1 介绍

1.1 什么是数据校验

- 什么是数据校验
  - 前台校验（客户端校验）：
    - 在前台页面中用js校验填写在表单中的参数是否合法
  - 后台校验（服务端校验）：
    - Controller层：校验前台页面提交过来的参数的合法性
    - Service层：校验service接口中使用的参数
    - DAO层：一般不校验
- 工作中，我们首先必须对方法传递过来的参数进行合法性校验，如果参数不合法，那么我们就使用抛异常的方式，告知方法的调用者传递的参数有问题。
  这也是Validated/Valid数据校验的本质。

1.2 Java提供的数据校验工具

- JSR-303、JSR-349、validation-api、hibernate-validator、Spring之间的关系
  - JSR-303是一项标准、一项规范，JSR-349是其升级版本，添加了一些新特性。他们规定一些校验规范即校验注解如@Null、@NotNull、@Pattern，他们位于javax.validation.constraints包下，只提供规范不提供实现，说白了就是一堆接口没有实现类，这堆接口放在validation-api.jar中。
  - hibernate-validator.jar是对这个规范的实现（这个包和数据库没有任何关系），并增加了一些其他校验注解，如@Email、@Length、@Range等，他们位于org.hibernate.validator.constraints包下。
  - Spring为了给开发者提供便捷，对hibernate-validator.jar进行了二次封装，让数据校验更加便捷、提供了一些新功能。

1.3 @Validated和@Valid的区别

区别	@Valid注解	@Validated注解
提供者	JSR-303规范	Spring
是否支持分组	不支持	支持
标注位置	METHOD, FIELD, CONSTRUCTOR, PARAMETER, TYPE_USE	TYPE, METHOD, PARAMETER
嵌套校验	支持	不支持

1.4 依赖导入

- 使用SpringBoot整合 -- 本文所有代码只引入下面两个依赖。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.5.Final</version>
</dependency>
<!--如果spring-boot版本小于2.3.x，引入spring-boot-starter-web即可，hibernate-validator将被自动引入-->
<!--如果spring-boot版本大于2.3.x，引入spring-boot-starter-web、hibernate-validator-->
```

- 本质依赖

```xml
<!--数据校验api接口-->
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>xxx</version>
</dependency>
<!--数据校验api实现-->
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>xxx</version>
</dependency>
<!--hibernate-validator只不过是一堆数据校验接口的实现类，和数据库没有半点关系-->
```

1.5 常用数据校验注解

```
// JSR提供的校验注解：
@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提供的校验注解：
@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
```

2 使用

2.1 校验实体类

2.1.1 表单校验

- cn.king.validation01.pojo.User

```java
package cn.king.validation01.pojo;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;
import javax.validation.constraints.*;
import java.util.Date;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/17 17:16
 * @version: 1.0.0
 * @description:
 */
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class User {

    private Integer id;

    // 每一个注解都包含了message字段，用于校验失败时作为提示信息。不写message将使用默认的错误提示信息。
```

```
    @Size(min = 5, max = 10, message = "请输入5-10个字符的用户名")
    private String username;

    private String password;

    @Min(18)
    private Integer age;

    @NotBlank(message = "手机号码不能为空")
    @Pattern(regexp = "^1(3|4|5|7|8)\\d{9}$", message = "手机号码格式错误")
    private String phone;

    @Email(message = "邮箱格式错误")
    private String email;

    @NotNull(message = "生日不能为空")
    @Past // 生日必须是一个过去的时间
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date birthday;

}
```

- cn.king.validation01.controller.AUserController

```
package cn.king.validation01.controller;

import cn.king.validation01.pojo.User;
import org.springframework.validation.BindingResult;
import org.springframework.validation.Errors;
import org.springframework.validation.FieldError;
import org.springframework.validation.ObjectError;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.validation.Valid;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/17 17:17
```

```
 * @version: 1.0.0
 * @description: 表单校验
 */
@RestController
@RequestMapping("/a/user")
public class AUserController {

