# 一 验证框架

采用Hibernate-validator来进行验证，Hibernate-validator实现了JSR-303验证框架支持注解风格的验证

## 1 导入jar包

hibernate-validator-5.0.1.final.jar

validation-api-1.1.0.final.jar

jboss-logging-3.1.3.GA.jar

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.0.1.Final</version>

</dependency>

## 2 配置

<mvc:annotation-driven validator=*"validator"* />

<!-- 配置资源文件 -->

<bean id=*"hibernateMessages"* class=*"org.springframework.context.support.ResourceBundleMessageSource"*>

<property name=*"defaultEncoding"* value=*"UTF-8"*></property>

<property name=*"basenames"*>

<list>

<value>ValidationMessages</value>

</list>

</property>

</bean>

<!-- 配置验证器 -->

<bean id=*"validator"* class=*"org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"*>

<property name=*"providerClass"* value=*"org.hibernate.validator.HibernateValidator"*></property>

<property name=*"validationMessageSource"* ref=*"hibernateMessages"*></property>

</bean>

## 3 实体类中使用的注解

1）空检查

@Null 验证对象是否为null

@NotNull 验证对象是否不为null, 无法查检长度为0的字符串

@NotBlank 检查约束字符串是不是Null还有被Trim的长度是否大于0,只对字符串,且会去掉前后空格.

@NotEmpty 检查约束元素是否为NULL或者是EMPTY.

2）Booelan检查

@AssertTrue 验证 Boolean 对象是否为 true

@AssertFalse 验证 Boolean 对象是否为 false

3）长度检查

@Size(min=, max=) 验证对象（Array,Collection,Map,String）长度是否在给定的范围之内

@Length(min=, max=) Validates that the annotated string is between min and max included.

4）日期检查

@Past 验证 Date 和 Calendar 对象是否在当前时间之前

@Future 验证 Date 和 Calendar 对象是否在当前时间之后

5）数值检查，建议使用在Stirng,Integer类型，不建议使用在int类型上，因为表单值为“”时无法转换为int，但可以转换为Stirng为"",Integer为null

@Min 验证 Number 和 String 对象是否大等于指定的值

@Max 验证 Number 和 String 对象是否小等于指定的值

@DecimalMax 被标注的值必须不大于约束中指定的最大值. 这个约束的参数是一个通过BigDecimal定义的最大值的字符串表示.小数存在精度

@DecimalMin 被标注的值必须不小于约束中指定的最小值. 这个约束的参数是一个通过BigDecimal定义的最小值的字符串表示.小数存在精度

@Digits 验证 Number 和 String 的构成是否合法

@Digits(integer=,fraction=) 验证字符串是否是符合指定格式的数字，interger指定整数精度，fraction指定小数精度。

@Range(min=, max=) Checks whether the annotated value lies between (inclusive) the specified minimum and maximum.

@Range(min=10000,max=50000,message="range.bean.wage")

private BigDecimal wage;

6）其他

@Pattern 验证 String 对象是否符合正则表达式的规则

@CreditCardNumber信用卡验证

@Email 验证是否是邮件地址，如果为null,不进行验证，算通过验证。

@URL(protocol=,host=, port=,regexp=, flags=)

@Valid 递归的对关联对象进行校验, 如果关联对象是个集合或者数组,那么对其中的元素进行递归校验,如果是一个map,则对其中的值部分进行校验.(是否进行递归验证)

## 4 控制层

@RequestMapping("/add.do")

//BindingResult 必须紧跟在@Valid修饰的变量后面

**public** String add(@Valid Employee emp, BindingResult bindResult, Model model){

**if**(bindResult.hasErrors()){

**for**(ObjectError objectError : bindResult.getAllErrors()){

FieldError fieldError = (FieldError) objectError;

model.addAttribute(fieldError.getField(), fieldError.getDefaultMessage());

}

// return "redirect:error.jsp";

**return** "error";

}

**try** {

empService.addEmp(emp);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

model.addAttribute("errorinfo", e.getMessage());

**return** "redirect:error.jsp";

}

**return** "redirect:list.do";

}

## 5 使用注解修饰方法上的参数

需要使用@Validated修饰控制器类，才能在方法中使用验证的注释。使用@Validate修饰类后，需要验证参数的方法不用写BindResult参数，因为如果验证不通过，会直接跳转到异常方法，不会进到该方法里

1)在springmvc的配置中使用

<bean class=*"org.springframework.validation.beanvalidation.MethodValidationPostProcessor"*/>

2)类上使用@Validated注解

@Controller

@Validated

**public** **class** UserController {

**public** Map<String, Object> updateUser(@NotEmpty(message="name can not null") String name){

User u = **new** User();

u.setName(name);

userService.updateUser(u);

Map<String, Object> map = **new** HashMap<>();

map.put("code", 1);

**return** map;

}

}

3)控制器中设置异常的语法：

@ExceptionHandler(ConstraintViolationException .**class**)

**public** Map<String, Object> exception(ConstraintViolationException e){

Iterator<ConstraintViolation<?>>

iterator = e.getConstraintViolations().iterator();

String message = **null**;

