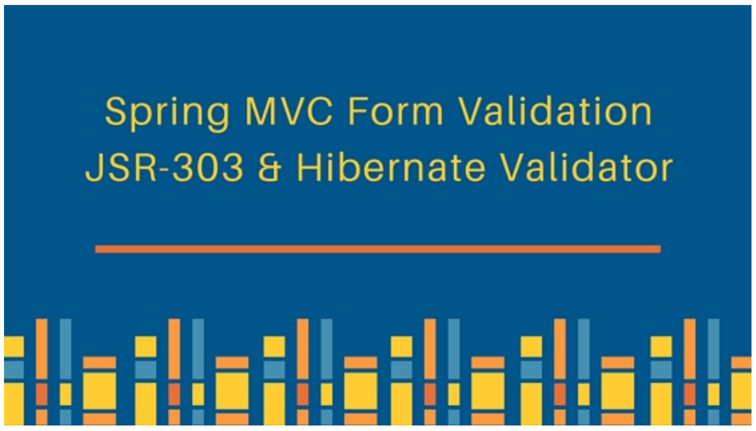
Spring Validation Example – Spring MVC Form Validator

APRIL 2, 2018 BY PANKAJ — 53 COMMENTS

When we accept user inputs in any web application, it become necessary to validate them. We can validate the user input at client side using JavaScript but it's also necessary to validate them at server side to make sure we are processing valid data incase user has javascript disabled.

Table of Contents [hide]

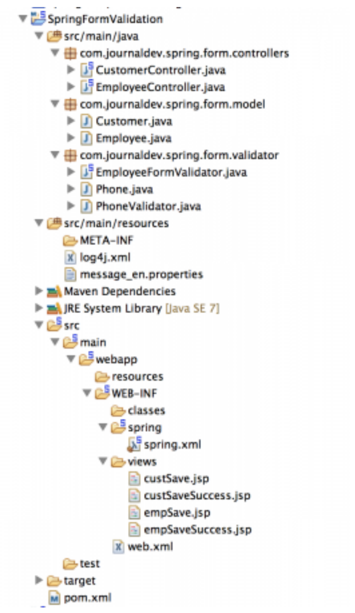
Spring Validation



Spring MVC Framework supports JSR-303 specs by default and all we need is to add JSR-303 and it's implementation dependencies in Spring MVC application. Spring also provides `@Validator` annotation and `BindingResult` class through which we can get the errors raised by Validator implementation in the controller request handler method.

We can create our custom validator implementations by two ways – first one is to create an annotation that confirms to the JSR-303 specs and implement it's Validator class. Second approach is to implement the `org.springframework.validation.Validator` interface and add set it as validator in the Controller class using `@InitBinder` annotation.

Let's create a simple Spring MVC project in Spring Tool Suite where we will use JSR-303 specs with it's implementation artifact **hibernate-validator**. We will use annotation based form validation and create our own custom validator based on JSR-303 specs standards. We will also create our own custom validator class by implementing `Validator` interface and use it in one of the controller handler methods. Our final project looks like below image.



```
SpringFormValidation
  src/main/java
    com.journaldev.spring.form.controllers
      CustomerController.java
      EmployeeController.java
    com.journaldev.spring.form.model
      Customer.java
      Employee.java
    com.journaldev.spring.form.validator
      EmployeeFormValidator.java
      Phone.java
      PhoneValidator.java
  src/main/resources
    META-INF
    log4j.xml
    message_en.properties
  Maven Dependencies
  JRE System Library [Java SE 7]
  src
    main
      webapp
        resources
        WEB-INF
          classes
          spring
            spring.xml
          views
            custSave.jsp
            custSaveSuccess.jsp
            empSave.jsp
            empSaveSuccess.jsp
          web.xml
    test
  target
  pom.xml
```

Let's look at each of the components one by one.

