**HibernateValidator学习文档**

# 相关网址

官网：<http://hibernate.org/validator/>

官方文档：<http://hibernate.org/validator/documentation/>

官方文档6.0：

<https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/>

官方文档5.4：

<https://docs.jboss.org/hibernate/validator/5.4/reference/en-US/html_single/>

教程：

<https://www.cnblogs.com/mr-yang-localhost/p/7812038.html>

github地址：

<https://github.com/dampce32/demo-master/tree/master/demo-master/demo-hibernate-validator-master>

# 简介

 在开发中经常需要写一些字段校验的代码，比如字段非空，字段长度限制，邮箱格式验证等等，写这些与业务逻辑关系不大的代码个人感觉有两个麻烦：

* 验证代码繁琐，重复劳动
* 方法内代码显得冗长
* 每次要看哪些参数验证是否完整，需要去翻阅验证逻辑代码

hibernate validator提供了一套比较完善、便捷的验证实现方式。

spring-boot-starter-web包里面有hibernate-validator包，不需要引用hibernate validator依赖。

spring boot 2.0.0.RELEASE版本hibernate validator 6.0.7

# Helloworld

## Pom.xml文件

<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>org.lys.demo</groupId>

<artifactId>demo-hibernate-validator-n0001</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>demo-hibernate-validator-n0001 Maven Webapp</name>

<url>http://maven.apache.org</url>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

<build>

<finalName>demo-hibernate-validator-n0001</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

## Model

**import** javax.validation.constraints.Min;

**import** javax.validation.constraints.NotNull;

**import** javax.validation.constraints.Size;

**public** **class** Car {

@NotNull(message="manufacturer must not be null")

**private** String manufacturer;

@NotNull(message="must not be null")

@Size(min = 2, max = 14,message="licensePlate size must be between 2 and 14")

**private** String licensePlate;

@Min(value=2,message="seatCount must be greater than or equal to 2")

**private** **int** seatCount;

**public** Car(String manufacturer, String licencePlate, **int** seatCount) {

**this**.manufacturer = manufacturer;

**this**.licensePlate = licencePlate;

**this**.seatCount = seatCount;

}

**public** String getManufacturer() {

**return** manufacturer;

}

**public** **void** setManufacturer(String manufacturer) {

**this**.manufacturer = manufacturer;

}

**public** String getLicensePlate() {

**return** licensePlate;

}

**public** **void** setLicensePlate(String licensePlate) {

**this**.licensePlate = licensePlate;

}

**public** **int** getSeatCount() {

**return** seatCount;

}

**public** **void** setSeatCount(**int** seatCount) {

**this**.seatCount = seatCount;

}

}

@NotNull null值判断

@Size 长度判断

@Min 最小值判断

## 测试类

**package** top.linyisong.model;

**import** **static** org.junit.Assert.*assertEquals*;

**import** java.util.Set;

**import** javax.validation.ConstraintViolation;

**import** javax.validation.Validation;

**import** javax.validation.Validator;

**import** javax.validation.ValidatorFactory;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**public** **class** CarTest {

**private** **static** Validator *validator*;

@BeforeClass

**public** **static** **void** setUpValidator() {

ValidatorFactory factory = Validation.*buildDefaultValidatorFactory*();

*validator* = factory.getValidator();

}

@Test

**public** **void** manufacturerIsNull() {

Car car = **new** Car(**null**, "DD-AB-123", 4);

Set<ConstraintViolation<Car>> constraintViolations = *validator*.validate(car);

*assertEquals*(1, constraintViolations.size());

*assertEquals*("manufacturer must not be null", constraintViolations.iterator().next().getMessage());

}

@Test

**public** **void** licensePlateTooShort() {

Car car = **new** Car("Morris", "D", 4);

Set<ConstraintViolation<Car>> constraintViolations = *validator*.validate(car);

*assertEquals*(1, constraintViolations.size());

*assertEquals*("licensePlate size must be between 2 and 14", constraintViolations.iterator().next().getMessage());

}

@Test

**public** **void** seatCountTooLow() {

Car car = **new** Car("Morris", "DD-AB-123", 1);

Set<ConstraintViolation<Car>> constraintViolations = *validator*.validate(car);

