SpringBoot如何优雅的校验参数

日拱一兵 2019-12-20

编者荐语：

如题，良好的API入口数据校验，是保证数据有效性的前提

以下文章来源于乱敲代码 ，作者乱敲代码

乱敲代码

专注分享Java开发技术相关文章，各种工具资源!

点击**阅读原文**更好的阅读体验

前言

做web开发有一点很烦人就是要校验参数，基本上每个接口都要对参数进行校验，比如一些格式校验 非空校验都是必不可少的。如果参数比较少的话还是容易 处理的一但参数比较多了的话代码中就会出现大量的 `IF ELSE` 就比如下面这样：

```
if(StringUtils.isBlank(userDTO.getUserName())){
    return ReturnVO.error("用户名不能为空");
}
if(StringUtils.isBlank(userDTO.getMobileNo())){
    return ReturnVO.error("手机号不能为空");
}
if(StringUtils.isBlank(userDTO.getEmail())){
    return ReturnVO.error("手机号不能为空");
}
if(null==userDTO.getAge()){
    return ReturnVO.error("年龄不能为空");
}
if(null==userDTO.getSex()){
    return ReturnVO.error("性别不能为空");
}
```

这个例子只是校验了一下空参数。如果需要验证邮箱格式和手机号格式校验的话代码会更多，所以介绍一下 `validator` 通过注解的方式进行校验参数。

什么是Validator

Bean Validation是Java定义的一套基于注解的数据校验规范，目前已经从JSR 303的1.0版本升级到JSR 349的1.1版本，再到JSR 380的2.0版本（2.0完成于2017.08），已经经历了三个版本 。在 `SpringBoot` 中已经集成在 `starter-web` 中，所以无需在添加其他依

赖。

```
<!--版本自行控制，这里只是简单举例-->
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.0.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.1.Final</version>
</dependency>
```

注解介绍

validator内置注解

注解	详细信息
`@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(value)`	被注释的元素必须符合指定的正则表达式

Hibernate Validator 附加的 constraint

注解	详细信息
`@Email`	被注释的元素必须是电子邮箱地址
`@Length`	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`	被注释的元素必须在合适的范围内
`@NotBlank`	验证字符串非null，且长度必须大于0

注意:

- @NotNull 适用于任何类型被注解的元素必须不能与NULL
- @NotEmpty 适用于String Map或者数组不能为Null且长度必须大于0
- @NotBlank 只能用于String上面 不能为null,调用trim()后，长度必须大于0

使用

使用起来也非常简单，下面略过创建项目

模拟用户注册封装了一个 `UserDTO`

当提交数据的时候如果使用以前的做法就是 `IF ELSE` 判断参数使用 `validator` 则是需要增加注解即可。

例如非空校验:

```java
public class UserDTO {
    /**
     * 用户名
     */
    @NotBlank(message = "用户姓名不能为空")
    @NotNull(message = "用户姓名不能为空")
    private String userName;
    /**
     * 手机号
     */
    @NotBlank(message = "手机号不能为空")
    @NotBlank(message = "手机号不能为空")
    private String mobileNo;
    /**
     * 性别
     */
    @NotNull(message = "性别不能为空")
    private Integer sex;
    /**
     * 年龄
     */
    @NotNull(message = "年龄不能为空")
    private Integer age;
    /**
     * 邮箱
     */
    @NotBlank(message = "邮箱不能为空")
    @NotNull(message = "邮箱不能为空")
    private String email;
```

然后需要在 `controller` 方法体添加 `@Validated` 不加 `@Validated` 校验会不起作用

```java
@PostMapping("/user")
public ReturnVO userRegistra(@RequestBody @Validated UserDTO userDTO){
    ReturnVO returnVO = userService.userRegistra(userDTO);
    return returnVO ;
}
```

然后请求一下请求接口,把Email参数设置为空

参数:

```
{
        "userName":"luomengsun",
        "mobileNo":"11111111111",
        "sex":1,
        "age":21,
        "email":""
}
```

返回结果：

后台抛出异常

这样是能校验成功，但是有个问题就是返回参数并不理想，前端也并不容易处理返回参数，所以我们添加一下全局异常处理，然后添加一下全局统一返回参数这样比较规范。

添加全局异常

创建一个 `GlobalExceptionHandler` 类，在类上方添加 `@RestControllerAdvice` 注解然后添加以下代码：

```
/**
     * 方法参数校验
     */
@ExceptionHandler(MethodArgumentNotValidException.class)
public ReturnVO handleMethodArgumentNotValidException(MethodArgumentNotValidException e) {
        log.error(e.getMessage(), e);
```

```
            returnnew ReturnVO().error(e.getBindingResult().getFieldError().getDefaultMessage());
        }
```

