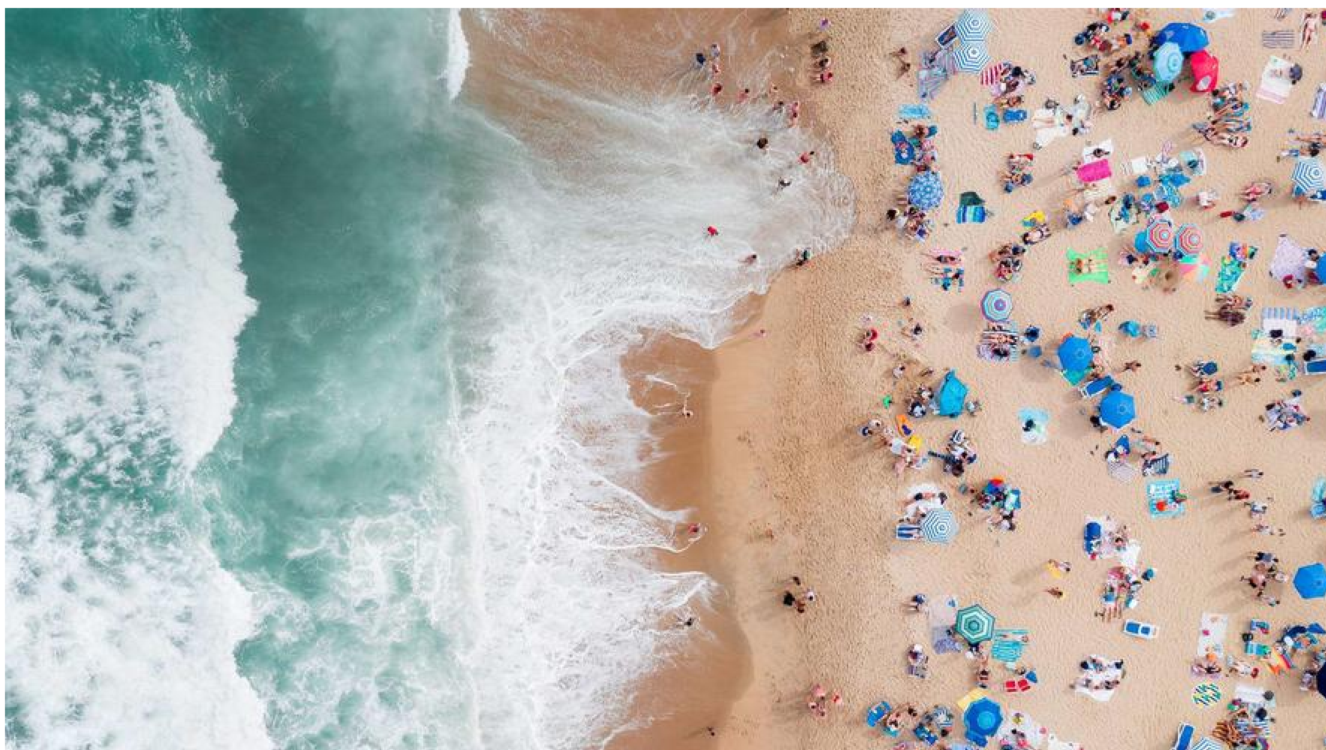
链滴

# javax.validation 参数校验

作者：zhaozhizheng
原文链接：https://ld246.com/article/1631008666145
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



# 参数校验往往让人心塞，各种判断匹配，基本的参数校验例
# 实体校验可以通过如下方法解决，但是因为我们的请求响应
# 果都是加密字符，需要解密后才能校验，所以在此的基础上
# 加第二段的方式，通过封装工具类的方法完成。

---

版权声明：本文为CSDN博主「LailaiMonkey」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请
上原文出处链接及本声明。

原文链接：https://blog.csdn.net/h273979586/article/details/105572872

如果是Spring项目那么这个架构依赖会自动引用，如果是非Spring项目得手动引用一下依赖。本文以
pring项目为主：

```xml
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.16.Final</version>
    <scope>compile</scope>
</dependency>
```