Spring MVC Form Validator

Our final pom.xml file looks like below. Apart from standard Spring MVC artifacts, we have validation-api and hibernate-validator dependencies in the project.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.journaldev</groupId>
        <artifactId>spring</artifactId>
        <name>SpringFormValidation</name>
        <packaging>war</packaging>
        <version>1.0.0-BUILD-SNAPSHOT</version>
        <properties>
                <java-version>1.7</java-version>
                <org.springframework-version>4.0.2.RELEASE</org.springframework-
version>
                <org.aspectj-version>1.7.4</org.aspectj-version>
                <org.slf4j-version>1.7.5</org.slf4j-version>
        </properties>
        <dependencies>
        <!-- Form Validation using Annotations -->
                <dependency>
                        <groupId>javax.validation</groupId>
```

Deployment Descriptor

When you create Spring MVC project from STS, it creates two context configuration files. I have cleaned it up a bit and have only one spring bean configuration file. My final web.xml file looks like below.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

        <!-- Processes application requests -->
        <servlet>
                <servlet-name>appServlet</servlet-name>
                <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <init-param>
                        <param-name>contextConfigLocation</param-name>
                        <param-value>/WEB-INF/spring/spring.xml</param-value>
                </init-param>
                <load-on-startup>1</load-on-startup>
        </servlet>

        <servlet-mapping>
                <servlet-name>appServlet</servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>
```

Spring Bean Configuration File

Usually we look into spring wirings at the last, but this time we don't have much configurations in the spring bean configuration file. Our final spring.xml file looks like below.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:beans="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
                http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

        <!-- DispatcherServlet Context: defines this servlet's request-processing
```

```
infrastructure -->

        <!-- Enables the Spring MVC @Controller programming model -->
        <annotation-driven />

        <!-- Handles HTTP GET requests for /resources/** by efficiently serving up
static resources in the ${webappRoot}/resources directory -->
        <resources mapping="/resources/**" location="/resources/" />
```

The only important point to note are `employeeValidator` bean that we will inject into one of the controller and `messageSource` bean to read the localized data from resource bundles. Rest of the part is to support annotations, view resolvers and providing package to scan for Controller classes and other components.

Model Classes

We have two model classes in this project – first one where we will use JSR-303 annotation and our custom annotation based validator and second one where we will use only our Validator implementation.

Customer.java code:

```
package com.journaldev.spring.form.model;

import java.util.Date;

import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Past;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;
import org.springframework.format.annotation.DateTimeFormat;

import com.journaldev.spring.form.validator.Phone;

public class Customer {

        @Size(min=2, max=30)
    private String name;

    @NotEmpty @Email
```

Notice that we are using @Email, @NotEmpty and @DateTimeFormat annotations that are additional to JSR-303 and provided by hibernate validator implementation. Some of the JSR-303 annotations that we are using are @Size, @NotNull etc.

@Phone annotation used is our custom implementation based on JSR-303 specs, we will look into it in next section.

Employee.java code:

```
package com.journaldev.spring.form.model;

public class Employee {

        private int id;
        private String name;
        private String role;

        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getRole() {
                return role;
```

Employee is a standard java bean and we will use our custom Validator implementation to validate the form with Employee bean.

Custom Validator Implementations

Phone.java code:

```
package com.journaldev.spring.form.validator;

import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import java.lang.annotation.ElementType;
import java.lang.annotation.RetentionPolicy;

import javax.validation.Constraint;
import javax.validation.Payload;

@Documented
@Constraint(validatedBy = PhoneValidator.class)
@Target( { ElementType.METHOD, ElementType.FIELD })
@Retention(RetentionPolicy.RUNTIME)
public @interface Phone {


    String message() default "{Phone}";

    Class<?>[] groups() default {};
```

Most of the part is boiler-plate code to confirm with JSR-303 specs. The most important part is @Constraint annotation where we provide the class that will be used for validation i.e `PhoneValidator`.

PhoneValidator.java code:

```
package com.journaldev.spring.form.validator;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

public class PhoneValidator implements ConstraintValidator<Phone, String> {

        @Override
        public void initialize(Phone paramA) {
        }

        @Override
        public boolean isValid(String phoneNo, ConstraintValidatorContext ctx) {
                if(phoneNo == null){
                        return false;
                }
                //validate phone numbers of format "1234567890"
        if (phoneNo.matches("\\d{10}")) return true;
        //validating phone number with -, . or spaces
        else if(phoneNo.matches("\\d{3}[-\\.\\s]\\d{3}[-\\.\\s]\\d{4}")) return true;
        //validating phone number with extension length from 3 to 5
        else if(phoneNo.matches("\\d{3}-\\d{3}-\\d{4}\\s(x|(ext))\\d{3,5}")) return
```

Our JSR-303 specs validator implementation should implement `javax.validation.ConstraintValidator` interface. If we are using some resource such as DataSource, we can initialize them in the `initialize()` method. The validation method is `isValid` and it returns true if data is valid else it should return false.

If you are new to regular expressions, you can read more about it at Java Regular Expressions Tutorial.

EmployeeFormValidator.java class code:

