**SPRING MVC USING BEAN & Hibernate ANNONATIONS**

**http://www.raistudies.com/spring/spring-mvc/form-validation-spring-mvc-3-hibernate-validator-jsr-303/**

[Previous example](http://www.raistudies.com/spring/spring-mvc/form-validation-spring-mvc-3-xml-config/) shows how form validation is done in Spring MVC 3 using validator class. In this tutorial we will learn **how Spring MVC uses Hibernate Bean Validator framework to apply form validation**.

We will modify our previous example on [Form Validation in Spring MVC 3 using XML Configuration](http://www.raistudies.com/spring/spring-mvc/form-validation-spring-mvc-3-xml-config/) to use Hibernate Bean Validator Framework (JSR 303 implementation).

Hibernate Bean Validator Framework uses annotations to apply validation rules on properties of a bean. To use Hibernate Bean Validator Framework with Spring MVC 3 we need to add some jar files related to bean validator and it’s dependencies. Following are the jar files we have to add :

* hibernate-validator-4.0.2.GA.jar
* log4j-1.2.16.jar
* slf4j-api-1.5.10.jar
* slf4j-log4j12-1.5.10.jar

These jars are included in the war file of this example. You can download the war file to get the jars.

Following are some of the annotations of Hibernate Validator that can be used to validate bean properties :

1. **@NotNull** : Property can not be null. Applicable to class instances only, not to primitive types.
2. **@NotEmpty** : Property can not be null or empty.
3. **@Null** : Property should be null. Applicable to class instances only, not to primitive types.
4. **@Length** : Used to define min and max length of a string.
5. **@NumberFormat** : Can define a particular number format or style to String.
6. **@Min** : Used to apply constraint on minimum value of number.
7. **@Max** : Used to apply constraint on maximum value of number.
8. **@Valid** : Applicable to an class instance. Class should also use Hibernate Validator constraint and the instance will be valid if all constraint defined in class are fulfilled.
9. **@AssertTrue** : Applicable to boolean property only. The property must be true.
10. **@AssertFalse** : Checks that the annotated elements are false;
11. **@CreditCardNumber** : Used to check if a number is a credit card number or not.
12. **@DecimalMax** : Applicable to BigDecimal, BigInteger,String,  byte, short,  int, long and respective wrapper classes. For min value check.
13. **@DecimalMin :** Applicable to BigDecimal, BigInteger,String,  byte, short,  int, long and respective wrapper classes. For max value check.
14. **@Future** : Applicable to Date or Calender instance. The time must be in future means after today.
15. **@Past** : Applicable to Date or Calender instance. The time must be in past means before today.

Now, lets start with example to implement the validation using hibernate bean validation in spring mvc 3. We will follow following steps to modify our previous example of form validation :

1. First of all we will modify our command class (**RegistrationBean.java**) to include hibernate bean validation annotations.
2. Then we will change our controller (**RegistrationController.java**) to make it annotation driven.
3. Will make changes in spring mvc 3 configuration file (**app-config.xml**) to make configuration annotation driven.
4. Then we will make changes to the existing validator class (**RegistrationValidator.java**) to handle only class level validation.

**RegistrationBean.java**

Our command class RegistrationBean.java will be modified as follows to make the support of hibernate bean validation :

|  |  |  |
| --- | --- | --- |
| 1 | package com.raistudies.domain; | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | import org.hibernate.validator.constraints.Length; |
| 4 | import org.hibernate.validator.constraints.NotEmpty; | |

|  |  |
| --- | --- |
| 5 | import org.springframework.format.annotation.NumberFormat; |
| 6 | import org.springframework.format.annotation.NumberFormat.Style; | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | public class RegistrationBean { | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | @NotEmpty(message="Name field is mandatory.") | |

|  |  |  |
| --- | --- | --- |
| 11 | private String name = null; | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | @NotEmpty(message="Username field is mandatory.") | |
| 14 | private String username = null; |