*assertEquals*(1, constraintViolations.size());

*assertEquals*("seatCount must be greater than or equal to 2", constraintViolations.iterator().next().getMessage());

}

@Test

**public** **void** carIsValid() {

Car car = **new** Car("Morris", "DD-AB-123", 2);

Set<ConstraintViolation<Car>> constraintViolations = *validator*.validate(car);

*assertEquals*(0, constraintViolations.size());

}

}

验证步骤：

1. 从验证工厂取出验证器validator
2. 对象验证，返回违反约束的结果Set集合

# hibernate的校验模式

## 1、普通模式（默认是这个模式）

　　普通模式(会校验完所有的属性，然后返回所有的验证失败信息)

## 2、快速失败返回模式

　　快速失败返回模式(只要有一个验证失败，则返回)

两种验证模式配置方式：（[参考官方文档](https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/" \l "section-provider-specific-settings" \t "_blank)）

failFast：true  快速失败返回模式    false 普通模式

ValidatorFactory validatorFactory = Validation.*byProvider( HibernateValidator.class )  
 .configure()  
 .failFast( true )  
 .buildValidatorFactory();  
Validator validator = validatorFactory.getValidator();*

和 （hibernate.validator.fail\_fast：true  快速失败返回模式    false 普通模式）

ValidatorFactory validatorFactory = Validation.*byProvider( HibernateValidator.class )  
 .configure()  
 .addProperty( "hibernate.validator.fail\_fast", "true" )  
 .buildValidatorFactory();  
Validator validator = validatorFactory.getValidator();*

package top.linyisong.model;

import static org.junit.Assert.assertEquals;

import java.util.Set;

import javax.validation.ConstraintViolation;

import javax.validation.Validation;

import javax.validation.Validator;

import javax.validation.ValidatorFactory;

import org.hibernate.validator.HibernateValidator;

import org.junit.Test;

public class FailFastTest {

@Test

public void testFailFast() {

ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.class )

.configure()

.failFast( true )

.buildValidatorFactory();

Validator validator = validatorFactory.getValidator();

Car car = new Car(null, "D", 1);

Set<ConstraintViolation<Car>> constraintViolations = validator.validate(car);

assertEquals(1, constraintViolations.size());

}

@Test

public void testNotFailFast() {

ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.class )

.configure()

.failFast(false)

.buildValidatorFactory();

Validator validator = validatorFactory.getValidator();

Car car = new Car(null, "D", 1);

Set<ConstraintViolation<Car>> constraintViolations = validator.validate(car);

assertEquals(3, constraintViolations.size());

}

}

# Web项目使用

## Pom.xml文件

<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>org.lys.demo</groupId>

<artifactId>demo-hibernate-validator-n0002</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>demo-hibernate-validator-n0002 Maven Webapp</name>

<url>http://maven.apache.org</url>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

<build>

<finalName>demo-hibernate-validator-n0002</finalName>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

## Model

package top.linyisong.model;

import javax.validation.constraints.AssertFalse;

import javax.validation.constraints.NotBlank;

import javax.validation.constraints.Pattern;

public class User {

@NotBlank(message = "用户名不能为空")

private String userName;

@NotBlank(message = "年龄不能为空")

@Pattern(regexp = "^[0-9]{1,2}$", message = "年龄不正确")

private String age;

@AssertFalse(message = "必须为false")

private Boolean isFalse;

/\*\*

\* 如果是空，则不校验，如果不为空，则校验

\*/

@Pattern(regexp = "^[0-9]{4}-[0-9]{2}-[0-9]{2}$", message = "出生日期格式不正确")

private String birthday;

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getAge() {

return age;

}

public void setAge(String age) {

this.age = age;

}

public Boolean getIsFalse() {

return isFalse;

}

public void setIsFalse(Boolean isFalse) {

this.isFalse = isFalse;

}

public String getBirthday() {

return birthday;

}

public void setBirthday(String birthday) {

this.birthday = birthday;

}

}

## controller

**package** top.linyisong.controller;

**import** javax.validation.Valid;

**import** org.springframework.validation.BindingResult;

**import** org.springframework.validation.ObjectError;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** top.linyisong.model.User;

@RequestMapping(value="/user")

@RestController

**public** **class** UserController {

@RequestMapping("/create")