    /**
     * @author: wjl@king.cn
     * @createTime: 2021/1/17 17:20
     * @param: user
     * @param: bindingResult
     * @return: java.lang.Object
     * @description: ..
     * 在需要校验的pojo前面加@Validated注解代表校验该参数。
     * 在需要校验的pojo后面加BindingResult参数，用来接收校验出错误时的提示信息。
     *  => @Validated注解和BindingResult参数必须配对使用，并且位置顺序固定。 如果要校验的参数有多个，入参写法：(@Validated Foo foo, BindingResult fooBindingResult, @Validated Bar bar, BindingResult barBindingResult);
     */
    @PostMapping
    public Object addUser(@Validated User user, BindingResult bindingResult) {
        System.out.println(user);
        // 获取校验错误的提示信息
        // 如果有错误提示信息
        if (bindingResult.hasErrors()) {
            // bindingResult.getFieldErrors() 字段的错误
            for (FieldError fieldError : bindingResult.getFieldErrors()) {
                // fieldError.getField() 绑定失败的字段名
                // fieldError.getDefaultMessage() 默认的错误提示信息
                System.out.println(fieldError.getField() + " : " + fieldError.getDefaultMessage());
            }
            // bindingResult.getAllErrors() 所有错误
            for (ObjectError objectError : bindingResult.getAllErrors()) {

System.out.println(objectError.getDefaultMessage());
```

```
            }
            return "fail";
        }
        return "success";
    }

    /**
     * @author: wjl@king.cn
     * @createTime: 2021/1/17 17:30
     * @param: user
     * @param: bindingResult
     * @return: java.lang.Object
     * @description: ..
     * => @Validated：Spring提供的数据校验
     * => @Valid：JSR303数据校验
     */
    @PutMapping("/fun1")
    public Object updateUser1(@Valid User user, BindingResult
bindingResult) {
        System.out.println(user);
        if (bindingResult.getErrorCount() > 0) {
            for (FieldError fieldError :
bindingResult.getFieldErrors()) {
                System.out.println(fieldError.getField() + " :
" + fieldError.getDefaultMessage());
            }
            for (ObjectError objectError :
bindingResult.getAllErrors()) {

System.out.println(objectError.getDefaultMessage());
            }
            return "fail";
        }
        return "success";
    }

    /**
     * @author: wjl@king.cn
     * @createTime: 2021/1/17 17:31
     * @param: user
     * @param: errors
     * @return: java.lang.Object
     * @description: ..
```

```
90      * BindingResult 继承了 Errors，所以将入参的BindingResult换成
    Errors也行
91      */
92     @PutMapping("/fun2")
93     public Object updateUser2(@Valid User user, Errors errors)
    {
94         System.out.println(user);
95         if (errors.getErrorCount() > 0) {
96             for (FieldError fieldError :
    errors.getFieldErrors()) {
97                 System.out.println(fieldError.getField() + " :
    " + fieldError.getDefaultMessage());
98             }
99             for (ObjectError objectError :
    errors.getAllErrors()) {
100
    System.out.println(objectError.getDefaultMessage());
101            }
102            return "fail";
103        }
104
105        if (errors.hasErrors()) {
106            for (FieldError fieldError :
    errors.getFieldErrors()) {
107                System.out.println(fieldError.getField() + " :
    " + fieldError.getDefaultMessage());
108            }
109            for (ObjectError objectError :
    errors.getAllErrors()) {
110
    System.out.println(objectError.getDefaultMessage());
111            }
112            return "fail";
113        }
114
115        return "success";
116    }
117
118    // 避免下面的写法。
119    @PutMapping("/fun3")
120    public Object updateUser3(@Valid User user) {
121        return null;
122    }
```

```
123
124     /*
125      * BindingResult 比  Errors 常用。
126      * bindingResult.hasErrors() 比
    bindingResult.getErrorCount() 常用。
127      */
128
129 }
```

2.1.2 RequestBody校验

- cn.king.validation01.controller.BUserController