这是注解包里面元素：

@AssertFalse 被注释的元素必须为 false

@AssertTrue 被注释的元素必须为 true

@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

@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

@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

@Email 被注释的元素必须是电子邮箱地址

@Future 被注释的元素必须是一个将来的日期

@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

@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

@NotBlank 验证注解的元素值不为空（不为null并去除首尾位空格）

@NotEmpty 验证注解的元素值不为null且不为空（字符串长度不为0、集合大小不为0）

@NotNull 不为null

@Null 为null

@Past 限制必须是一个过去的日期

@Pattern(value) 限制必须符合指定的正则表达式

@Size(max,min) `B限制字符长度必须在min到max之间

验证规则

在Spring项目中定义一个需要验证的model内容如下：

@Data

public class ValidatorModel {

```
@NotBlank(message = "姓名不能为空")
    private String name;

    @Size(min = 3, max = 6)
    private List<String> friendNames;

    @Min(value = 18, message = "年龄未满18岁")
    @Max(value = 60, message = "年龄必须在60岁以下")
    private Integer age;

    private String tel;

    @DecimalMin(value = "30", message = "金钱不能小于30")
    private BigDecimal money;
```

service接口内容如下：

因为是javax的验证构架所以在没有Spring时候也可以进行验证，注解得写在接口上，如果写在实现将无效！！！

```
//开启验证注解
@Validated
public interface ValidatorService {
    void validator(@Valid ValidatorModel model, @NotNull String param);
}
```

service实现类就是正常逻辑处理这里不在演示了。

controller内容如下：

原文链接：javax.validation 参数校验

controller接收参数传递给service，在service接口进行参数验证，如果失败将抛出异常。

```
@RestController
public class ValidatorController {

    @Autowired
    private ValidatorService validatorService;

    @GetMapping
    public void validator() {
        ValidatorModel model = new ValidatorModel();
        model.setName("张三");
        model.setFriendNames(Arrays.asList("李四", "王五"));
        model.setAge(20);
        model.setTel("188888888881");
        model.setMoney(new BigDecimal(100));
        validatorService.validator(model,"也可以在参数上做验证");
    }
}
```

异常截图：

以上是Javax Validation校验常用方法，可以满足多数情况，但是项目中遇到的都是少数情况，例如：

添加时候需要对所有参数进行验证，更新时候只需要对某个几参数或有值的参数进行验证（分组验证
。

可不可以自定义验证规则，我要验证性别，只能是男或女（自定义验证规则）。

分组验证

添加时候需要验证friendNames字段，更新时候需要验证name字段，这时得引用分组验证，修改ValiatorModel如下：

```
@Data
public class ValidatorModel {

    /**
     * 添加分组
     */
    public interface Create {
    }

    /**
     * 更新分组
     */
    public interface Update {
    }

    /**
     * 分组可以指定多个，用{}表示
     * 分组为：默认分组、更新分组
     */
    @NotBlank(message = "姓名不能为空", groups = {Default.class, Update.class})
    private String name;

    /**
```

```
     * 分组为添加分组
     */
    @Size(min = 3, max = 6, groups = {Create.class})
    private List<String> friendNames;

    @Min(value = 18, message = "年龄未满18岁")
    @Max(value = 60, message = "年龄必须在60岁以下")
    private Integer age;

    private String tel;

    @DecimalMin(value = "30", message = "金钱不能小于30")
    private Integer money;
}
```

把校验参数注解写在Controller上，别忘了在Controller上加@Validated注解哈！！！

```
@Validated
@RestController
public class ValidatorController {

    @Autowired
    private ValidatorService validatorService;

    @GetMapping
    public void validator() {
        ValidatorModel model = new ValidatorModel();
        model.setName("张三");
        model.setFriendNames(Arrays.asList("李四", "王五"));
        model.setAge(2);
        model.setTel("188888888881");
        model.setMoney(100);
        model.setEmail("a");
        validatorService.validator(model, "也可以在参数上做验证");
    }

    @GetMapping("/group")
    public void validatorGroup(@Validated({ValidatorModel.Create.class}) ValidatorModel mod
l) {
        validatorService.validatorGroup(model, "也可以在参数上做验证");
    }
}
```

这样就可以实现分组验证了。

注意：model里有一个默认分组，如果指定分组就按分组来验证参数，而没有指定的分组属于默认分，上图的Controller不会对默认分组进行验证，写成@Validated({ValidatorModel.Create.class, Default.class})把默认分组加进去就可以验证了。如下：

```
@GetMapping("/group")
    public void validatorGroup(@Validated({ValidatorModel.Create.class, Default.class}) Validat
rModel model) {
        validatorService.validatorGroup(model, "也可以在参数上做验证");
    }
```

原文链接：javax.validation 参数校验

自定义验证规则

那如果我想验证手机怎么办呢？它里面没有提供，不要慌小朋友，使用自定义注解验证就可以了，它供了接口我们只需要实现一下就好了。

首先需要我们自定义一个验证手机号的注解（MonkeyTel），并指定其实现类，在实现写验证的逻辑可以了。注解如下：

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.PARAMETER, ElementType.FIELD})
// 指定真正实现校验规则的类
@Constraint(validatedBy = MonkeyJavaTelImpl.class)
public @interface MonkeyJavaTel {
    String message() default "手机号必须是11位";