**public** **void** create(@RequestBody @Valid User user, BindingResult result){

**if**(result.hasErrors()){

**for** (ObjectError error : result.getAllErrors()) {

System.***out***.println(error.getDefaultMessage());

}

}

}

}

## Application

**package** top.linyisong;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** Application {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Application.**class**, args);

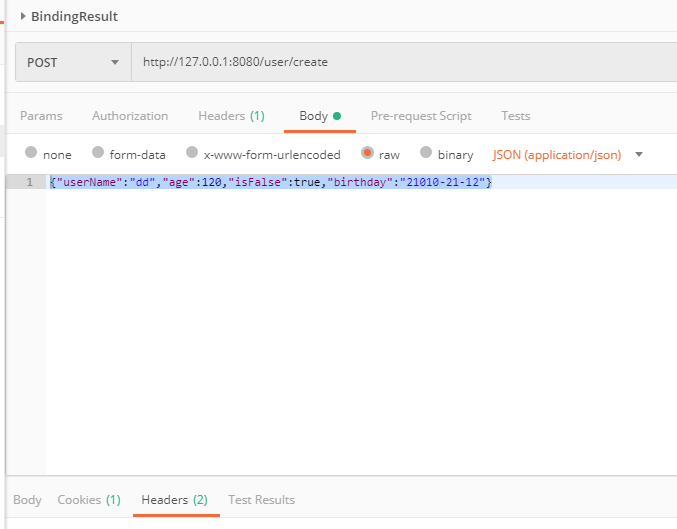
}

}

## 访问测试

运行Application，使用postman访问<http://127.0.0.1:8080/user/create>，参数

{"userName":"dd","age":120,"isFalse":true,"birthday":"21010-21-12"}



控制台显示



## 配置快速失败返回模式

**package** top.linyisong.config;

**import** javax.validation.Validation;

**import** javax.validation.Validator;

**import** javax.validation.ValidatorFactory;

**import** org.hibernate.validator.HibernateValidator;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

@Configuration

**public** **class** ValidatorConfiguration {

@Bean

**public** Validator validator(){

ValidatorFactory validatorFactory = Validation.*byProvider*( HibernateValidator.**class** )

.configure()

.addProperty( "hibernate.validator.fail\_fast", "true" )

.buildValidatorFactory();

Validator validator = validatorFactory.getValidator();

**return** validator;

}

}

重新启动Application后，访问程序，控制台显示



## 请求参数校验

验证请求参数时，在@RequestBody User user之间加注解 @Valid，然后后面加BindindResult即可；如：

@RequestMapping("/create")

**public** **void** create(@RequestBody @Valid User user, BindingResult result){

**if**(result.hasErrors()){

**for** (ObjectError error : result.getAllErrors()) {

System.***out***.println(error.getDefaultMessage());

}

}

}

## GET参数校验(@RequestParam参数校验)

使用校验bean的方式，没有办法校验RequestParam的内容，一般在处理Get请求(或参数比较少)的时候，会使用下面这样的代码：

@RequestMapping(value = "/demo3", method = RequestMethod.GET)

public void demo3(@RequestParam(name = "grade", required = true) int grade,@RequestParam(name = "classroom", required = true) int classroom) {

System.out.println(grade + "," + classroom);

}

使用@Valid注解，对RequestParam对应的参数进行注解，是无效的，需要使用@Validated注解来使得验证生效。如下所示：

a.此时需要使用MethodValidationPostProcessor 的Bean：

@Bean

public MethodValidationPostProcessor methodValidationPostProcessor() {  
　　　　 /\*\*默认是普通模式，会返回所有的验证不通过信息集合\*/

return new MethodValidationPostProcessor();

}

或 可对MethodValidationPostProcessor 进行设置Validator（因为此时不是用的Validator进行验证，Validator的配置不起作用）

[复制代码](javascript:void(0);)

@Bean

public MethodValidationPostProcessor methodValidationPostProcessor() {

MethodValidationPostProcessor postProcessor = new MethodValidationPostProcessor();  
　　　　　/\*\*设置validator模式为快速失败返回\*/

postProcessor.setValidator(validator());

return postProcessor;

}

@Bean

public Validator validator(){

ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.class )