```
1  package cn.king.validation01.controller;
2
3  import cn.king.validation01.pojo.User;
4  import cn.king.validation01.vo.RestResult;
5  import org.springframework.http.HttpStatus;
6  import org.springframework.http.ResponseEntity;
7  import org.springframework.validation.BindingResult;
8  import org.springframework.validation.FieldError;
9  import org.springframework.validation.annotation.Validated;
10 import org.springframework.web.bind.annotation.*;
11 import javax.validation.Valid;
12 import java.util.HashMap;
13 import java.util.Map;
14
15 /**
16  * @author: wjl@king.cn
17  * @time: 2021/1/17 17:17
18  * @version: 1.0.0
19  * @description: RequestBody校验
20  */
21 @RestController
22 @RequestMapping("/b/user")
23 public class BUserController {
24
25     @PostMapping(value = "/fun1")
26     public ResponseEntity<RestResult<Object>> addUser(@Validated
    @RequestBody User user, BindingResult bindingResult) {
27         System.out.println(user);
28         if (bindingResult.hasErrors()) {
```

```
29              return new ResponseEntity<>
(getValidateError(bindingResult),
HttpStatus.UNPROCESSABLE_ENTITY);
30          }
31          return ResponseEntity.ok(RestResult.ok());
32      }
33
34      @PutMapping(value = "/fun2")
35      public RestResult<Object> updateUser(@Valid @RequestBody
User user, BindingResult bindingResult) {
36          System.out.println(user);
37          if (bindingResult.hasErrors()) {
38              return getValidateError(bindingResult);
39          }
40          return RestResult.ok();
41      }
42
43      // 下面写法正确
44      // 校验失败会抛出 MethodArgumentNotValidException 异常
45      @PutMapping(value = "/fun3")
46      public RestResult<Object> updateUser2(@Valid @RequestBody
User user) {
47          return null;
48      }
49
50      /**
51       * 该方法可封装成工具类
52       */
53      static public RestResult<Object>
getValidateError(BindingResult bindingResult) {
54
55          if (!bindingResult.hasErrors()) {
56              return null;
57          }
58
59          Map<String, String> fieldErrors = new HashMap<>();
60
61          for (FieldError error : bindingResult.getFieldErrors())
{
62              fieldErrors.put(error.getField(), error.getCode() +
" | " + error.getDefaultMessage());
63          }
64
```

```
        HashMap<String, Object> result = new HashMap<>();
        result.put("fieldErrors", fieldErrors);

        return RestResult.error(HttpStatus.UNPROCESSABLE_ENTITY.value(), "参数错误", result);
    }

}
```

2.2 校验普通参数

2.2.1 RequestParam校验

```
// 下面写法正确
// 校验失败会抛出 ConstraintViolationException 异常。
@GetMapping("/fun3")
public Object fun3(@Length(min = 5, max = 10) @NotNull String username) {
    // 校验通过才会执行业务逻辑
    return "ok";
}

// 下面的写法错误，不能加BindingResult
@GetMapping("/fun4")
public Object fun4(@Length(min = 5, max = 10) @NotNull String username, BindingResult bindingResult) {
    return null;
}
```

2.2.2 PathVariable校验

```java
// 下面写法正确
// 校验失败会抛出 ConstraintViolationException 异常。
@GetMapping("/fun1/{userId}")
public Object fun1(@PathVariable @Min(1000L) Long userId) {
    // 校验通过才会执行业务逻辑
    return "ok";
}

// 下面的写法错误，不能加BindingResult
@GetMapping("/fun2/{userId}")
public Object fun2(@PathVariable @Min(1000L) Long userId,
BindingResult bindingResult) {
    return null;
}
```

2.3 数据校验+全局异常处理

- cn.king.validation02.pojo.User 同上
- cn.king.validation02.controller.AUserController

```java
package cn.king.validation02.controller;

import cn.king.validation02.pojo.User;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/18 22:16
 * @version: 1.0.0
 * @description:
 */
@RestController
@RequestMapping("/a/user")
public class AUserController {

