# org.ygy.common.validator-0.0.1.jar

## 简介

该工具用于对方法参数字段的校验，源于“防御性编程”的理念。避免字段校验逻辑和业务代码相互嵌入所造成的混乱，让程序员专注于业务实现，保持代码的纯粹性，是该工具的目标。该工具基于使用广泛的Spring框架，定义封装校验规则的注解@Validate，使用AOP对添加了该注解的方法进行切入，实现根据规则对参数字段进行校验的目的。校验代码只是业务代码的辅助，只是为了代码流程更健壮一点，所以它应该简单一点，基本原则是不符合校验格式的不进行校验，对多个字段进行校验，当有一个校验不通过时，默认不再进行后续字段的校验，可配置。对于使用来说也应该简单（觉得用注解就比较简单）。当然将校验规则写在配置文件里也支持。对于校验结果的处理，提供默认处理方式，可以自定制。

## 规则说明

规则大体结构有三种，如下（用在校验控制器方法比较好）：

**（1）字段名：校验规则**

针对SpringMVC中的控制器方法，使用request.getParameter("age")获取请求参数的形式。

**（2）字段位置：校验规则**

针对所有方法，包括SpringMVC中的控制器方法,使用参数名映射的方式。出于程序员的自我修养，字段位置从0开始。

**（3）字段名：字段位置：校验规则**

和结构（2）一样，适用所有方法。之所以增加一个字段名，只是为了在配置文件上使用时好检查一点。

该工具预设了9种校验规则，为 must,int,strin,len,date,reg,long,float,double。

注：暂时只支持基本数据类型的校验，复杂对象类型暂不支持。

### must

该规则指定字段是必要的，不能为null或空串。如name:must(true)表示字段name是必要的。

### int

该规则指定如果该字段非null或空串，则为int型数据，且在规定大小内。如age:int(1,10)表示字段age为int型数据，且要求1<age<10。当然可以使用方括号“[ ”“]”。如age:int[1,10)要求1<=age<10，圆括号、方括号的使用与数学上的意义一样。如只要求大于某个值，可以age:int(1,M),只要求小于某个值，可以age:int(m,10)。其中小写m表示int型数据可表示的最小值，大写M表示int型数据可表示的最大值。

### strin

该规则指定如果该字段非null或空串，要求为指定字串数组中的一个。如status:strin(01,02,03)表示字段status必须为01、02、03其中一个。

### len

该规则指定如果该字段非null或空串，要求为指定长度。如phone:len(11)表示字段phone长度必须为11。

### date

该规则指定如果该字段非null或空串，要求为指定日期格式。如startTime:date(yyyy-MM-dd)表示字段startTime格式为yyyy-MM-dd。

### reg

该规则指定如果该字段非null或空串，要求匹配指定正则表达式。

### long

该规则指定如果该字段非null或空串，则为long型数据，使用方式与int型规则相似。

### float

该规则指定如果该字段非null或空串，则为float型数据，使用方式与int型规则相似。

### double

该规则指定如果该字段非null或空串，则为double型数据，使用方式与int型规则相似。

## 使用说明

### 环境配置文件

1、pom.xml添加Spring AOP相关jar包

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>4.3.1.RELEASE</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjrt</artifactId>

<version>1.8.9</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.8.9</version>

</dependency>

<dependency>

<groupId>asm</groupId>

<artifactId>asm</artifactId>

<version>3.3.1</version>

</dependency>

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

<version>3.2.4</version>

</dependency>

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib-nodep</artifactId>

<version>3.2.4</version>

</dependency>

<dependency>

<groupId>aopalliance</groupId>

<artifactId>aopalliance</artifactId>

<version>1.0</version>

</dependency>

2、Spring配置文件启动aop注解和组件扫描

<beans

*//省略*

xmlns:aop=*"http://www.springframework.org/schema/aop"*

xsi:schemaLocation=*"*

*//省略*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop.xsd"*>

<aop:aspectj-autoproxy/> <!-- aspect注解生效-->

<context:component-scan base-package="org.ygy.common.validator" />

</beans>

3、web.xml添加检验配置文件路径

<filter>

<filter-name>sessionFilter</filter-name> <filter-class>

org.ygy.common.validator.GetContextFilter

</filter-class>

<init-param>

<param-name>validateConfig</param-name>

<param-value>conf/validate.properties</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>sessionFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

如果没有使用到配置文件，可以不配以下信息

<init-param>

<param-name>validateConfig</param-name>

<param-value>conf/validate.properties</param-value>

</init-param>

### @Validate说明

（1）

@Validate

@ResponseBody

@RequestMapping("getPersonInfo")

**public** Map<String,Object> getPersonInfo(HttpServletRequest request, HttpServletResponse response)

校验规则在配置文件中，如validate.properties中配置：

getPersonInfo = age:int(1,10)|name:must(true)

（2）

@Validate("age:int(1,10)")

@ResponseBody

@RequestMapping("getPersonInfo")

**public** Map<String,Object> getPersonInfo(HttpServletRequest request, HttpServletResponse response)

检验规则在注解里，如果validate.properties配置文件中有相关校验规则，不予理会，注解里的校验规则优先级高于配置文件。这种情况下，使用的是规定的9种规则，具体校验内容在@Validate("age:int(1,10)")中，如果应用中全是类似这样的注解配置校验规则，sessionFilter过滤其中的validate.properties配置文件可以没有。