.configure()

.addProperty( "hibernate.validator.fail\_fast", "true" )

.buildValidatorFactory();

Validator validator = validatorFactory.getValidator();

return validator;

}

[复制代码](javascript:void(0);)

b.方法所在的Controller上加注解@Validated

[复制代码](javascript:void(0);)

@RequestMapping("/validation")

@RestController

@Validated

public class ValidationController {

/\*\*如果只有少数对象，直接把参数写到Controller层，然后在Controller层进行验证就可以了。\*/

@RequestMapping(value = "/demo3", method = RequestMethod.GET)

public void demo3(@Range(min = 1, max = 9, message = "年级只能从1-9")

@RequestParam(name = "grade", required = true)

int grade,

@Min(value = 1, message = "班级最小只能1")

@Max(value = 99, message = "班级最大只能99")

@RequestParam(name = "classroom", required = true)

int classroom) {

System.out.println(grade + "," + classroom);

}

}

[复制代码](javascript:void(0);)

c.返回验证信息提示

可以看到：验证不通过时，抛出了ConstraintViolationException异常，使用同一捕获异常处理：

[复制代码](javascript:void(0);)

@ControllerAdvice

@Component

public class GlobalExceptionHandler {

@ExceptionHandler

@ResponseBody

@ResponseStatus(HttpStatus.BAD\_REQUEST)

public String handle(ValidationException exception) {

if(exception instanceof ConstraintViolationException){

ConstraintViolationException exs = (ConstraintViolationException) exception;

Set<ConstraintViolation<?>> violations = exs.getConstraintViolations();

for (ConstraintViolation<?> item : violations) {  
　　　　　　　　　　/\*\*打印验证不通过的信息\*/

System.out.println(item.getMessage());

}

}

return "bad request, " ;

}

}

[复制代码](javascript:void(0);)

d.验证

**浏览器服务请求地址：http://localhost:8080/user/**requestParam**?grade=18&classroom=888**

没有配置快速失败返回的MethodValidationPostProcessor 时输出信息如下：

年级只能从1-9

班级最大只能99

配置了快速失败返回的MethodValidationPostProcessor 时输出信息如下：

年级只能从1-9

**浏览器服务请求地址：**

**http://localhost:8080/user/**requestParam**?grade=0&classroom=0**

没有配置快速失败返回的MethodValidationPostProcessor 时输出信息如下：

年级只能从1-9  
班级最小只能1

配置了快速失败返回的MethodValidationPostProcessor 时输出信息如下：

年级只能从1-9

## model校验

待校验的model：

[复制代码](javascript:void(0);)

@Data

public class Demo2 {

@Length(min = 5, max = 17, message = "length长度在[5,17]之间")

private String length;

/\*\*@Size不能验证Integer，适用于String, Collection, Map and arrays\*/

@Size(min = 1, max = 3, message = "size在[1,3]之间")

private String age;

@Range(min = 150, max = 250, message = "range在[150,250]之间")

private int high;

@Size(min = 3,max = 5,message = "list的Size在[3,5]")

private List<String> list;

}

[复制代码](javascript:void(0);)

验证model，以下全部验证通过：

[复制代码](javascript:void(0);)

@Autowired

private Validator validator;

@RequestMapping("/demo3")

public void demo3(){

Demo2 demo2 = new Demo2();

demo2.setAge("111");

demo2.setHigh(150);

demo2.setLength("ABCDE");

demo2.setList(new ArrayList<String>(){{add("111");add("222");add("333");}});

Set<ConstraintViolation<Demo2>> violationSet = validator.validate(demo2);

for (ConstraintViolation<Demo2> model : violationSet) {

System.out.println(model.getMessage());

}

}

[复制代码](javascript:void(0);)

## 对象级联校验

对象内部包含另一个对象作为属性，属性上加@Valid，可以验证作为属性的对象内部的验证：（验证Demo2示例时，可以验证Demo2的字段）

[复制代码](javascript:void(0);)

**package** top.linyisong.model;

**import** java.util.List;

**import** javax.validation.Valid;

**import** javax.validation.constraints.NotNull;

**import** javax.validation.constraints.Size;

**import** org.hibernate.validator.constraints.Length;

**import** org.hibernate.validator.constraints.Range;

**public** **class** Demo2 {

@Length(min = 5, max = 17, message = "length长度在[5,17]之间")