```
package com.journaldev.spring.form.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import com.journaldev.spring.form.model.Employee;

public class EmployeeFormValidator implements Validator {

        //which objects can be validated by this validator
        @Override
        public boolean supports(Class<?> paramClass) {
                return Employee.class.equals(paramClass);
        }

        @Override
        public void validate(Object obj, Errors errors) {
                ValidationUtils.rejectIfEmptyOrWhitespace(errors, "id",
"id.required");
```

EmployeeFormValidator is the validator implementation that is specific to Spring Framework. `supports()` method implementation by Spring Framework to know objects on which this validation can be used.

We implement `validate()` method and add errors if any field validation fails. Spring provides `org.springframework.validation.ValidationUtils` utility class for basic validations such as null or empty. Once this method returns, spring framework binds the Errors object to the BindingResult object that we use in our controller handler method.

Notice that `ValidationUtils.rejectIfEmptyOrWhitespace()` last argument takes the key name for message resources. This way we can provide localized error messages to the user. For more information about i18n in Spring, read Spring i18n Example.

Controller Classes

We have two controller classes, one for annotation based form validation and another for our custom validator.

CustomerController.java class code:

```
package com.journaldev.spring.form.controllers;

import java.util.HashMap;
import java.util.Map;

import javax.validation.Valid;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.journaldev.spring.form.model.Customer;

@Controller
public class CustomerController {

        private static final Logger logger = LoggerFactory
                        .getLogger(CustomerController.class);
```

When we use annotation based form validation, we just need to make little changes in our controller handler method implementation to get it working.

First we need to annotate model object that we want to validate with `@Valid` annotation. Then we need to have BindingResult argument in the method, spring takes care of populating it with error messages. The handler method logic is very simple, if there are any errors we are responding with the same page or else we are redirecting user to the success page.

Another important point to note is that we are adding “customer” attribute to the model, this is necessary to let Spring framework know which model object to use in the form page. If we won’t do it, object binding to form data will not take place and our form validation will not work.

EmployeeController.java class code:

```
package com.journaldev.spring.form.controllers;

import java.util.HashMap;
import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.validation.Validator;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.journaldev.spring.form.model.Employee;
```

For using custom validator, first we need to inject it in the controller class. We are using spring bean auto wiring to achieve this using `@Autowired` and `@Qualifier` annotations.

Next we need to have a method that will take WebDataBinder as argument and we set our custom validator to be used. This method should be annotated with `@InitBinder` annotation.

Using `@ModelAttribute` is another way to add our bean object to the Model. Rest of the code is similar to customer controller implementation.

Form Validation Error Messages Resource Bundle

It's time to look at our resource bundle where we have different types of messages to be used for validation errors.

message_en.properties file:

```
#application defined error messsages
id.required=Employee ID is required
name.required=Employee Name is required
role.required=Employee Role is required
negativeValue={0} can't be negative or zero

#Spring framework error messages to be used when conversion from form data to bean
fails
typeMismatch.int={0} Value must be an integer
typeMismatch.java.lang.Integer={0} must be an integer
typeMismatch={0} is of invalid format

#application messages for annotations, {ValidationClass}.{modelObjectName}.{field}
#the {0} is field name, other fields are in alphabatical order, max and then min
Size.customer.name=Customer {0} should be between {2} and {1} characters long
NotEmpty.customer.email=Email is a required field
```

```
NotNull.customer.age=Customer {0} should be in years

#Generic annotation class messages
Email=Email address is not valid
NotNull=This is a required field
NotEmpty=This is a required field
```

I have provided message key details in the comment itself, so I will skip them here. The only important point to note here is the way messages will be looked up, first key name {ValidationClass}.{modelObjectName}.{field} is looked up and if that is not found then {ValidationClass}.{modelObjectName} is looked up. If that is missing, then finally {ValidationClass} key is looked up. If nothing is found then the default message provided will be returned.

Read more about resource messages at Spring Localization Example.

View Pages with Form and Errors

Since we are using Spring framework validation implementation, we will have to use Spring Form tags to get the errors and set the form bean and variable names.

Our custSave.jsp file code is given below.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://www.springframework.org/tags/form"
        prefix="springForm"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Customer Save Page</title>
<style>
.error {
        color: #ff0000;
        font-style: italic;
        font-weight: bold;
}
</style>
</head>
<body>

        <springForm:form method="POST" commandName="customer"
                action="save.do">
```

`commandName="customer"` is used to set the name of the model attribute under which form object is exposed. It's default value is "command" by default, hence we should set it to the model attribute name we are using in our controller classes.