|  |  |
| --- | --- |
| 15 |  |
| 16 | @NotEmpty(message="Email field is mandatory.") | |

|  |  |  |
| --- | --- | --- |
| 17 | private String email = null; | |
| 18 |  |

|  |  |
| --- | --- |
| 19 | @Length(max=10,min=10,message="Phone number is not valid. Should be of length 10.") |
| 20 | @NotEmpty(message="Phone field is mendatory.") @NumberFormat(style= Style.NUMBER) |

|  |  |  |
| --- | --- | --- |
| 21 | private String phone; | |
| 22 |  |

|  |  |  |
| --- | --- | --- |
| 23 | @NotEmpty(message="Password should not be empty.") | |
| 24 | private String password = null; |

|  |  |
| --- | --- |
| 25 |  |
| 26 | @NotEmpty(message="Retype Password should not be empty.") | |

|  |  |  |
| --- | --- | --- |
| 27 | private String rePassword = null; | |
| 28 |  |

|  |  |  |
| --- | --- | --- |
| 29 | // Getter and setter are omitted to make the code short. | |
| 30 | } |

You can apply our custom error message using “message” attributes in annotations. Other attributes are required as per the annotations.

**RegistrationController.java**

We have to change our controller class to work with annotation configurations.

|  |  |  |
| --- | --- | --- |
| 1 | package com.raistudies.controllers; | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | import javax.validation.Valid; | |
| 4 |  |

|  |  |  |
| --- | --- | --- |
| 5 | import org.springframework.beans.factory.annotation.Autowired; | |
| 6 | import org.springframework.stereotype.Controller; |

|  |  |
| --- | --- |
| 7 | import org.springframework.ui.ModelMap; |
| 8 | import org.springframework.validation.BindingResult; | |

|  |  |  |
| --- | --- | --- |
| 9 | import org.springframework.web.bind.annotation.ModelAttribute; | |
| 10 | | import org.springframework.web.bind.annotation.RequestMapping; | |

|  |  |  |
| --- | --- | --- |
| 11 | import org.springframework.web.bind.annotation.RequestMethod; | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | import com.raistudies.domain.RegistrationBean; |
| 14 | import com.raistudies.validator.RegistrationValidator; | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | @Controller | |

|  |  |  |
| --- | --- | --- |
| 17 | @RequestMapping(value="/registration.htm") | |
| 18 | public class RegistrationController { |

|  |  |  |
| --- | --- | --- |
| 19 | RegistrationValidator validator = null; | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | public RegistrationValidator getValidator() { | |
| 22 | return validator; |

|  |  |  |
| --- | --- | --- |
| 23 | } | |
| 24 |  |

|  |  |
| --- | --- |
| 25 | @Autowired |
| 26 | public void setValidator(RegistrationValidator validator) { | |

|  |  |  |
| --- | --- | --- |
| 27 | this.validator = validator; | |
| 28 | } |

|  |  |
| --- | --- |
| 29 |  |
| 30 | @RequestMapping(method=RequestMethod.GET) | |

|  |  |
| --- | --- |
| 31 | public String showForm(ModelMap model){ |
| 32 | RegistrationBean userRegis = new RegistrationBean(); | |

|  |  |  |
| --- | --- | --- |
| 33 | model.addAttribute("Registration", userRegis); | |
| 34 | return "registration"; |

|  |  |  |
| --- | --- | --- |
| 35 | } | |
| 36 |  |

|  |  |
| --- | --- |
| 37 | @RequestMapping(method=RequestMethod.POST) |
| 38 | public String processForm(@ModelAttribute(value="Registration") @Valid RegistrationBean userRegis,BindingResult result){ | |

|  |  |  |
| --- | --- | --- |
| 39 | validator.validate(userRegis, result); | |
| 40 | if(result.hasErrors()){ |

|  |  |  |
| --- | --- | --- |
| 41 | return "registration"; | |
| 42 | }else{ |

|  |  |  |
| --- | --- | --- |
| 43 | return "success"; | |
| 44 | } |

|  |  |  |
| --- | --- | --- |
| 45 | } | |
| 46 | } |

You can find some new things in our controller class. Here is the explanation :