**private** String length;

/\*\***@Size不能验证Integer**，适用于String, Collection, Map and arrays\*/

@Size(min = 1, max = 3, message = "size在[1,3]之间")

**private** String age;

@Range(min = 150, max = 250, message = "range在[150,250]之间")

**private** **int** high;

@Size(min = 3,max = 5,message = "list的Size在[3,5]")

**private** List<String> list;

@NotNull(message="级联对象不能为空")

@Valid

**private** Demo3 demo3;

**public** String getLength() {

**return** length;

}

**public** **void** setLength(String length) {

**this**.length = length;

}

**public** String getAge() {

**return** age;

}

**public** **void** setAge(String age) {

**this**.age = age;

}

**public** **int** getHigh() {

**return** high;

}

**public** **void** setHigh(**int** high) {

**this**.high = high;

}

**public** List<String> getList() {

**return** list;

}

**public** **void** setList(List<String> list) {

**this**.list = list;

}

**public** Demo3 getDemo3() {

**return** demo3;

}

**public** **void** setDemo3(Demo3 demo3) {

**this**.demo3 = demo3;

}

}

**package** top.linyisong.model;

**import** org.hibernate.validator.constraints.Length;

**public** **class** Demo3 {

@Length(min = 5, max = 17, message = "级联对象属性extField：length长度在[5,17]之间")

**private** String extField;

**public** String getExtField() {

**return** extField;

}

**public** **void** setExtField(String extField) {

**this**.extField = extField;

}

}

[复制代码](javascript:void(0);)

级联校验：

[复制代码](javascript:void(0);)

*/\*\*前面配置了快速失败返回的Bean\*/*

*@Autowired*

*private Validator validator;*

@RequestMapping("/model2")

**public** **void** model2(){

Demo2 demo2 = **new** Demo2();

demo2.setAge("111");

demo2.setHigh(150);

demo2.setLength("ABCDE");

List<String> list = **new** ArrayList<String>();

list.add("111");

list.add("222");

list.add("333");

demo2.setList(list);

Set<ConstraintViolation<Demo2>> violationSet = validator.validate(demo2);

**for** (ConstraintViolation<Demo2> model : violationSet) {

System.***out***.println(model.getMessage());

}

}

@RequestMapping("/model3")

**public** **void** model3(){

Demo2 demo2 = **new** Demo2();

demo2.setAge("111");

demo2.setHigh(150);

demo2.setLength("ABCDE");

List<String> list = **new** ArrayList<String>();

list.add("111");

list.add("222");

list.add("333");

demo2.setList(list);

Demo3 demo3 = **new** Demo3();

demo3.setExtField("ABC");

demo2.setDemo3(demo3);

Set<ConstraintViolation<Demo2>> violationSet = validator.validate(demo2);

**for** (ConstraintViolation<Demo2> model : violationSet) {

System.***out***.println(model.getMessage());

}

}[复制代码](javascript:void(0);)

可以校验Demo3的extField字段。

## 分组校验

**结论：分组顺序校验时，按指定的分组先后顺序进行验证，前面的验证不通过，后面的分组就不行验证。**

有这样一种场景，新增用户信息的时候，不需要验证userId（因为系统生成）；修改的时候需要验证userId，这时候可用用户到validator的分组验证功能。

设置validator为普通验证模式（**"hibernate.validator.fail\_fast", "false"**），用到的验证GroupA、GroupB和model：

GroupA、GroupB：

public interface GroupA {

}

public interface GroupB {

}

验证model：Person

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

@Data

public class Person {

@NotBlank

@Range(min = 1,max = Integer.MAX\_VALUE,message = "必须大于0",groups = {GroupA.class})

/\*\*用户id\*/

private Integer userId;

@NotBlank

@Length(min = 4,max = 20,message = "必须在[4,20]",groups = {GroupB.class})

/\*\*用户名\*/

private String userName;

@NotBlank

@Range(min = 0,max = 100,message = "年龄必须在[0,100]",groups={Default.class})

/\*\*年龄\*/

private Integer age;

@Range(min = 0,max = 2,message = "性别必须在[0,2]",groups = {GroupB.class})