    /**
     * 使用 @Valid 和 @Validated 都可以。
```

```
     * RequestBody参数校验，校验失败会抛出 MethodArgumentNotValidException 异常。
     */
    @PostMapping("/fun1")
    public Object fun1(@RequestBody @Validated User user) {
        // 校验通过，才会执行业务逻辑处理
        return "ok";
    }

}
```

- cn.king.validation02.controller.BUserController

```
package cn.king.validation02.controller;

import org.hibernate.validator.constraints.Length;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/18 22:16
 * @version: 1.0.0
 * @description:
 */
@Validated
@RestController
@RequestMapping("/b/user")
public class BUserController {

    // RequestMapping / PathVariable 参数校验。校验失败会抛出 ConstraintViolationException 异常。
    @GetMapping("/fun2/{userId}")
    public Object fun2(@PathVariable @Min(10000L) Long userId) {
        // 校验通过，才会执行业务逻辑处理
        return "ok";
    }

```

```
    // RequestMapping / PathVariable 参数校验。校验失败会抛出 ConstraintViolationException 异常。
    @GetMapping("fun3")
    public Object fun3(@Length(min = 5, max = 10) @NotNull String username) {
        // 校验通过，才会执行业务逻辑处理
        return "ok";
    }

}
```

- cn.king.validation02.exception.AGlobalExceptionHandler

```
package cn.king.validation02.exception;

import org.springframework.http.HttpStatus;
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;
import javax.validation.ConstraintViolationException;
import java.util.HashMap;
import java.util.Map;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/16 17:38
 * @version: 1.0.0
 * @description: 全局异常处理器
 * AUserController、BUserController 如果校验失败，会抛出 MethodArgumentNotValidException 或者 ConstraintViolationException 异常。
 * 在实际项目开发中，通常会用统一异常处理来返回一个更友好的提示。
 * 比如我们系统要求无论发送什么异常，http的状态码必须返回200，由业务码去区分系统的异常情况。
 * <p>
 * AGlobalExceptionHandler 和 BGlobalExceptionHandler 注释掉一个进行比较
 */
//@RestControllerAdvice
```

```
26 public class AGlobalExceptionHandler {
27
28     @ExceptionHandler({MethodArgumentNotValidException.class})
29     @ResponseStatus(HttpStatus.OK)
30     @ResponseBody
31     public Object
handleMethodArgumentNotValidException(MethodArgumentNotValidExce
ption ex) {
32         BindingResult bindingResult = ex.getBindingResult();
33         StringBuilder sb = new StringBuilder("校验失败:");
34         for (FieldError fieldError :
bindingResult.getFieldErrors()) {
35
sb.append(fieldError.getField()).append(" : ").append(fieldError.
getDefaultMessage()).append(", ");
36         }
37         String msg = sb.toString();
38
39         Map<String, Object> map = new HashMap<>();
40         map.put("code", -2);
41         map.put("msg", msg);
42         return map;
43     }
44
45     @ExceptionHandler({ConstraintViolationException.class})
46     @ResponseStatus(HttpStatus.OK)
47     @ResponseBody
48     public Object
handleConstraintViolationException(ConstraintViolationException
ex) {
49         Map<String, Object> map = new HashMap<>();
50         map.put("code", -2);
51         map.put("msg", ex.getMessage());
52         return map;
53     }
54
55 }
```

- cn.king.validation02.exception.BGlobalExceptionHandler

```
1 package cn.king.validation02.exception;
2
3 import lombok.extern.slf4j.Slf4j;
```

```java
import org.springframework.validation.BindingResult;
import org.springframework.validation.FieldError;
import
org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import javax.servlet.http.HttpServletRequest;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Slf4j
//@RestControllerAdvice
public class BGlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public Object notValidException(HttpServletRequest request,
MethodArgumentNotValidException e) {
        log.info("请求的url为{}出现数据校验异常,异常信息为:",
request.getRequestURI(), e);
        BindingResult bindingResult = e.getBindingResult();
        List<String> errorMsgList = new ArrayList<>();
        for (FieldError fieldError :
bindingResult.getFieldErrors()) {
            errorMsgList.add(fieldError.getDefaultMessage());
        }