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

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

实现类如下：

```
/**
 * @Author: LailaiMonkey
 * @Description：
 * String：参数类型
 * @Date：Created in 2020-04-17 15:19
 * @Modified By：
 */
 public class MonkeyJavaTelImpl implements ConstraintValidator<MonkeyJavaTel, String> {

    /**
     * 手机号是十一位为ture
     * @param value 值
     * @param context
     * @return
     */
     @Override
     public boolean isValid(String value, ConstraintValidatorContext context) {
        return  value.length() == 11;
     }
 }
```

项目结构如下：

忽略monkeyValidator包，自定义注解校验规则，有兴趣的小伙伴可以看下一篇博客：自定义校验注框架自定义校验注解框架

自定义验证规则

缺点：

此注解只能写有接口或Controller上才管用，如果写在实现类上是无效的。而且逻辑都写在实现类上不知道哪些参数进行验证，还得返回到接口或Controller上查询才可以。

不支持List集合验证，实现起来困难，只支持简单model验证。

原文链接：javax.validation 参数校验

分组验证只能写在Controller上，写在接口不起作用（坑）。

如果一个接口有多个实现类这个注解就不能再用了。

优点：

自定义注解实现简单，上手快。

使用工具类校验

校验工具类：

```
package com.alibaba.banff.web.util;

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

import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.validation.groups.Default;

/**
 * 校验工具类
 *
 * @author lizhilong
 */
public class ValidationUtils {

    private static Validator validator = Validation.buildDefaultValidatorFactory().getValidator();

    public static <T> ValidationResult validateEntity(T obj) {
        ValidationResult result = new ValidationResult();
        Set<ConstraintViolation<T>> set = validator.validate(obj, Default.class);
        // if( CollectionUtils.isNotEmpty(set) ){
        if (set != null && set.size() != 0) {
            result.setHasErrors(true);
            Map<String, String> errorMsg = new HashMap<String, String>();
            for (ConstraintViolation<T> cv : set) {
                errorMsg.put(cv.getPropertyPath().toString(), cv.getMessage());
            }
            result.setErrorMsg(errorMsg);
        }
        return result;
    }

    public static <T> ValidationResult validateProperty(T obj, String propertyName) {
        ValidationResult result = new ValidationResult();
        Set<ConstraintViolation<T>> set = validator.validateProperty(obj, propertyName, Defau
t.class);
        if (set != null && set.size() != 0) {
            result.setHasErrors(true);
            Map<String, String> errorMsg = new HashMap<String, String>();
            for (ConstraintViolation<T> cv : set) {
                errorMsg.put(propertyName, cv.getMessage());
```

```
            }
            result.setErrorMsg(errorMsg);
        }
        return result;
    }
}
```

校验工具类返回的数据ValidationResult（省略getset）：

```
package com.aliyun.prophet.facade.partner.flaw;

import java.util.Map;

/**
 * 校验结果
 *
 * @author lizhilong
 */
public class ValidationResult {

    // 校验结果是否有错
    private boolean             hasErrors;

    // 校验错误信息
    private Map<String, String> errorMsg;
}
```

Person类（省略getset）：

```
package com.aliyun.prophet.facade.partner.flaw;

import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.NotEmpty;
import org.hibernate.validator.constraints.Range;

public class Person {
    @Length(max=20,message="姓名长度不能大于20")
    @NotEmpty(message="姓名不能为空")
    private String name;
    @Range(min = 0, max = 1, message = "性别只能输入只能输入0或1")
    private Integer gender;
    private Integer age;
}
```

使用方法：

```
/**
 *
 * @author: lizhilong
 */
public class Test {
    @org.junit.Test
    public void testValidation(){
        Person person = new Person();
        person.setAge(12);
```

```
        person.setGender(2);
//      person.setName("李智龙");
        ValidationResult result = ValidationUtils.validateEntity(person);
        Map<String, String> map = result.getErrorMsg();
        boolean isError = result.isHasErrors();
        System.out.println("isError: " +isError);
        System.out.println(map);
    }
}
```

打印结果：

```
isError: true
{gender=性别只能输入只能输入0或1, name=姓名不能为空}
```

是不是很简单呀，几步搞定，根本不用spring便可封装一个好用的工具类

---

版权声明：本文为CSDN博主「熬夜是小狗」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上文出处链接及本声明。
原文链接：https://blog.csdn.net/icannotdebug/article/details/78708202