/\*\*性别 0：未知；1：男；2：女\*/

private Integer sex;

}

[复制代码](javascript:void(0);)

如上Person所示，3个分组分别验证字段如下：

* GroupA验证字段userId；
* GroupB验证字段userName、sex；
* Default验证字段age(Default是Validator自带的默认分组)

### 分组

只验证GroupA、GroupB标记的分组：

[复制代码](javascript:void(0);)

@RequestMapping("/demo5")  
public void demo5(){  
 Person p = new Person();  
 /\*\*GroupA验证不通过\*/  
 p.setUserId(-12);  
 /\*\*GroupA验证通过\*/  
 //p.setUserId(12);  
 p.setUserName("a");  
 p.setAge(110);  
 p.setSex(5);  
 Set<ConstraintViolation<Person>> validate = validator.validate(p, GroupA.class, GroupB.class);  
 for (ConstraintViolation<Person> item : validate) {  
 System.out.println(item);  
 }  
}

[复制代码](javascript:void(0);)

或

[复制代码](javascript:void(0);)

@RequestMapping("/demo6")

public void demo6(@Validated({GroupA.class, GroupB.class}) Person p, BindingResult result){

if(result.hasErrors()){

List<ObjectError> allErrors = result.getAllErrors();

for (ObjectError error : allErrors) {

System.out.println(error);

}

}

}

[复制代码](javascript:void(0);)

GroupA、GroupB、Default都验证不通过的情况：

验证信息如下所示：

ConstraintViolationImpl{interpolatedMessage='必须在[4,20]', propertyPath=userName, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='必须在[4,20]'}  
ConstraintViolationImpl{interpolatedMessage='必须大于0', propertyPath=userId, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='必须大于0'}  
ConstraintViolationImpl{interpolatedMessage='性别必须在[0,2]', propertyPath=sex, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='性别必须在[0,2]'}

GroupA验证通过、GroupB、Default验证不通过的情况：

验证信息如下所示：

ConstraintViolationImpl{interpolatedMessage='必须在[4,20]', propertyPath=userName, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='必须在[4,20]'}  
ConstraintViolationImpl{interpolatedMessage='性别必须在[0,2]', propertyPath=sex, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='性别必须在[0,2]'}

### 组序列

除了按组指定是否验证之外，还可以指定组的验证顺序，前面组验证不通过的，后面组不进行验证：

指定组的序列（GroupA》GroupB》Default）：

@GroupSequence({GroupA.class, GroupB.class, Default.class})

public interface GroupOrder {

}

测试demo：

[复制代码](javascript:void(0);)

@RequestMapping("/demo7")

public void demo7(){

Person p = new Person();

/\*\*GroupA验证不通过\*/

//p.setUserId(-12);

/\*\*GroupA验证通过\*/

p.setUserId(12);

p.setUserName("a");

p.setAge(110);

p.setSex(5);

Set<ConstraintViolation<Person>> validate = validator.validate(p, GroupOrder.class);

for (ConstraintViolation<Person> item : validate) {

System.out.println(item);

}

}

[复制代码](javascript:void(0);)

或

[复制代码](javascript:void(0);)

@RequestMapping("/demo8")

public void demo8(@Validated({GroupOrder.class}) Person p, BindingResult result){

if(result.hasErrors()){

List<ObjectError> allErrors = result.getAllErrors();

for (ObjectError error : allErrors) {

System.out.println(error);

}

}

}

[复制代码](javascript:void(0);)

GroupA、GroupB、Default都验证不通过的情况：

验证信息如下所示：

ConstraintViolationImpl{interpolatedMessage='必须大于0', propertyPath=userId, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='必须大于0'}

GroupA验证通过、GroupB、Default验证不通过的情况：

验证信息如下所示：

ConstraintViolationImpl{interpolatedMessage='必须在[4,20]', propertyPath=userName, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='必须在[4,20]'}  
ConstraintViolationImpl{interpolatedMessage='性别必须在[0,2]', propertyPath=sex, rootBeanClass=class validator.demo.project.model.Person, messageTemplate='性别必须在[0,2]'}