`springForm:errors` is used to render the errors, if any, found when the page is rendered. `path` attribute is used to define the object property to be used for data binding. Rest of the code is standard HTML with some CSS for error messages styling.

Our custSaveSuccess.jsp file is given below.

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

<%@ page session="false" %>
<html>
<head>
        <title>Customer Saved Successfully</title>
</head>
<body>
<h3>
        Customer Saved Successfully.
</h3>

<strong>Customer Name:${customer.name}</strong><br>
<strong>Customer Email:${customer.email}</strong><br>
<strong>Customer Age:${customer.age}</strong><br>
<strong>Customer Gender:${customer.gender}</strong><br>
<strong>Customer Birthday:<fmt:formatDate value="${customer.birthday}" type="date" />
</strong><br>

</body>
</html>
```

Simple JSP page showing the customer values if there are no validation errors and this page is returned as response. It's name is empSave.jsp.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
        pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="http://www.springframework.org/tags/form"
        prefix="springForm"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Employee Save Page</title>
<style>
.error {
        color: #ff0000;
        font-style: italic;
        font-weight: bold;
}
</style>
</head>
<body>

        <springForm:form method="POST" commandName="employee"
                action="save.do">
```

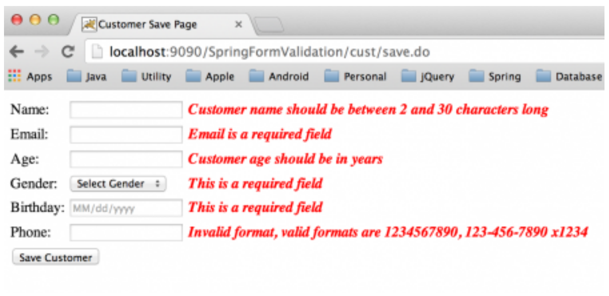
empSaveSuccess.jsp file:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
```

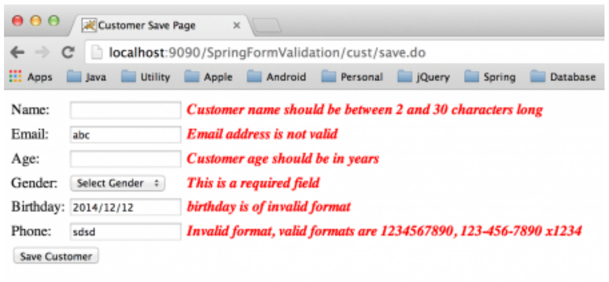
```
<head>
        <title>Employee Saved Successfully</title>
</head>
<body>
<h3>
        Employee Saved Successfully.
</h3>

<strong>Employee ID:${emp.id}</strong><br>
<strong>Employee Name:${emp.name}</strong><br>
<strong>Employee Role:${emp.role}</strong><br>

