<http://blog.csdn.net/asd_op/article/details/53232039>

# [SpringMVC4+thymeleaf3的一个简单实例（篇四：form表单数据验证）](http://blog.csdn.net/asd_op/article/details/53232039)

标签： [java](http://www.csdn.net/tag/java)[spring mvc](http://www.csdn.net/tag/spring%20mvc)[thymeleaf](http://www.csdn.net/tag/thymeleaf)[web](http://www.csdn.net/tag/web)

2016-11-19 16:58 1413人阅读 [评论](http://blog.csdn.net/asd_op/article/details/53232039#comments)(4) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/asd_op/article/details/53232039#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

SpringMVC（4） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg thymeleaf（4） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg java（5） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许你可以随意转走。

关于表单数据验证有很多中方法，这里我仅介绍JSR303注解验证。

注意在[**spring**](http://lib.csdn.net/base/javaee)的配置文件spring-mvc.xml中要有这句代码：<mvc:annotation-driven/>，有了它，spring框架会自动加载classpath的jsr303的实现。

JSR303仅仅是一个规范，这里我们要用到它的一个实现：[**hibernate**](http://lib.csdn.net/base/javaee)-validator。

开工之前，我们需要引入以下lib文件到WEB-INF/lib，并添加到classpath：

validation-api-1.1.0.Final.jar

classmate-1.3.1.jar

jboss-logging-3.3.0.Final.jar

hibernate-validator-5.3.2.Final.jar

以上jar文件都在hibernate-validator-5.3.2.Final-dist.zip这里，官网下载<http://hibernate.org/validator/>

延续前面篇节的内容。

一、修改AnimalForm.[**Java**](http://lib.csdn.net/base/javase)类，在oname，ocount，memo字段上分别加上验证注解，代码如下：

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. **package** com.zoo.web.form;
3. **import** javax.validation.constraints.NotNull;
4. **import** javax.validation.constraints.Size;
6. **import** org.hibernate.validator.constraints.NotEmpty;
7. **import** org.hibernate.validator.constraints.Range;
9. **public** **class** AnimalForm {
11. **private** **long** id;
13. @NotEmpty(message="动物名: 不能为空")
14. **private** String oname;
16. @Range(min = 1, message="数量: 必须大于0")
17. @NotNull(message="数量: 不能为空")
18. **private** **int** ocount;
20. @Size(max = 10, message="备注: 长度不能超过10个字符")
21. **private** String memo;
23. /\*\* 省略getter和setter \*\*/

26. }

解释：

**@NotEmpty：**这个注解表示检查oname字段是不是为空字符串""或者是不是为null，如果是则给出提示信息："动物名：不能为空"。

它支持的类型包括：字符序列CharSequence(CharBuffer, Segment, String, StringBuffer, StringBuilder)；集合Collection(ArrayList, HashSet, Stack, Vector等，很多)；Map以及数组arrays。它将检查所给对象的是不是为empty或者null，empty也就是长度为0，对于字符串来说就是""。

**@NotNull：**检查所标注元素ocount不能为null，如果是则给出提示信息：“数量：不能为空”。

它支持任意类型，检查标注对象是否为null。**注意和@NotEmpty的区别**，她不检查对象是不是为empty。empty对于字符串来说是空字符串，对于集合以及map或数组来说就是所含元素数量为0。

**@Range(min=, max=)：**表示ocount元素的最小值是1，如果小于1,则给出信息：“数量：必须大于0”。

支持类型：BigDecimal, BigInteger, CharSequence, byte, short, int, long 以及这些原始类型对应的wrapper（包装类）。它将检查所给对象的值是不是大于等于min且小于等于max。

**@Size(min=, max=)：**检查memo对象的长度不能超过10, 否则提示：“备注：长度不能超过10个字符”。

适用于CharSequence, Collection, Map 以及数组，检查标注对象的size是大于等于min并且小于等于max。

**注意这么做验证是有问题的**，比如oname输入几个空格它会验证通过，而对于ocount，在输入整数的情况下这完全没有问题，但是如果我们输入带小数点的数字或者输入非数字，或者空字符串的时候程序就会出现exception，这不是我们所希望的，具体的改进代码我们在篇末说明。

关于hibernate validator详细介绍请参阅 reference <http://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/>

二、修改ZooController里的doAdd方法：

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. @RequestMapping(path = "/list", params = {"save"}, method = RequestMethod.POST)
2. **public** String doAdd(Model model, @Valid AnimalForm form, BindingResult result){
3. System.out.println("动物名：" + form.getOname());
4. System.out.println("数量：" + form.getOcount());
5. System.out.println("备注：" + form.getMemo());
6. **if**(result.hasErrors()){
7. model.addAttribute("MSG", "出错啦！");
8. }**else**{
9. model.addAttribute("MSG", "提交成功！");
10. }
11. **return** "zoolist";
12. }