**结论：分组顺序校验时，按指定的分组先后顺序进行验证，前面的验证不通过，后面的分组就不行验证。**

## 自定义验证器

一般情况，自定义验证可以解决很多问题。但也有无法满足情况的时候，此时，我们可以实现validator的接口，自定义自己需要的验证器。

如下所示，实现了一个自定义的大小写验证器：

[复制代码](javascript:void(0);)

public enum CaseMode {

UPPER,

LOWER;

}

@Target( { ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION\_TYPE })

@Retention(RetentionPolicy.RUNTIME)

@Constraint(validatedBy = CheckCaseValidator.class)

@Documented

public @interface CheckCase {

String message() default "";

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

Class<? extends Payload>[] payload() default {};

CaseMode value();

}

public class CheckCaseValidator implements ConstraintValidator<CheckCase, String> {

private CaseMode caseMode;

public void initialize(CheckCase checkCase) {

this.caseMode = checkCase.value();

}

public boolean isValid(String s, ConstraintValidatorContext constraintValidatorContext) {

if (s == null) {

return true;

}

if (caseMode == CaseMode.UPPER) {

return s.equals(s.toUpperCase());

} else {

return s.equals(s.toLowerCase());

}

}

}

[复制代码](javascript:void(0);)

要验证的Model：

[复制代码](javascript:void(0);)

public class Demo{

@CheckCase(value = CaseMode.LOWER,message = "userName必须是小写")

private String userName;

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

}

[复制代码](javascript:void(0);)

validator配置：

[复制代码](javascript:void(0);)

@Bean

public Validator validator(){

ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.class )

.configure()

.addProperty( "hibernate.validator.fail\_fast", "true" )

.buildValidatorFactory();

Validator validator = validatorFactory.getValidator();

return validator;

}

[复制代码](javascript:void(0);)

验证测试：

[复制代码](javascript:void(0);)

@RequestMapping("/demo4")

public void demo4(){

Demo demo = new Demo();

demo.setUserName("userName");

Set<ConstraintViolation<Demo>> validate = validator.validate(demo);

for (ConstraintViolation<Demo> dem : validate) {

System.out.println(dem.getMessage());

}

}

[复制代码](javascript:void(0);)

输出结果：

userName必须是小写

## 常见的注解

  Bean Validation 中内置的 constraint       
@Null   被注释的元素必须为 null       
@NotNull    被注释的元素必须不为 null       
@AssertTrue     被注释的元素必须为 true       
@AssertFalse    被注释的元素必须为 false       
@Min(value)     被注释的元素必须是一个数字，其值必须大于等于指定的最小值       
@Max(value)     被注释的元素必须是一个数字，其值必须小于等于指定的最大值       
@DecimalMin(value)  被注释的元素必须是一个数字，其值必须大于等于指定的最小值       
@DecimalMax(value)  被注释的元素必须是一个数字，其值必须小于等于指定的最大值       
@Size(max=, min=)   被注释的元素的大小必须在指定的范围内       
@Digits (integer, fraction)     被注释的元素必须是一个数字，其值必须在可接受的范围内       
@Past   被注释的元素必须是一个过去的日期       
@Future     被注释的元素必须是一个将来的日期       
@Pattern(regex=,flag=)  被注释的元素必须符合指定的正则表达式       
Hibernate Validator 附加的 constraint       
@NotBlank(message =)   验证字符串非null，且长度必须大于0       
@Email  被注释的元素必须是电子邮箱地址       
@Length(min=,max=)  被注释的字符串的大小必须在指定的范围内       
@NotEmpty   被注释的字符串的必须非空       
@Range(min=,max=,message=)  被注释的元素必须在合适的范围内

//大于0.01，不包含0.01  
@NotNull  
@DecimalMin(value = "0.01", inclusive = false)  
private Integer greaterThan;  
  
//大于等于0.01  
@NotNull  
@DecimalMin(value = "0.01", inclusive = true)  
private BigDecimal greatOrEqualThan;  
  
@Length(min = 1, max = 20, message = "message不能为空")  
//不能将Length错用成Range  
//@Range(min = 1, max = 20, message = "message不能为空")  
private String message;

# 实际项目使用

## 配置

import javax.validation.Validation;

import javax.validation.Validator;

import javax.validation.ValidatorFactory;

import org.hibernate.validator.HibernateValidator;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.validation.beanvalidation.MethodValidationPostProcessor;

