链滴

Spring Boot 整合 Hibernate Validator

作者：zpwd63
原文链接：https://ld246.com/article/1552530264658
来源网站：链滴

Spring Boot 整合Hibernate Validator

Hibernate Validator提供了对请求参数的校验，方便我们在开发中管理校验信息。本文通过简单的例描述在Spring Boot项目中使用Hibernate Validator。（本文仅描述Spring Boot如何使用，不再独展示单独使用的说明，如果需要了解更多信息，可以参考官方文档）

在Spring Boot的web包中已经为我们引入了Validator的依赖，所以不再需要添加额外的依赖包，直就可以使用。

一、校验模式

Validator提供两种校验模式：**普通校验模式、快速校验模式**

- 普通校验模式：普通校验模式会一次性校验所有参数，并返回所有不符合要求的错误信息。
- 快速校验模式：快速校验模式在校验过程中，当遇到第一个不满足条件的参数时就立即返回，不再续后面参数的校验。

二、配置

在Spring Boot中使用Validator需要注意一个问题。默认情况下在控制器中通过使用@Valid注解和验模型只能在POST请求中生效，GET请求没有效果。所以，在配置时，我们需要用到Spring对Validator的拓展。配置信息如下：

```
@Configuration
public class ValidatorConfiguration {

    //配置1
    @Bean
    public Validator validator() {
        ValidatorFactory validatorFactory = Validation.byProvider(HibernateValidator.class)
                .configure()
                .addProperty("hibernate.validator.fail_fast", "true") //快速验证模式，有第一个参数不足条件直接返回
                .buildValidatorFactory();
        return validatorFactory.getValidator();
    }

    //配置2
    @Bean
    public MethodValidationPostProcessor methodValidationPostProcessor() {
        MethodValidationPostProcessor postProcessor = new MethodValidationPostProcessor();
        postProcessor.setValidator(validator());
        return postProcessor;
    }

}
```

配置1是Validator的默认配置，而配置2则是对默认配置的拓展，解决了GET请求参数的校验。同时在校验模式上直接使用快速校验，只要满足一个参数不符合就立即返回错误信息。如果所有信息都返

，则取消配置即可。

三、错误信息处理

在Validator配置中，默认的错误返回方式是通过BindingResult对象进行返回的，而在Spring拓展中通过ConstraintViolation集合返回，对应的异常信息也不一样。

在web项目中，参数校验异常信息都需要经过自定义处理，封装成统一的格式进行输出。定义全局异处理并捕获对应的异常进行处理是比较通用的一种方式。代码实现如下：

```
@ControllerAdvice
public class GlobalExceptionHandler {

    /**
     * 未被关注的异常信息，统一返回给客户端为“系统异常”
     *
     * @param e
     * @return
     */
    @ExceptionHandler(RuntimeException.class)
    @ResponseBody
    public JSONResult handler(RuntimeException e) {
        e.printStackTrace();
        return new JSONResult(-1, "系统异常");
    }

    /**
     * Hibernate Validator参数校验异常处理
     *
     * @param e
     * @return
     */
    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseBody
    public JSONResult handler(MethodArgumentNotValidException e) {
        BindingResult bindingResult = e.getBindingResult();
        ObjectError objectError = bindingResult.getAllErrors().get(0);
        return new JSONResult(-1, objectError.getDefaultMessage());
    }

    /**
     * Spring Validator参数校验异常处理
     *
     * @param e
     * @return
     */
    @ExceptionHandler(ConstraintViolationException.class)
    @ResponseBody
    public JSONResult handler(ConstraintViolationException e) {
        Set<ConstraintViolation<?>> constraintViolations = e.getConstraintViolations();
        for (ConstraintViolation<?> constraintViolation : constraintViolations) {
            String message = constraintViolation.getMessage();
            if (!StringUtils.isEmpty(message)) {
                //直接返回第一个错误信息
```

```
                return new JSONResult(-1, message);
            }
        }
        return new JSONResult(-1, "参数错误");
    }
}
```

由于采用的快速校验模式，在处理异常时直接返回第一个错误信息即可。

四、校验模型和参数定义

- POST请求

在post请求中我们需要定义一个校验模型，并配置@Valid注解进行使用。

```
public class UserLoginDto implements Serializable {

    //账号
    @NotBlank(message = "账号不能为空")
    private String username;

    //验证码
    @NotBlank(message = "验证码不能为空")
    private String code;

    //Getter Setter...
}

  @PostMapping(value = "/login")
    public User login(@RequestBody @Valid UserLoginDto account) {
        //do ...
        return null;
    }
```

- GET请求

在get请求中稍有变化。

```
  @GetMapping(value = "/info")
    public JSONResult userInfo(@NotEmpty(message = "ID不能为空") String userId) {
        //do ...
        return JSONResult.result(null);
    }
```

除了在参数中需要定义特定的注解信息以外，还需要在控制器类名上加上@Validated注解配置使用
此处只是做简单的示例，以下为常用注解：

@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 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

以上就是Validator在Spring Boot中的简单使用，实际功能远不止这些，具体情况建议参考官方文档。