* **@Valid** : the new annotation @Valid has been associated with the ModelAttribute or command instance. It is the part of the Hibernate Bean Validation. It will indicate to Spring MVC that the RegistrationBean instance should be a valid instance according to Hibernate Bean Validation Framework and if any error are there then put the errors in BindingResult instance.
* **@Autowired** : We have also used our previous validator class with containing only one class level validation and make our validation class a spring bean. @Autowired annotation tells Spring IoC to inject the instance of our valitator bean to our controller. It is mainly a part of Spring IoC and will take detail description in our next tutorial.

**app-config.xml**

|  |  |
| --- | --- |
| 1 | <?xml version="1.0" encoding="UTF-8"?> |
| 2 | <beans xmlns="http://www.springframework.org/schema/beans" | |

|  |  |
| --- | --- |
| 3 | xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" |
| 4 | xmlns:context="http://www.springframework.org/schema/context" | |

|  |  |  |
| --- | --- | --- |
| 5 | xmlns:mvc="http://www.springframework.org/schema/mvc" | |
| 6 | xsi:schemaLocation=" |

|  |  |  |
| --- | --- | --- |
| 7 | http://www.springframework.org/schema/beans    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd | |
| 8 | | http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd"> | | |
| 10 | |  |

|  |  |  |
| --- | --- | --- |
| 11 | | <!-- Application Message Bundle --> |
| 12 | <bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource"> | | |

|  |  |  |
| --- | --- | --- |
| 13 | <property name="basename" value="/WEB-INF/messages" /> | |
| 14 | <property name="cacheSeconds" value="3000" /> |

|  |  |  |
| --- | --- | --- |
| 15 | </bean> | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | <!-- Scans the classpath of this application for @Components to deploy as beans --> | |
| 18 | <context:component-scan base-package="com.raistudies" /> |

|  |  |
| --- | --- |
| 19 |  |
| 20 | <!-- Configures the @Controller programming model --> | |

|  |  |  |
| --- | --- | --- |
| 21 | <mvc:annotation-driven /> | |
| 22 |  |

|  |  |
| --- | --- |
| 23 | <!-- Resolves view names to protected .jsp resources within the /WEB-INF/views directory --> |
| 24 | <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"> |

|  |  |  |
| --- | --- | --- |
| 25 | <property name="prefix" value="/WEB-INF/jsp/"/> | |
| 26 | <property name="suffix" value=".jsp"/> |

|  |  |  |
| --- | --- | --- |
| 27 | </bean> | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | </beans> |

Our Spring MVC configuration file is changed only to make support of annotation based configuration in our example project.

**RegistrationValidator.java**

RegistrationValidator.java has been changed to remove property level validations and to make the class as spring bean using annotation.

|  |  |  |
| --- | --- | --- |
| 1 | package com.raistudies.validator; | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | import org.springframework.stereotype.Component; | |
| 4 | import org.springframework.validation.Errors; |

|  |  |  |
| --- | --- | --- |
| 5 | import org.springframework.validation.Validator; | |
| 6 |  |

|  |  |  |
| --- | --- | --- |
| 7 | import com.raistudies.domain.RegistrationBean; | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | @Component | |
| 10 | | public class RegistrationValidator implements Validator { | |

|  |  |
| --- | --- |
| 11 |  |
| 12 | @Override | |

|  |  |
| --- | --- |
| 13 | public boolean supports(Class c) { |
| 14 | return RegistrationBean.class.isAssignableFrom(c); | |

|  |  |  |
| --- | --- | --- |
| 15 | } | |
| 16 |  |

|  |  |
| --- | --- |
| 17 | @Override |
| 18 | public void validate(Object command, Errors errors) { | |