        Map<String, Object> map = new HashMap<>();
        map.put("code", 500);
        map.put("msg", errorMsgList);
        return map;
    }

}
```

2.4 分组校验

- cn.king.validation03.pojo.User

```java
package cn.king.validation03.pojo;

```

```java
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.validator.constraints.Length;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/17 17:16
 * @version: 1.0.0
 * @description: 有时候，为了区分业务场景，对于不同场景下的数据验证规则可能不一样（例如新增时可以不用传递 ID，而修改时必须传递ID），可以使用分组校验。
 */
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class User {

    // groups：标识此校验规则属于哪个分组，可以指定多个分组
    @NotNull(groups = Update.class)
    @Min(value = 10000L, groups = Update.class)
    private Long userId;

    @NotNull(groups = {Save.class, Update.class})
    @Length(min = 2, max = 10, groups = {Save.class, Update.class})
    private String userName;

    @NotNull(groups = {Save.class, Update.class})
    @Length(min = 6, max = 20, groups = {Save.class, Update.class})
    private String account;

    @NotNull(groups = {Save.class, Update.class})
    @Length(min = 6, max = 20, groups = {Save.class, Update.class})
    private String password;

    /**
```

```
     * 一个校验分组
     * 保存的时候校验分组
     */
    public interface Save {
        // 校验分组中不需要定义任何方法，该接口仅仅是为了区分不同的校验规则
    }

    /**
     * 一个校验分组
     * 更新的时候校验分组
     */
    public interface Update {
    }

}
```

- cn.king.validation03.controller.UserController

```
package cn.king.validation03.controller;

import cn.king.validation03.pojo.User;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/16 17:50
 * @version: 1.0.0
 * @description:
 */
@RestController
@RequestMapping("/user")
public class UserController {

    @PostMapping
    public Object saveUser(@RequestBody @Validated(User.Save.class) User user) {
        // 校验通过，才会执行业务逻辑处理
        return "ok";
    }

    @PutMapping
```

```
    public Object updateUser(@RequestBody @Validated(User.Update.class) User user) {
        // 校验通过，才会执行业务逻辑处理
        return "ok";
    }

}
```

2.5 嵌套校验

- cn.king.validation04.pojo.User

```
package cn.king.validation04.pojo;

import lombok.Data;
import org.hibernate.validator.constraints.Length;
import javax.validation.Valid;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

/**
 * @author: wjl@king.cn
 * @time: 2021/1/18 23:05
 * @version: 1.0.0
 * @description: 如果POJO中包含了自定义的实体类，就需要用到嵌套校验。
 * POJO中的某个字段也是一个对象，这种情况下，可以使用嵌套校验。
 */
@Data
public class User {

    @Min(value = 1L, groups = Update.class)
    private Long userId;

    @NotNull(groups = {Save.class, Update.class})
    @Length(min = 2, max = 10, groups = {Save.class, Update.class})
    private String userName;

    @NotNull(groups = {Save.class, Update.class})
```

```
27      @Length(min = 6, max = 20, groups = {Save.class,
    Update.class})
28      private String account;
29
30      @NotNull(groups = {Save.class, Update.class})
31      @Length(min = 6, max = 20, groups = {Save.class,
    Update.class})
32      private String password;
33
34      /**
35       * 此时DTO类的对应字段必须标记@Valid注解
36       */
37      @Valid
38      @NotNull(groups = {Save.class, Update.class})
39      private Job job;
40
41      @Data
42      public static class Job {
43
44          @NotNull(groups = {Update.class})
45          @Min(value = 1, groups = Update.class)
46          private Long jobId;
47
48          @NotNull(groups = {Save.class, Update.class})
49          @Length(min = 2, max = 10, groups = {Save.class,
    Update.class})
50          private String jobName;
51
52          @NotNull(groups = {Save.class, Update.class})
53          @Length(min = 2, max = 10, groups = {Save.class,
    Update.class})
54          private String position;
55      }
56
57      /**
58       * 保存的时候校验分组
59       */
60      public interface Save {
61      }
62
63      /**
64       * 更新的时候校验分组
65       */
```

