)) {
                                        if (i == 0) {
                                            focusField = field;
                                        }
                                        fields[i++] = oRegno[x][1];
                                        bValid = false;
                                }
                          }
                    }
                    if (fields.length > 0) {
                        focusField.focus();
                        alert(fields.join('\n'));
                    }
                    return bValid;
            }
```

2. validator-rules.xml 에 검증 룰 추가(계속)

```
        function checkRegno(regno){
                var regex = /^\d{6}[1-4]\d{6}$/;
                var result = false;
                if(regex.test(regno)){
                        var birthYear = (regno.charAt(6)<=2 ? "19" : "20")
                                        +regno.substr(0, 2);
                        var birthMonth = regno.substr(2, 2)-1;
                        var birthDate = regno.substr(4, 2);
                        var birth = new Date(birthYear, birthMonth, birthDate);
                        if(birth.getFullYear()==birthYear &&
                                birth.getMonth()==birthMonth &&
                                birth.getDate()==birthDate ){
                                var intArray = [2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4, 5];
                                var sum = 0;
                                for(var idx in intArray){
                                        sum += parseInt(regno.charAt(idx))*intArray[idx];
                                }
                                var checkDigit = (11-(sum%11))%10;
                                if(checkDigit==regno.charAt(12)){
                                        result = true;
                                }
                        }
                }

                return result;
        }
        ]]>
        </javascript>
</validator>
```

3. validator.xml 에 추가한 검증룰에 따라 검증할 대상 설정.

```
<formset>
        <form name="MemberVO">
                <field property="mem.regno" depends="required, regno">
                        <arg0 key="member.mem_regno" />
                </field>
        </form>
</formset>
```

4. 에러 메시지 추가

```
## validator errors message –
errors.regno=유효하지 않은 주민등록 번호 입니다.
#.....
```

5. form view 에 클라이언트 검증을 위한 코드 추가

```
<%@taglib uri="http://www.springmodules.org/tags/commons-validator" prefix="validator" %>


<SCRIPT type="text/javascript" src="${pageContext.request.contextPath }/validator.do"></SCRIPT>

<validator:javascript staticJavascript="false" formName="MemberVO" xhtml="true" cdata="false" />

<form id="memForm" action="${pageContext.request.contextPath }/member/memberInsert.do"
        onsubmit="return validateMemberVO(this)"
>
```