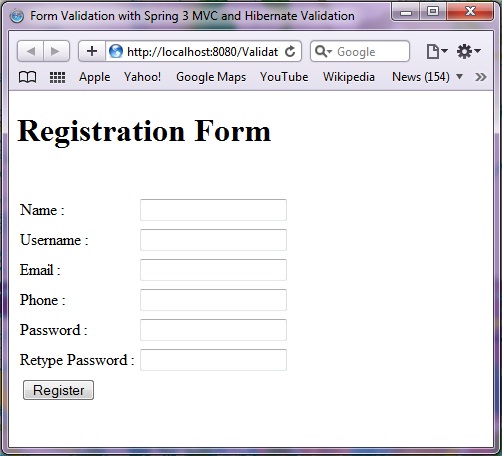
|  |  |
| --- | --- |
| 19 | RegistrationBean regBean = (RegistrationBean)command; |
| 20 | if(!regBean.getPassword().equals(regBean.getRePassword())) | |

|  |  |  |
| --- | --- | --- |
| 21 | errors.rejectValue("rePassword","password.notmatch"); | |
| 22 | } |

|  |  |
| --- | --- |
| 23 | } |

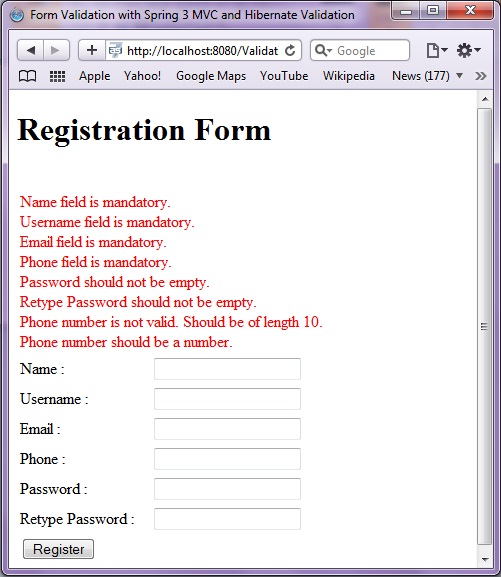
Here, you will find a new annotation @Component. This is a class level annotation that makes the class a spring bean ans spring can inject instance of this class when ever required in spring dependency solution.

Now, deploy the war in tomcat 6 and hit the url in browser you will get following screen :

[](http://www.raistudies.com/wp-content/uploads/2011/08/Form-Validation-in-Spring-MVC-3-using-Annotation1.jpg)

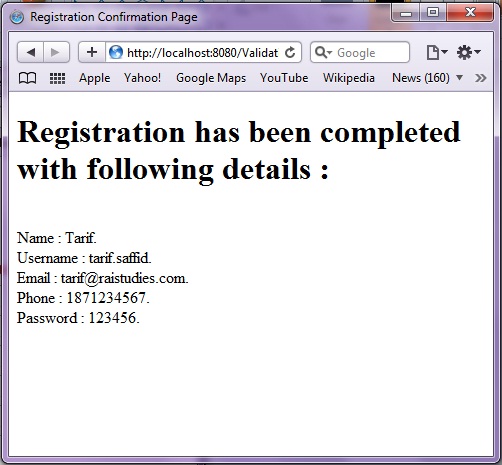
Form Validation in Spring MVC 3 using Annotation

You can check the validation by simply clicking on “Register” button without filling any data, you will get the screen :

[](http://www.raistudies.com/wp-content/uploads/2011/08/Validation-Errors-in-Spring-MVC-31.jpg)

Validation Errors in Spring MVC 3 using Hibernate Validator

Try out other validations and then when you fill correct values, you will get following screen :

[](http://www.raistudies.com/wp-content/uploads/2011/08/Form-Validation-Success-in-Spring-MVC-3-using-Annotation1.jpg)

Form Validation Success in Spring MVC 3 using Annotation

That all from Spring MVC form validation using Hibernate Bean Validator Framework. You can download