</body>
</html>
```

Test the Spring MVC Form Validation Application

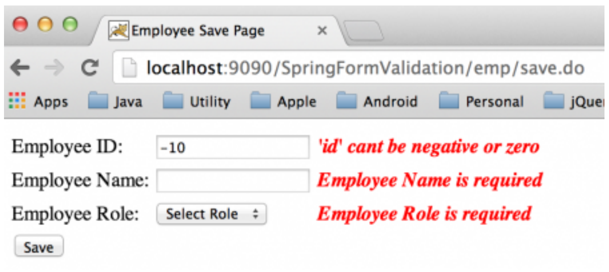
Our application is ready to deploy and run some tests, deploy it in your favorite servlet container. I am using Apache Tomcat 7 and below images show some of the pages with validation error messages. Based on your input data, you might get different error messages too.


Customer Save Page
localhost:9090/SpringFormValidation/cust/save.do
Apps Java Utility Apple Android Personal jQuery Spring Database
Select Gender
MM/dd/yyyy
Save Customer

Name:		*Customer name should be between 2 and 30 characters long*
Email:		*Email is a required field*
Age:		*Customer age should be in years*
Gender:	Select Gender	*This is a required field*
Birthday:	MM/dd/yyyy	*This is a required field*
Phone:		*Invalid format, valid formats are 1234567890, 123-456-7890 x1234*


Customer Save Page
localhost:9090/SpringFormValidation/cust/save.do
Apps Java Utility Apple Android Personal jQuery Spring Database
Select Gender
Save Customer

Name:		*Customer name should be between 2 and 30 characters long*
Email:	abc	*Email address is not valid*
Age:		*Customer age should be in years*
Gender:	Select Gender	*This is a required field*
Birthday:	2014/12/12	*birthday is of invalid format*
Phone:	sdsd	*Invalid format, valid formats are 1234567890, 123-456-7890 x1234*


Employee Save Page
localhost:9090/SpringFormValidation/emp/save.do
Apps Java Utility Apple Android Personal jQue
Select Role
Save

Employee ID:	-10	*'id' cant be negative or zero*
Employee Name:		*Employee Name is required*
Employee Role:	Select Role	*Employee Role is required*

That's all for Spring MVC Form validation with different ways and using resource bundles for localized error messages. You can download the sample project from below link and play around with it to learn more.

Download Spring Form Validation Project

« PREVIOUS
Spring MVC Exception Handling – @ControllerAdvice, @ExceptionHandler, HandlerExceptionResolver

NEXT »
Spring MVC Interceptor HandlerInterceptorAdapter, HandlerInterceptor Example

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

shan says
DECEMBER 12, 2017 AT 4:43 AM
Hi,
The article is so good , but in real time no body will go for spring validation approach to validate form .
client side validation is simple and good,
any body really use spring validator for validation ?
Reply

pastor Zion says
DECEMBER 28, 2017 AT 5:22 AM
Hi Shan,
Client side validation could cause you unexpected troubles.The input field is an access door into the

heart of your app and controlling what gets in is very important.The trouble with doing validation with javascript is that it can be disabled at client side.For this singular reason consider server side validation.

Reply

Arun SIngh says

NOVEMBER 26, 2017 AT 12:54 AM

explained in detail
thanks

Reply

Manas Kumar says

NOVEMBER 21, 2017 AT 10:45 AM

Cannot find class [com.journaldev.spring.form.validator.EmployeeFormValidator] for bean with name 'employeeValidator' defined in ServletContext resource [/WEB-INF/spring/spring.xml]; nested exception is java.lang.ClassNotFoundException: com.journaldev.spring.form.validator.EmployeeFormValidator
I am getting this error while running.I have checked all possibilities but unable to resolve Could you please help me.??

Reply

Ankit kumar gupta says

FEBRUARY 27, 2017 AT 2:37 AM

please help me out in one problem related to validation
how to validate the Path Variable in spring application? if possible then please provide me example also.

Reply

Aatif says

NOVEMBER 30, 2016 AT 5:13 AM

localhost:8080/SpringFormValidation/emp/save.do showing HTTP 404 error. Not getting the form page. !!A

Reply

Praveen says

DECEMBER 20, 2016 AT 10:30 PM

Aatif please try the below URL.
localhost:8080/spring/emp/save.do

Reply

Farhan says