此方法主要捕捉 MethodArgumentNotValidException 异常然后对异常结果进行封装，如果需要在自行添加其他异常处理。

添加完之后我们在看一下运行结果，调用接口返回:

```
{
    "code": "9999",
    "desc": "邮箱不能为空",
    "data": null
}
```

OK 已经对异常进行处理。

校验格式

如果想要校验邮箱格式或者手机号的话也非常简单。

校验邮箱

```
/**
     * 邮箱
     */
    @NotBlank(message = "邮箱不能为空")
    @NotNull(message = "邮箱不能为空")
    @Email(message = "邮箱格式错误")
    private String email;
```

使用正则校验手机号

校验手机号使用正则进行校验，然后限制了一下位数

```
/**
     * 手机号
     */
    @NotNull(message = "手机号不能为空")
    @NotBlank(message = "手机号不能为空")
    @Pattern(regexp ="^[1][3,4,5,6,7,8,9][0-9]{9}$", message = "手机号格式有误")
    @Max(value = 11,message = "手机号只能为{max}位")
    @Min(value = 11,message = "手机号只能为{min}位")
    private String mobileNo;
```

查看一下运行结果

传入参数:

```
{
        "userName":"luomengsun",
        "mobileNo":"111111a",
        "sex":1,
        "age":21,
```

```
        "email":"1212121"
}
```

返回结果：

```
{
    "code": "9999",
    "desc": "邮箱格式错误",
    "data": null
}
```

这里不再验证手机号的例子

自定义注解

上面的注解只有这么多，如果有特殊校验的参数我们可以使用 `Validator` 自定义注解进行校验

首先创建一个 `IdCard` 注解类

```
@Documented
@Target({ElementType.PARAMETER, ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = IdCardValidator.class)
public@interface IdCard {

    String message() default "身份证号码不合法";

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

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

}
```

在UserDTO中添加 `@IdCard` 注解即可验证，在运行时触发，本文不对自定义注解做过多的解释，下篇文章介绍自定义注解

- message 提示信息
- groups 分组
- payload 针对于Bean

然后添加 `IdCardValidator` 主要进行验证逻辑

```java
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class IdCardValidator implements ConstraintValidator<IdCard, Object> {
    @Override
    public void initialize(IdCard constraintAnnotation) {
    }
    @Override
    public boolean isValid(Object o, ConstraintValidatorContext constraintValidatorContext) {
        return IdCardValidator.is18ByteIdCardComplex(o.toString());
    }
```

上面调用了 `is18ByteIdCardComplex` 方法，传入参数就是手机号，验证身份证规则自行百度:see_no_evil:

然后使用

```java
@NotNull(message = "身份证号不能为空")
    @IdCard(message = "身份证不合法")
    private String IdCardNumber;
```

分组

就比如上面我们定义的UserDTO中的参数如果要复用的话怎么办?

在重新定义一个类然后里面的参数要重新添加注解?

`Validator` 提供了分组方法完美了解决DTO服用问题

现在我们注册的接口修改一下规则，只有用户名不能为空其他参数都不进行校验

先创建分组的接口

```java
public interface Create  extends Default {
}
```

我们只需要在注解加入分组参数即可例如:

```java
/**
     * 用户名
     */
    @NotBlank(message = "用户姓名不能为空",groups = Create.class)
    @NotNull(message = "用户姓名不能为空",groups = Create.class)
    private String userName;

    @NotBlank(message = "邮箱不能为空",groups = Update.class)
    @NotNull(message = "邮箱不能为空",groups = Update.class)
    @Email(message = "邮箱格式错误",groups = Update.class)
    private String email;
```

然后在修改Controller在 `@Validated` 中传入 `Create.class`

```
@PostMapping("/user")
    public ReturnVO userRegistra(@RequestBody @Validated(Create.class) UserDTO userDTO){
        ReturnVO returnVO = userService.userRegistra(userDTO);
        return returnVO ;
    }
```

然后调用传入参数：

```
{
       "userName":"",
}
```

返回参数：

```
{
    "code": "9999",
    "desc": "用户姓名不能为空",
    "data": null
}
```

OK 现在只对Create的进行校验，而Updata组的不校验，如果需要复用DTO的话可以使用分组校验

校验单个参数

在开发的时候一定遇到过单个参数的情况,在参数前面加上注解即可

```
@PostMapping("/get")
    public ReturnVO getUserInfo(@RequestParam("userId") @NotNull(message = "用户ID不能为空") String userId){
        returnnew ReturnVO().success();
    }
```

然后在Controller类上面增加 `@Validated` 注解,注意不是增加在参数前面。