**if** (iterator.hasNext()) {

message = iterator.next().getMessage();

System.***out***.println(message);

}

System.***out***.println("456" + e.getMessage());

Map<String, Object> map = **new** HashMap<>();

map.put("code", 0);

map.put("info", e.getMessage());

**return** map;

}

# 二 json数据处理

默认使用jackson

## 1导入的jar包

Spring相关jar包；

jackson-annotations-XXX.jar

jackson-core-XXX.jar

jackson-databind-XXX.jar

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.9.5</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.5</version>

</dependency>

//通用方式

@RequestMapping("/json1")

**public** **void** test(HttpServletRequest request, HttpServletResponse response){

//设置返回的数据的格式

response.setContentType("text/json");

response.setCharacterEncoding("utf8");

String info = "{\"msg\":\"success\"}";

**try** {

response.getWriter().write(info);

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

## 2 使用注解

@ResponseBody

返回json格式

@RequestMapping("/map2json")

@ResponseBody //可以将对象转为json格式数据

**public** Map<String, Object> map2json(){

Map<String, Object> map = **new** HashMap<>();

map.put("msg", "success");

map.put("code", 101);

**return** map;

}

//将实体类对象转为json格式数据

@RequestMapping("/bean2json")

**public** @ResponseBody User ben2json(){

User u = **new** User();

u.setId(123);

u.setName("zhangsan");

**return** u;

}

//将List中数据转为json格式数据

@RequestMapping("/list2json")

@ResponseBody

**public** List<User> list2json(){

List<User> l = **new** ArrayList<>();

l.add(**new** User(1, "zhangsan"));

l.add(**new** User(2, "lisi"));

**return** l;

}

## 3 日期格式处理

1）在实体类的get方法上使用注解

@JsonFormat(pattern="yyyy-MM-dd HH:mm:ss",timezone = "GMT+8")

**public** Date getBirth() {

**return** birth;

}

2）使用配置

Springmvc 默认使用jackson

<mvc:annotation-driven>

<!-- 处理json里的日期数据，默认日期被fastjson转为时间戳数据 -->

<mvc:message-converters>

<bean class=*"org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"*>

<property name=*"objectMapper"*>

<bean class=*"com.fasterxml.jackson.databind.ObjectMapper"*>

<property name=*"dateFormat"*>

<bean class=*"java.text.SimpleDateFormat"*>

<constructor-arg type=*"java.lang.String"* value=*"yyyy-MM-dd HH:mm:ss"* />

</bean>

</property>

</bean>

</property>

</bean>

</mvc:message-converters>

</mvc:annotation-driven>

# 三 Spring与SpringMvc整合

## 1引入相关jar文件

## 2 web.xml的配置

<!-- post提交方式中文乱码的处理 -->

<filter>

<filter-name>characterEncoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncoding</filter-name>

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

</filter-mapping>

<!-- spring mvc 的核心配置 -->

<servlet>

<servlet-name>DispatchServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 指定加载哪个配置文件 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-mvc.xml</param-value>

</init-param>

<!-- tomcat服务器启动时，创建servlet对象 -->

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>DispatchServlet</servlet-name>

<!-- 过滤任何资源，会造成静态资源不能访问 -->

<url-pattern>/</url-pattern>

</servlet-mapping>

<!-- spring的核心配置 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<!-- classpath相当于/WEB-INF/classes -->

<param-value>classpath:spring-bean.xml</param-value>

<!-- <param-value>/WEB-INF/classes/bean.xml</param-value> -->

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

## 3 spring-mvc.xml

Springmvc的配置中，如果采用如下方案扫描控制层注解，需要设置use-default-filters=*"false"*，不使用默认的过滤器扫描注解。

<!-- 扫描注解 -->

<context:component-scan base-package=*"com.rr.user"* use-default-filters=*"false"*>

<!-- 指定扫描哪些注解 -->

<context:include-filter type=*"annotation"* expression=*"org.springframework.stereotype.Controller"*/>

</context:component-scan>

<mvc:annotation-driven />

<mvc:resources location=*"/js/"* mapping=*"/js/\*\*"*></mvc:resources>

<!-- 视图解析器 -->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 前缀 -->

<property name=*"prefix"* value=*"/"*></property>

<!-- 后缀 -->

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

## 4 spring-bean.xml

<!-- 扫描注解 -->

<context:component-scan base-package=*"com.rr.user"*>

<!-- 扫描时，排除指定的注解 -->

<context:exclude-filter type=*"annotation"* expression=*"org.springframework.stereotype.Controller"*/>

</context:component-scan>

# 四 Spring-test

导入spring-test的包

导入junit的jar包

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

通过spring，结合junit进行单元测试

@RunWith(SpringJUnit4ClassRunner.class)//使用junit

@ContextConfiguration("/spring-bean.xml") //指定加载的spring配置文件

@Transactional

public class BaseTest {

}

具体调用方式：

@Test

@Rollback(false)

public void testAddEmp() {

Employee emp = new Employee();

emp.setAge(23);

emp.setName("GGGGGG");

try{

empService.addEmp(emp);

}catch(Exception e){

System.out.println(e.getMessage());

}

}