OCTOBER 25, 2016 AT 11:33 PM

Hi Pankaj,

Thanks, example is good. I have two question regarding Spring MVC validation. I want to know how to register more than one validators for single command object by using @IntBinder annotated method so that I need not to call them explicitly in my controller method for validation. My second question is I want to use JSR 303 validation annotation such as @NotBlank as well as my own custom designed validation annotation for example @Phone (which you have shown in this example) but these two validation are annotations are different then how can I use them without loosing other one? Can I use @Valid @Validated to command argument of request handling method of controller or there is some other way.
Wanting for your reply
Reply

Stephane Eybert says
JANUARY 25, 2017 AT 12:21 AM
You can have multiple @IntBinder("EntityNameMatchingTheOneToBeValidated") annotated methods with method names of your liking, and adding in each method the validator you need, that will be used for the entity type specified in the annotation attribute.
Reply

IMRAN AHMAD says
AUGUST 11, 2016 AT 4:56 AM
How to validation int array in server side in java spring
Reply

kunal says
JUNE 15, 2016 AT 8:45 PM
Hi,
Your articles are very useful.
I have tried to built the application and deploy it on JBOSS.
But I got an error " ERROR [org.apache.catalina.core.ContainerBase.[jboss.web].[defaulthost].[/spring-1.0.0-BUILD-SNAPSHOT]] (ServerService Thread Pool — 57) JBWEB000289: Servlet spring threw load() exception: java.io.FileNotFoundException:
Could not open ServletContext resource [/WEB-INF/spring/spring.xml]" .
Could you please help me to resolve the same.
Thank you
Reply

Pankaj says
JUNE 16, 2016 AT 2:50 AM
It clearly says file not found, you need to see where is spring.xml file and whether it's being packaged in your WAR or not.
Reply

David Adkins says
MAY 18, 2016 AT 3:08 PM
Your tutorials are some of the best I have seen, Thank You
Reply

IMRAN AHMAD says
AUGUST 11, 2016 AT 4:58 AM
Please suggest how to validate int array in server side in java spring
Reply

koushal says
APRIL 15, 2016 AT 3:24 AM
Thanks a lot , its very useful.
Reply

Dinesh Pise says
NOVEMBER 30, 2015 AT 6:41 AM
Hi Pankaj,
Can we have custom validation and annotation validation together. Say for example in Customer I want to have annotation validation for all field except for age, for age I want to use validator class. Is that possible.
Regards,
Dinesh Pise
Reply

Zac C says
SEPTEMBER 25, 2015 AT 2:36 PM
For the save page, when you run save.do, the number 0 is defaulted into the "Employee ID" field. How'd you get that to default to zero? What if I wanted employer name and employee role to default to null.
Reply

Zac C says
SEPTEMBER 18, 2015 AT 2:57 PM
"When you create Spring MVC project from STS, it creates two context configuration files. I have cleaned it up a bit and have only one spring bean configuration file. My final web.xml file looks like below."
Where are the original two? I only see the one web.xml.
Reply

Pankaj says
SEPTEMBER 18, 2015 AT 10:45 PM
web.xml is deployment descriptor, spring.xml is the beans configuration file.
Reply

Zac C says
SEPTEMBER 21, 2015 AT 8:16 AM
where is spring.xml?
Reply

Zac C says
SEPTEMBER 21, 2015 AT 8:52 AM
I found "spring.xml" by doing a google search and finding out my question is a common question. Therefore, I suggest you clarify that statement about the two config files. My "spring.xml" file is called "servlet-context.xml".
Reply

Vinod Paliwal says
AUGUST 16, 2015 AT 4:50 AM
Hi Pankaj Sir,
I do appreciate you each ans every topics explanation as which helped a lot to develop my website.
I have deployed the above project and without using pom.xml (means not maven) and facing below error.
Am i missing any jar file ?
Please provide your guide line .