解释：

**方法中Model参数**，用于存放任意数据以便传递到页面，注意Model仅仅是一个接口，spring框架会帮我们实例化具体的类并设置到该方法当中；上例我们在该model里放了一个key为“MSG”的attribute，页面上通过表达式就可以取得其值。

**@Valid AnimalForm form**，@Valid表示要对该form进行验证，具体验证规则就是根据上面【一】里提到；spring框架会根据字段名称将页面传递过来的值绑定到animalForm中。

**BindingResult result**，spring框架会将验证结果设置到该参数，并将该参数放到model传递给页面。

springMVC是非常灵活的，以下几种写法可以达到同样的效果：

(1）

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. @RequestMapping(path = "/list", params = {"save"}, method = RequestMethod.POST)
2. **public** ModelAndView doAdd(@Valid AnimalForm form, BindingResult result){
3. ModelAndView model = **new** ModelAndView();
4. System.out.println("动物名：" + form.getOname());
5. System.out.println("数量：" + form.getOcount());
6. System.out.println("备注：" + form.getMemo());
7. **if**(result.hasErrors()){
8. model.addObject("MSG", "出错啦！");
9. }**else**{
10. model.addObject("MSG", "提交成功！");
11. }
12. model.setViewName("zoolist");
13. **return** model;
14. }

(2)

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. @RequestMapping(path = "/list", params = {"save"}, method = RequestMethod.POST)
2. **public** ModelAndView doAdd(ModelAndView model, @Valid AnimalForm form, BindingResult result){
3. System.out.println("动物名：" + form.getOname());
4. System.out.println("数量：" + form.getOcount());
5. System.out.println("备注：" + form.getMemo());
6. **if**(result.hasErrors()){
7. model.addObject("MSG", "出错啦！");
8. }**else**{
9. model.addObject("MSG", "提交成功！");
10. }
11. model.setViewName("zoolist");
12. **return** model;
13. }

(3)

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. @RequestMapping(path = "/list", params = {"save"}, method = RequestMethod.POST)
2. **public** String doAdd(@Valid AnimalForm form, BindingResult result){
3. System.out.println("动物名：" + form.getOname());
4. System.out.println("数量：" + form.getOcount());
5. System.out.println("备注：" + form.getMemo());
7. **return** "zoolist";
8. }

注意(1)(2)仅仅是ModelAndView实例化的方式不同而已，一个是自己手动实例化，一个是框架实例化；

(3)中我们去掉了model参数，但这并不影响我们的验证以及将验证结果传递到页面，只不过是你不能通过model设置一些attribute到页面了。

这里说一下**Model和ModelAndView的区别**：

**Model**主要用于将数据传递到页面，一般采用model.addAttribute("key", object)的方式，页面通过各种表达式将其显示出来；

**ModelAndView**有两个作用，一个是上面Model的作用；另一个就是可以设置view，也就是跳转方向，view既可以是字符串，也可以是View类型的object。

三、添加代码到zoolist.html

**[html]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. **<div** th:text="${MSG}"**>**这里是信息提示.**</div>**
2. **<br>**
3. **<div** th:errors="${animalForm.oname}"**></div>**
4. **<div** th:errors="${animalForm.ocount}"**></div>**
5. **<div** th:errors="${animalForm.memo}"**></div>**