（3）

@Validate(url="ygy")

@ResponseBody

@RequestMapping("getPersonInfo")

**public** Map<String,Object> getPersonInfo(HttpServletRequest request, HttpServletResponse response)

校验规则在配置文件中，对应配置文件中的键为“ygy”,如validate.properties中配置：

ygy = age:int(1,10)|name:must(true)

（4）

@Validate(value="age:int(1,10)|name:must(true)",all=**true**)

@ResponseBody

@RequestMapping("getPersonInfo")

**public** Map<String,Object> getPersonInfo(HttpServletRequest request, HttpServletResponse response)

检验规则在注解里,对多个字段校验，当某个校验不通过时，all=**true表示**后续校验依旧进行（默认all=false）

（5）

@Validate(value="age:int(1,10)",handler="ygyHandler")

@ResponseBody

@RequestMapping("getPersonInfo")

**public** Map<String,Object> getPersonInfo(HttpServletRequest request, HttpServletResponse response)

@ResponseBody

@RequestMapping("ygyHandler")

**public** Object validateHandler(HttpServletRequest request, HttpServletResponse response){

List<ValidateResult> validateResults2 = (List<ValidateResult>) request.getAttribute("validateResults");

//省略

}

其中request.getAttribute("validateResults");中存放的是校验错误结果信息。

默认情况下，不需要编写自己的校验错误处理器，有默认。如果要编写自己的默认处理器，如上即可。处理器映射若为@RequestMapping("ygyHandler")，则@Validate中得添加handler="ygyHandler"。该例的校验错误处理器@Validate(value="age:int(1,10)",handler="ygyHandler")只针对当前控制器方法的校验错误处理。如果要设置所有方法校验错误统一的处理器，可以在validate.properties中配置：

validateErrorHandlerUrl = ygyValidateErrorHandlerUrl

其中ygyValidateErrorHandlerUrl便是校验错误处理器的映射url。这样@Validate中可以不设置handler，如果设置了，优先使用@Validate中设置的handler，validate.properties上述配置不予理会。推荐使用配置文件配置，这样的好处在于，统一了校验错误处理器的返回值规范，如果这个返回值规范与正常执行的控制器方法返回值规范一致，这就统一了所有返回值规范，有利于前端人员处理。当然这个检验错误处理可以是多种形式，比如跳转指定页面等，不局限于返回json。

相较于0.0.1版本，增加了自定义校验规则的设计。

### 自定义校验规则

步骤一、实现接口

**public** **interface** IValidateRuleHandler {

**public** **boolean** validate(RuleInfo ruleInfo, Object filedValue);

}

如以自定义绝对值校验器为例，如下：

**public** **class** Abs **implements** IValidateRuleHandler{

@Override

**public** **boolean** validate(RuleInfo ruleInfo, Object filedValue) {

**try** {

**if** (ruleInfo.isFormatCorrect()){

**int** value = 0;

**if** (ruleInfo.isRequestParamRule() ) {

value = Integer.*parseInt*((String) filedValue);

} **else** {

value = (Integer)filedValue;

}

**int** validateValue = Integer.*parseInt*(ruleInfo.getRuleContent()[0]);

**if** ( !(Math.*abs*(value) == validateValue) ) {

**return** **false**;

}

}

} **catch** (Exception e) {

}

**return** **true**;

}

}

步骤二、validate.propeties配置如下

addRule = abs

abs = org.ygy.test.Abs

addRule = abs的意思是增加添加的自定义规则为abs,如果添加多个，规则间以逗号分隔如addRule = abs,equal。

abs = org.ygy.test.Abs配置自定义规则的处理器的类完整路径

步骤三、

@Validate("age:abs(1)")

@ResponseBody

@RequestMapping("getPersonInfo")

**public** Map<String,Object> getPersonInfo(HttpServletRequest request, HttpServletResponse response)

# org.ygy.common.validator-0.0.2.jar

除了代码bug的修复外，相较于0.0.1版本有如下几点不同。

1. 支持简单的复杂对象(即属性都为基本类型)校验

当入参为单个复杂对象时，可添加isComplexObject=**true**标识为复杂对象，如：

@Validate(value="age:int(1,M)|status:strin(0,1,2)",isComplexObject=**true**)

@ResponseBody

@RequestMapping("getInfo")

**public** Map<String,Object> getInfo(Person p){

校验Person对象中的age和status属性。

1. 增加字段全局的必要性设置

我们知道0.0.1版本可以通过must校验规则实现字段必要性的校验，但如果想对所有字段都要求必要性校验，那就比较麻烦。故新增字段全局必要性设置。

@Validate(value="age:int(1,M)!|status:strin(0,1,2)",must=**true**)

@ResponseBody

@RequestMapping("getInfo")

**public** Map<String,Object> getInfo(HttpServletRequest request, HttpServletResponse response){

must=true表示所有字段不能为空，规定age为大于1的int整数，单个感叹号“!”表示age可以为空（优先级高于must）。当然must没有配置时默认为false，及所有字段不要求必要性校验。如果某个字段要求必要性校验，可以在校验表达式后添加两个感叹号“!!”即可。如@Validate(value="age:int(1,M)!!|status:strin(0,1,2)|name:len(4)")表示除了age要求必要性校验，其他字段不作要求。