```
66      public interface Update {
67      }
68
69  }
```

- cn.king.validation04.controller.UserController

```
1   package cn.king.validation04.controller;
2
3   import cn.king.validation04.pojo.User;
4   import org.springframework.validation.annotation.Validated;
5   import org.springframework.web.bind.annotation.*;
6
7   /**
8    * @author: wjl@king.cn
9    * @time: 2021/1/21 21:49
10   * @version: 1.0.0
11   * @description:
12   */
13  @RestController
14  @RequestMapping("/user")
15  public class UserController {
16
17      @PostMapping
18      public Object saveUser(@RequestBody
    @Validated(User.Save.class) User user) {
19          // 校验通过，才会执行业务逻辑处理
20          return "ok";
21      }
22
23      @PutMapping
24      public Object updateUser(@RequestBody
    @Validated(User.Update.class) User user) {
25          // 校验通过，才会执行业务逻辑处理
26          return "ok";
27      }
28
29  }
```

2.6 集合校验

- cn.king.validation05.pojo.ValidationList

```java
package cn.king.validation05.pojo;

import lombok.Data;
import lombok.experimental.Delegate;
import javax.validation.Valid;
import java.util.ArrayList;
import java.util.List;

/**
 * 如果请求体直接传递了json数组给后台，并希望对数组中的每一项都进行参数校验。
 * 此时，如果我们直接使用java.util.Collection下的list或者set来接收数据，参数校验并不会生效！我们可以使用自定义list集合来接收参数：
 * <p>
 * 包装 List类型，并声明 @Valid 注解
 */
@Data
public class ValidationList<E> implements List<E> {

    @Delegate    // @Delegate是lombok注解
    @Valid       // 一定要加@Valid注解
    public List<E> list = new ArrayList<>();

    // 一定要记得重写toString方法
    @Override
    public String toString() {
        return list.toString();
    }

    /*
     * @Delegate注解受lombok版本限制，1.18.6以上版本可支持。
     * 如果校验不通过，会抛出 NotReadablePropertyException， 同样可以使用统一异常进行处理。
     */

}
```

- cn.king.validation05.pojo.User

```java
package cn.king.validation05.pojo;

```

```java
import lombok.Data;
import org.hibernate.validator.constraints.Length;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

@Data
public class User {

    @NotNull
    @Min(value = 1L)
    private Long userId;

    @NotNull
    @Length(min = 2, max = 10)
    private String userName;

    @NotNull
    @Length(min = 6, max = 20)
    private String account;

    @NotNull
    @Length(min = 6, max = 20)
    private String password;

}
```

- cn.king.validation05.controller.ValidationListController

```java
package cn.king.validation05.controller;

import cn.king.validation05.pojo.User;
import cn.king.validation05.pojo.ValidationList;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/valid/list")
public class ValidationListController {

    @PostMapping
```

```
16      public Object saveList(@RequestBody @Validated
    ValidationList<User> userList) {
17          // 校验通过，才会执行业务逻辑处理
18          return "ok";
19      }
20
21  }
```

2.7 自定义校验规则

- 见github代码

2.8 手动校验

- 见github代码

2.9 基于方法校验

- 见github代码

2.10 编程式校验和快速失败

- 见github代码

2.11 Service层入参校验

- Service层如何进行数据校验呢？
    - 前台传来的参数已经在Controller中进行校验了，那么Service层方法的入参是不是就不需要进行校验了？看具体业务场景！如果前台传来的参数是加密的，到达Controller之后进行解密再传到Service，此时就需要校验Service的该入参。

- 我对于数据校验的理解是，只要数据可能不符合规范或者可能会出现空指针，那么该数据就必须进行校验，至于是直接抛异常还是返回null还是返回空集合，看具体业务来决定，如果抛出异常，就要进行好异常处理；如果返回null，就要在方法的调用者处进行好非空判断。

3 其他的数据校验框架

3.1 使用drools作为规则引擎