完整的代码：

**[html]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. <!DOCTYPE html**>**
2. **<html** xmlns:th="http://www.thymeleaf.org"**>**
3. **<head>**
4. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
5. **<title>**zoo list**</title>**
6. **</head>**
7. **<body>**
8. **<a** href='.'**>**首页**</a>**-**>**动物列表
9. **<br><br>**
10. **<div** th:text="${MSG}"**>**这里是信息提示.**</div>**
11. **<br>**
12. **<div** th:errors="${animalForm.oname}"**></div>**
13. **<div** th:errors="${animalForm.ocount}"**></div>**
14. **<div** th:errors="${animalForm.memo}"**></div>**
15. **<br>**
16. **<form** id="iform" th:action="@{/list.html?save}" th:method="post" th:object="${animalForm}"**>**
17. **<table** border="1"**>**
18. **<tr>**
19. **<th>**动物名称**</th>**
20. **<th>**数量**</th>**
21. **<th>**备注**</th>**
22. **<th>**Action**</th>**
23. **</tr>**
24. **<tr>**
25. **<td><input** type="text" name="oname" value="" th:value="\*{oname}"**/></td>**
26. **<td><input** type="text" name="ocount" value="" th:value="\*{ocount}"**/></td>**
27. **<td><input** type="text" name="memo" value="" th:value="\*{memo}"**/></td>**
28. **<td><input** type="submit" value="添加"**/></td>**
29. **</tr>**
30. **</table>**
31. **</form>**
32. **<hr>**
33. **<table** border="1"**>**
34. **<tr>**
35. **<th>**序号**</th>**
36. **<th>**动物名称**</th>**
37. **<th>**数量**</th>**
38. **<th>**备注**</th>**
39. **</tr>**
40. **<tr>**
41. **<td>**1**</td>**
42. **<td>**大马猴**</td>**
43. **<td>**10**</td>**
44. **<td>**机灵古怪，俏皮活泼**</td>**
45. **</tr>**
46. **<tr>**
47. **<td>**2**</td>**
48. **<td>**大熊猫**</td>**
49. **<td>**80**</td>**
50. **<td>**体型笨重，喜欢吃竹子**</td>**
51. **</tr>**
52. **<tr>**
53. **<td>**3**</td>**
54. **<td>**澳洲羊驼**</td>**
55. **<td>**13**</td>**
56. **<td>**长相奇特，大国人俗称其草泥马**</td>**
57. **</tr>**
58. **<tr>**
59. **<td>**4**</td>**
60. **<td>**峨眉山猴**</td>**
61. **<td>**90**</td>**
62. **<td>**不怕人，有时候发贱抢游客面包吃**</td>**
63. **</tr>**
64. **</table>**
65. **</body>**
66. **</html>**

解释：

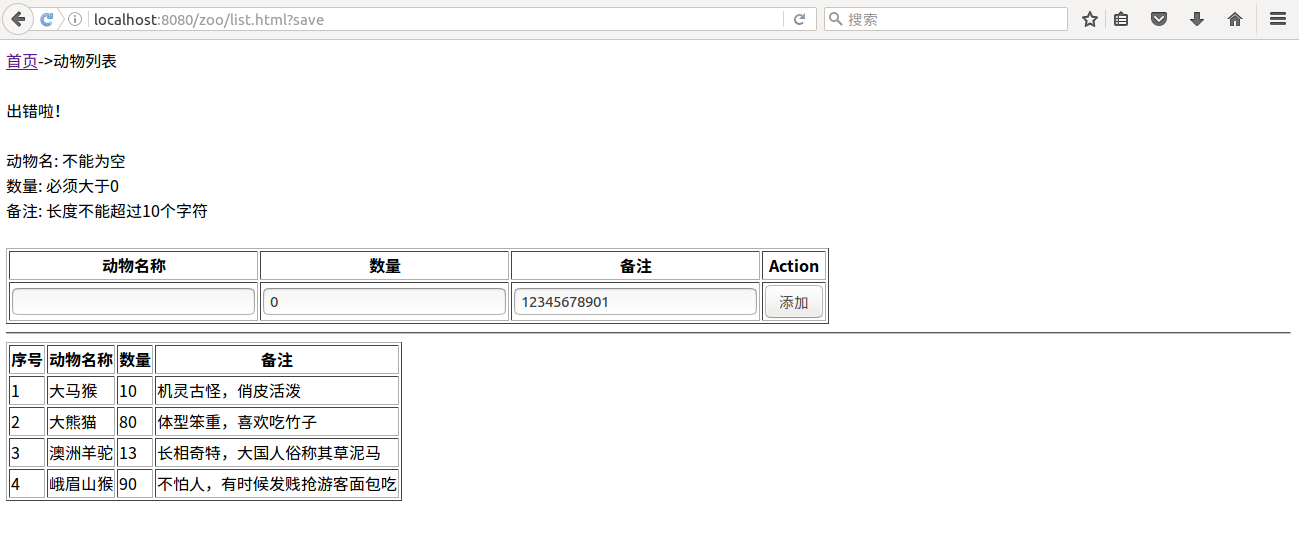
**<div th:text="${MSG}">**这里是信息提示.</div>，还记得前面controller的代码中我们在返回页面的model中放了一个attribute名字叫“MSG”么，对了，在这里我们就可以通过表达式th:text="${MSG}"取得其值了。thymeleaf解析这个标签的时候会将“这里是信息提示.”这个字符串替换成“MSG”对应的内容。