@Configuration

public class ValidatorConfiguration {

@Bean

public MethodValidationPostProcessor methodValidationPostProcessor() {

MethodValidationPostProcessor postProcessor = new MethodValidationPostProcessor();

postProcessor.setValidator(validator());

return postProcessor;

}

@Bean

public Validator validator(){

ValidatorFactory validatorFactory = Validation.byProvider( HibernateValidator.class )

.configure()

.addProperty( "hibernate.validator.fail\_fast", "true" )

.buildValidatorFactory();

Validator validator = validatorFactory.getValidator();

return validator;

}

}

## Model

@ApiModel(value="国家法定节假日")

@TableName("t\_statutory\_holiday")

**public** **class** StatutoryHoliday **implements** Serializable {

@TableField(exist = **false**)

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

/\*\*

\* id

\*/

@TableId

@ApiModelProperty(value="id")

@NotNull(message ="id不能为空",groups={Update.**class**})

**private** Long id;

/\*\*

\* 是否法定假期1是0否

\*/

@ApiModelProperty(notes="是否法定假期：1是0否",required=**true**,allowableValues="1,0",example="1")

@NotNull(message ="是否法定假期不能为空",groups={Create.**class**,Update.**class**})

**private** Integer isStatutoryHoliday;

/\*\*

\* 假期名称

\*/

@ApiModelProperty(value="假期名称",example="清明节")

**private** String holidayName;

/\*\*

\* 假期类别

\*/

@ApiModelProperty(value="假期类别：STATUTORY假日SUPPLEMENTARY补班",required=**true**,allowableValues="STATUTORY,SUPPLEMENTARY",example="STATUTORY")

@NotBlank(message ="假期类别不能为空",groups={Create.**class**,Update.**class**})

**private** String vacationCategory;

/\*\*

\* 假期日期

\*/

@ApiModelProperty(value="假期日期",required=**true**,example="2019-05-01")

@NotBlank(message ="假期日期不能为空",groups={Create.**class**,Update.**class**})

**private** String vacationDate;

/\*\*

\* 创建时间

\*/

@ApiModelProperty(value="创建时间",required=**true**,example="2019-01-01")

@NotBlank(message ="创建时间不能为空",groups={Create.**class**,Update.**class**})

**private** String createTime;

**public** **interface** Create {

}

**public** **interface** Update {

}

}

## Controller

@Validated

**public** **class** StatutoryHolidayController **extends** BaseController {

@Log("新增国家法定节假日")

@ApiOperation(value = "新增国家法定节假日")

@PostMapping("/add")

@ResponseBody

Result<String> add(@RequestBody @Validated(StatutoryHoliday.Create.**class**) StatutoryHoliday statutoryHoliday) {

String vacationDate = statutoryHoliday.getVacationDate();//假期日期

Wrapper<StatutoryHoliday> wrapper = **new** EntityWrapper<StatutoryHoliday>();

wrapper.eq("vacation\_date", vacationDate);

StatutoryHoliday oldM = statutoryHolidayService.selectOne(wrapper);

**if**(oldM!=**null**){

**return** Result.*fail*("假期日期已存在");

}

statutoryHolidayService.insert(statutoryHoliday);

**return** Result.*ok*();

}

}

## 全局异常捕捉

@RestControllerAdvice

**public** **class** ApplicationExceptionHandler {

@ExceptionHandler(MethodArgumentNotValidException.**class**)

**public** Result<String> handleMethodArgumentNotValidException(org.springframework.web.bind.MethodArgumentNotValidException e) {

/\* log.error(e.getMessage(), e);\*/

StringBuilder sb = **new** StringBuilder();

BindingResult result = e.getBindingResult();

**if**(result.hasErrors()){

**for** (ObjectError error : result.getAllErrors()) {

sb.append(sb.toString().length()==0?error.getDefaultMessage():","+error.getDefaultMessage());

}

}

**return** Result.*build*(EnumErrorCode.***badRequest***.getCode(),**false**, sb.toString());

}

}

**public** **static** <T> Result<T> build(Integer code,Boolean success, String msg) {

**return** **new** Result<T>(code,success, msg, **null**);

}