```
org.springframework.beans.factory.BeanCreationException: Error creating bean with name
'org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping#0': Initialization of
bean failed; nested exception is org.springframework.beans.factory.BeanCreationException: Error
creating bean with name 'customerController' defined in file
[D:\WORK\Spring3MVC4Site\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\Spring3MVCFormValidation\WEB-
INF\classes\com\journaldev\spring\form\controllers\CustomerController.class]: Instantiation of bean
failed; nested exception is java.lang.NoClassDefFoundError: Could not initialize class
com.journaldev.spring.form.controllers.CustomerController
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.java:5
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.createBean(AbstractAutowireCapableBeanFactory.java:450
org.springframework.beans.factory.support.AbstractBeanFactory$1.getObject(AbstractBeanFactory.java:290)
org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:222)
org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:287)
org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:189)
org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:562)
org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:871)
org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:423)
org.springframework.web.servlet.FrameworkServlet.createWebApplicationContext(FrameworkServlet.java:443)
org.springframework.web.servlet.FrameworkServlet.createWebApplicationContext(FrameworkServlet.java:459)
org.springframework.web.servlet.FrameworkServlet.initWebApplicationContext(FrameworkServlet.java:340)
org.springframework.web.servlet.FrameworkServlet.initServletBean(FrameworkServlet.java:307)
org.springframework.web.servlet.HttpServletBean.init(HttpServletBean.java:127)
javax.servlet.GenericServlet.init(GenericServlet.java:160)
org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:502)
org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:99)
org.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:953)
org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:408)
org.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:1023)
org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:589)
org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:310)
java.util.concurrent.ThreadPoolExecutor$Worker.runTask(Unknown Source)
java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
java.lang.Thread.run(Unknown Source)
```

Thanks,
Vinod Paliwal

Reply

Andy says
OCTOBER 13, 2015 AT 12:06 AM
update your hibernate-validator version and try again, this is work for me.
org.hibernate
hibernate-validator
5.1.0.Final
Reply

Hieu says
AUGUST 11, 2015 AT 3:28 PM
I add new plugins in pom.xml
org.apache.tomcat.maven
tomcat7-maven-plugin
2.2
and
org.eclipse.jetty
jetty-maven-plugin
9.0.6.v20130930
/${project.artifactId}
When I start tomcat by command : mvn tomcat7:run or mvn jetty-run. I have an error:
Caused by: java.lang.AbstractMethodError:
org.hibernate.validator.engine.ConfigurationImpl.getDefaultParameterNameProvider()Ljavax/validation/ParameterNameProvider;
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:606)
at org.springframework.util.ReflectionUtils.invokeMethod(ReflectionUtils.java:196)
at org.springframework.util.ReflectionUtils.invokeMethod(ReflectionUtils.java:181)
at
org.springframework.validation.beanvalidation.LocalValidatorFactoryBean.configureParameterNameProviderIfPossible(LocalValidatorFactoryB
at
org.springframework.validation.beanvalidation.LocalValidatorFactoryBean.afterPropertiesSet(LocalValidatorFactoryBean.java:245)
at
org.springframework.validation.beanvalidation.OptionalValidatorFactoryBean.afterPropertiesSet(OptionalValidatorFactoryBean.java:40)
at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFactory.ja
at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.java:15
… 66 more
Can you explain my error and how to fix it.
Thanks
Reply

Hieu says
AUGUST 11, 2015 AT 3:33 PM
I add new plugins in pom.xml
For tomcat plugin

