springboot 之前端参数验证

作者：hjljy
原文链接：https://ld246.com/article/1587985688318
来源网站：链滴

对于任何一个应用而言，在客户端做的数据有效性验证主要目的是规范用户的输入，而真实的数据验工作都是在服务后端代码当中实现的，但在实际的项目当中，也经常会因为各种各样的原因：懒得写觉得前端验证了，后端没有太多的必要等等没有进行数据验证，其实养成数据的有效性验证是一个非好的习惯。

1 可以避免很多数据有效性导致的BUG，防范其余开发者的基础攻击

2 在前后端进行接口联调的时候，不需要因为参数的问题沟通很久。

springboot 参数验证

JSR-303 是 JAVA EE 6 中的一项子规范，叫做 Bean Validation，官方参考实现是Hibernate Validat r。JSR 303 用于对 Java Bean 中的字段的值进行验证。 主要是 javax.validation 包下面的注解，用进行参数的验证。在 spring-boot当中存在 hibernate-validator 验证包，这个包里面包含了一些 jav x.validation 没有的注解。算是spring对于JSR验证的的扩展吧!

常用验证注解：

注解	用法
@NotNull	限制必须不为null
@Null	限制必须为null
@NotEmpty（字符串长度不为0、集合大小不为0）	验证注解的元素值不为 null 且不为
@NotBlank时会去除字符串的空格	@NotBlank只应用于字符串且在比
@Size(min,max)min 到 max 之间	限制字符串或者集合长度必须在
@Max(value)	限制必须为一个不大于指定值的数字

原文链接：springboot 之前端参数验证

@Min(value) 限制必须为一个不小于指定值的数字

@Past 限制必须是一个过去的日期

@Future 限制必须是一个将来的日期

@Email 验证注解的元素值是Email，可以通过正表达式和flag指定自定义的email格式

@Pattern(value) 限制必须符合指定的正则表达式

参数验证具体使用

1 创建需要验证的实体类

```
/**
 * @author 海加尔金鹰
 */
@Data
public class TestVo {

    @NotNull(message = "id 不能为空")
    private Integer id;

    @NotBlank(message = "name 不能为空字符串")
    private String name;

    @NotEmpty(message = "empty不能为空集合")
    private List<String> empty;

    @Max(value = 99,message = "排序最大值不能超过99")
    private int sort;
}
```

2 创建对应的请求接口 在需要校验的参数上加上@valid注解或者加上@Validated 注解 备注(由于测试所有这里不加上BindingResult 参数)

```
/**
 * @author 海加尔金鹰
 */
@RestController
public class TestController {
    @GetMapping("/id")
    public TestVo getTestVo(@RequestBody @Valid TestVo vo){
        return vo;
    }
}
```

3 发送请求查询返回的信息

```
GET http://localhost:8080/id
Content-Type: application/json

{
 "name" : "content",
```

```
 "sort": 55
}

  "errors": [
   {
     "codes": [
       "NotEmpty.testVo.empty",
       "NotEmpty.empty",
       "NotEmpty.java.util.List",
       "NotEmpty"
     ],
     "arguments": [
       {
         "codes": [
           "testVo.empty",
           "empty"
         ],
         "arguments": null,
         "defaultMessage": "empty",
         "code": "empty"
       }
     ],
     "defaultMessage": "empty不能为空集合",
     "objectName": "testVo",
     "field": "empty",
     "rejectedValue": null,
     "bindingFailure": false,
     "code": "NotEmpty"
   },
   {
     "codes": [
       "NotNull.testVo.id",
       "NotNull.id",
       "NotNull.java.lang.Integer",
       "NotNull"
     ],
     "arguments": [
       {
         "codes": [
           "testVo.id",
           "id"
         ],
         "arguments": null,
         "defaultMessage": "id",
         "code": "id"
       }
     ],
     "defaultMessage": "id 不能为空",
     "objectName": "testVo",
     "field": "id",
     "rejectedValue": null,
     "bindingFailure": false,
     "code": "NotNull"
   }
```

```
]
```

springboot参数的验证就成功的实现了。很简单，不复杂。

注意事项

● @valid 这个注解是JSR-303 规范原生的验证注解　@Validated 注解是spring针对@valid 进行的个封装，提供了一些额外的功能。

● 如果在接口上面加上了BindingResult 这个参数的话，验证后的错误信息不会抛出来，会被封装到个类当中。 如果需要获取到验证的错误信息，需要从这个类手动当中获取。

● @Max　@Min 在对包装类型进行验证的时候，如果包装类为null，是可以通过验证的，需要配合NotNull注解一起使用

springboot参数通过切面进行统一验证返回

在测试用例当中，返回的数据格式非常不友好，通常实际情况下都是通过切面的方式，获取BindingResult 参数的数据，如果有验证错误信息，就返回给前端参数相关的错误的信息

```
/**
 * @author 海加尔金鹰
 */
@Aspect
@Component
public class BindingResultAspect {

    @Around("execution(* cn.hjljy.springboot.validdemo.controller.*.*(..)) && args(..,bindingResult)")
    public Object validateParam(ProceedingJoinPoint joinPoint, BindingResult bindingResult) throws Throwable {
        Object obj = null;
        if (bindingResult.hasErrors()) {
            // 有校验错误
            System.out.println(bindingResult.getAllErrors().toString());
            return "这里返回错误的信息";
        } else {
            // 没有错误方法继续执行
            obj = joinPoint.proceed();
        }
        return obj;
    }
}
```