**<div th:errors="${animalForm.oname}"></div>**，thymeleaf使用th:errors表达式可以取得错误信息的内容，${animalForm.oname}表示取得animalForm里oname字段的错误信息；如果验证oname字段时出现错误，那么在这个div里面会显示出该错误信息。

关于thymeleaf的validate表达式详细介绍请参阅 <http://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html#validation-and-error-messages>

好了，我们该修改的都完成了，重启tomcat进入浏览器吧，我的效果如下：

出错的情况：



正确的情况：



好啦，如果你也做到这个样子，也就算达到本篇的目的了，这个页面虽然很简陋，用的也都是静态数据，但这基本上展示了如何使用@valid做form验证。

关于验证改进：

**对于oname**：我们可以换成@NotBlank注解，它可以将全是空格的字符串作为空字符串，和@NotEmpty不同的是，@NotBlank只可用于CharSequence类型，并检查该元素是否为null或者该元素经过trim之后的长度是否为0。

**对于ocount**：我们把验证代码修改为：

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. @NotBlank(message="数量: 不能为空")
2. @Pattern(regexp="[1-9]{1,3}", message="数量X: 必须为正整数，并且0<X<1000")
3. **private** String ocount;

@Pattern，一看便知这是用正则表达式做检查，[1-9]{1,3}表示三位正整数并且要大于0。

如果你对正则表达式还不了解，可以参阅javaSE的api文档：<http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>

对于一些特殊的验证，你可以定义自己的验证类，既可以实现springMVC的validator接口，也可以实现JSR303的注解约束方式验证，这里采用JSR303方式。

1, 创建package：com.zoo.constraint，以及com.zoo.constraint.impl。

2, 在com.zoo.constraint中定义Annotation类型Memo：

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. **package** com.zoo.constraint;
3. **import** **static** java.lang.annotation.ElementType.FIELD;
4. **import** **static** java.lang.annotation.ElementType.METHOD;
5. **import** **static** java.lang.annotation.RetentionPolicy.RUNTIME;
7. **import** java.lang.annotation.Retention;
8. **import** java.lang.annotation.Target;
10. **import** javax.validation.Constraint;
11. **import** javax.validation.Payload;
13. **import** com.zoo.constraint.impl.MemoValidator;
15. @Retention(RUNTIME)
16. @Target({ FIELD, METHOD })
17. @Constraint(validatedBy=MemoValidator.**class**)
18. **public** **@interface** Memo {
20. String message() **default** "请输入正确的备注";
22. Class<?>[] groups() **default** {};
24. Class<? **extends** Payload>[] payload() **default** {};
25. }

3, 在com.zoo.constraint.impl中定义类MemoValidator：

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. **package** com.zoo.constraint.impl;
3. **import** java.util.HashSet;
5. **import** javax.validation.ConstraintValidator;
6. **import** javax.validation.ConstraintValidatorContext;
8. **import** com.zoo.constraint.Memo;
10. **public** **class** MemoValidator **implements** ConstraintValidator<Memo, String> {
12. @Override
13. **public** **void** initialize(Memo arg0) {
14. }
16. @Override
17. **public** **boolean** isValid(String arg0, ConstraintValidatorContext arg1) {
18. HashSet<String> memoSet = **new** HashSet<String>();
19. memoSet.add("圈养");
20. memoSet.add("散养");
21. **return** memoSet.contains(arg0);
22. }
24. }

4, 修改AnimalForm类：

**[java]** [view plain](http://blog.csdn.net/asd_op/article/details/53232039) [copy](http://blog.csdn.net/asd_op/article/details/53232039)

1. **package** com.zoo.web.form;
3. **import** javax.validation.constraints.Pattern;
5. **import** org.hibernate.validator.constraints.NotBlank;
7. **import** com.zoo.constraint.Memo;
9. **public** **class** AnimalForm {
11. **private** **long** id;
13. @NotBlank(message="动物名: 不能为空")
14. **private** String oname;
16. @Pattern(regexp="[1-9]{1,3}", message="数量X不能为空，必须为正整数，并且0<X<1000")
17. **private** String ocount;
19. @Memo(message = "备注不能为空，且只能填写\"圈养\"，或者\"散养\"")
20. **private** String memo;
22. /\*\* getters and setters \*\*/
23. }

ok！重启tomcat，进入浏览器，在三个字段都不输入的情况下，我的界面如下：



是不是很简单呢，不过希望大家在遇到问题时学会自己看reference和api，很多时候你遇到的问题都可以在那里找到。

下一篇是数据持久化之保存在[**MySQL数据库**](http://lib.csdn.net/base/mysql)。