```
org.apache.tomcat.maven
tomcat7-maven-plugin
```

```
2.2
and for jetty plugin
org.eclipse.jetty
jetty-maven-plugin
9.0.6.v20130930
/${project.artifactId}
When I start tomcat by command : mvn tomcat7:run or mvn jetty-run. I have an error:

Caused by: java.lang.AbstractMethodError:
org.hibernate.validator.engine.ConfigurationImpl.getDefaultParameterNameProvider()Ljavax/validation/ParameterNameP
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:606)
at org.springframework.util.ReflectionUtils.invokeMethod(ReflectionUtils.java:196)
at org.springframework.util.ReflectionUtils.invokeMethod(ReflectionUtils.java:181)
at
org.springframework.validation.beanvalidation.LocalValidatorFactoryBean.configureParameterNameProviderIfPossible(L
at
org.springframework.validation.beanvalidation.LocalValidatorFactoryBean.afterPropertiesSet(LocalValidatorFactoryBe
at
org.springframework.validation.beanvalidation.OptionalValidatorFactoryBean.afterPropertiesSet(OptionalValidatorFac
at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCap
at
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapabl
… 66 more
Can you explain my error and how to fix it.
Thanks
```

Reply

Vijay G says

AUGUST 10, 2015 AT 7:31 AM

Hi,

Please give me one solution in spring frame work for the below scenario,

1. If I click or submit any action in jsp, that action name should be configured aginst the controller in the spring config xml.
2. Once the flow has gone to Controller, ModelView will return success, failure or any constant string.
3. Based on that string, view will be redirected based on the configuration in the same config.
4. Different method name will be used in the same controller.

Problem should be addressed::

1. We have to identify the method name in the controller by action name and go to that method and come back to the spring config xml and redirect the view based on the string returned from the controller.

I am waiting for your response..

Thank You…

Vijay G

.

Reply

Rajasekhar Balusupati says
AUGUST 6, 2015 AT 10:52 AM
Very nice explanation, Pankaj!!
I have a situation to validate the date for Future and today!! We have @Future annotation. Can you please help me to validate today's date?
Appreciate your help. Thanks in advance!!
Reply

NHP says
JUNE 17, 2015 AT 3:02 AM
Tomcat Server showing blank page when started from eclipse : 'Your Project is not working'
WARN : org.springframework.web.servlet.PageNotFound – No mapping found for HTTP request with URI [/spring/custSave] in DispatcherServlet with name 'appServlet'
Reply

bindu says
MARCH 9, 2015 AT 3:10 AM
hi pankaj,well explained.
I am facing problem while running this project.My tomcat is going blank after running this project on eclipse usin tomcat:run in configuration.
Kindly let me know the reason this.
Outside of eclipse tomcat is running perfectly
Reply

Pankaj says
MARCH 9, 2015 AT 3:17 AM
Hi Bindu,
If application is running fine when tomcat is not launched from Eclipse, it means everything is fine. I would suggest you to add a fresh server to Eclipse and try again, some issues with the current configured server might be causing this. Also check the URL in browser when app is launched in Eclipse.
Reply

abdul says
DECEMBER 17, 2014 AT 11:52 PM
Thanks alot very helpful.
Reply

Max says
DECEMBER 12, 2014 AT 9:03 AM
Hello, Pankaj!
Please, change the Hibernate version in dependency from 5 to 4:
org.hibernate
hibernate-validator
4.1.0.Final

Reply

Pankaj says
DECEMBER 12, 2014 AT 9:57 AM
Thanks, corrected the typo error.
Reply

Kanika says
DECEMBER 12, 2014 AT 3:14 AM
Could you include a snippet to give errors in a json result and include Ajax Request?
Reply

Adarsh says
DECEMBER 27, 2016 AT 4:55 AM
Hi Kanika
The JSON request bind with JSONRequest of controller context
public override object BindModel(ControllerContext controllerContext,
ModelBindingContext bindingContext)
{
if (!IsJSONRequest(controllerContext))
{
return base.BindModel(controllerContext, bindingContext);
}
pls see above code for manipulate the JSON Request
Reply

Sandeep says
NOVEMBER 21, 2014 AT 10:57 PM
Thanks for the post, it is really helpful.
I was wondering how validation can be done in case we have @OneToMany mapping (say Employee class has Set of Address and none of the address field is inputted by the user from frontend, thus returning NULL to Spring MVC)?
Thanks in advance,
Reply

Adarsh says
DECEMBER 27, 2016 AT 4:51 AM
Hi Sandeep,
@OneToMany mapping is totally depends on @Request Mapping of URL
and NULL value is coming only on that case
Regards
Adarsh
Reply

Ashirwad Kumar says
OCTOBER 7, 2014 AT 2:10 AM
Not able to download the zip even after following on Twitter.
Reply

Rahul says
OCTOBER 2, 2014 AT 8:56 PM
I see that you have defined a messageSource bean in your Spring bean configuration file. Where does this bean get used? I don't see a place where it gets auto wired. Can you please explain how Spring internally works to get messages from this bean. Is the bean name "messageSource" special to Spring MVC? Does this bean need to be named with that name?
Reply

Pankaj says
OCTOBER 2, 2014 AT 9:59 PM
It's to load messages property files, it's used for localization, read more at
https://www.journaldev.com/2610/spring-mvc-internationalization-i18n-and-localization-l10n-example
You can keep any other name, but in that case you will have to bind it manually, with this name it's autowired.
Reply

Rahul says
OCTOBER 4, 2014 AT 8:49 PM
Pankaj, thanks for the quick feedback. You are right. I just tried this. Yes I need to name the Message Source bean as "messageSource" for it to be picked up.
Reply

Dmytro says
SEPTEMBER 16, 2014 AT 11:44 AM
Thank you for this useful tutorial!
Please add message.properties file to src/main/resources folder. Spring cannot resolve bound if there is no default properties file and client uses another locale. For example, for my locale "_ru" I've got default text message or exception stack trace instead of text from message_en.properties.
Reply

Pankaj says
SEPTEMBER 16, 2014 AT 1:26 PM
Thanks for the suggestion.
Reply

Vlad says
APRIL 17, 2018 AT 5:19 AM
I have default message from stack trace as well. What's wrong? I've added message.properties file to resources folder

Reply

kumar says
JULY 29, 2014 AT 7:41 AM
hi pankaj very nice explanation ….
could you post webservice example.wre webservices r useful in real time?
Reply

dev says
JULY 22, 2014 AT 10:54 PM
such nice deep concept……
Reply

Tanuja says
JULY 17, 2014 AT 4:31 AM
Really great concepts of vaidation thanks a lot……………..
Reply

santhosh says
JULY 16, 2014 AT 12:13 AM
Your site is really good with the examples provided for java and spring framework.
Could you post an article on Spring AOP concepts using annotations
Reply

Pankaj says
JULY 16, 2014 AT 2:52 AM
Its already posted, check the Spring category articles.
Reply

Anil says
JULY 6, 2014 AT 5:12 AM
{Phone} validation is not showing the value from resource bundle
Adjacent to the phone it is showing key instead of value
{Phone}
Reply

Rajesh Antony says
JUNE 15, 2014 AT 3:13 PM

Hi Pankaj, the article has been very useful for me to pickup the basics. Thank you very much

Reply

eduardo says

JUNE 11, 2014 AT 9:59 AM

hi, coud you upload again the zip file. thanks

Reply

Pankaj says

JUNE 11, 2014 AT 10:33 AM

it was an issue with the plugin I use for managing downloads, solved it. Please try now.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

SPRING FRAMEWORK

RECOMMENDED TUTORIALS

SQL Server

Learn Programming Online

Create Your Own Website

Java Tutorial for Beginners

Python Programming Tutorial

Oracle Database Tools

JSP Interview Questions

Interior Design Software