# 此项目在 源码vialidate 中

# 1.先写from表单类

**public class** AnimalForm {  
  
 **private long id**;  
  
 @NotEmpty(message=**"动物名: 不能为空"**)  
 **private** String **oname**;  
  
 @Range(min = 1, message=**"数量: 必须大于0"**)  
 @NotNull(message=**"数量: 不能为空"**)  
 **private int ocount**;  
  
 @Size(max = 10, message=**"备注: 长度不能超过10个字符"**)  
 **private** String **memo**;  
  
 */\*\* 省略getter和setter \*\*/*

## 1.1代码解释：

### @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"。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。**

### @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。**

### @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”。**

### @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”。**

## 1.2.代码问题：

**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。  
\*/**

# 2.controller

**public class** AnimalController {  
  
 @RequestMapping(value = **"/list.html"**, method = RequestMethod.***GET***)  
 **public** String doAdd(Model model, @Valid AnimalForm form, BindingResult result){  
 System.***out***.println(**"动物名："** + form.getOname());  
 System.***out***.println(**"数量："** + form.getOcount());  
 System.***out***.println(**"备注："** + form.getMemo());  
 **if**(result.hasErrors()){  
 List<ObjectError> list = result.getAllErrors();  
 **for** (ObjectError error : list) {  
 System.***out***.println(error.getDefaultMessage());  
 }  
 model.addAttribute(**"MSG"**, **"出错啦！"**);  
 }**else**{  
 model.addAttribute(**"MSG"**, **"提交成功！"**);  
 }  
 **return "vialidate"**;  
 }  
}

## 2.1解释：

### 2.1.1@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。

### 2.1.2 BindingResult result，

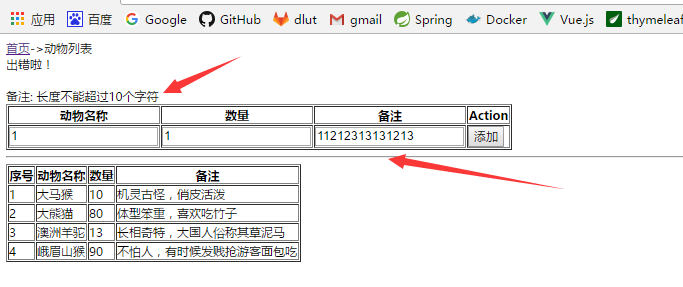
spring框架会将验证结果设置到该参数，并将该参数放到model传递给页面。  
 springMVC是非常灵活的，以下几种写法可以达到同样的效果：

# 3.html

<**br**/>  
<**div th:text="${MSG}"**>这里是信息提示.</**div**>  
<**br**/>  
<**div th:errors="${animalForm.oname}"**></**div**>  
<**div th:errors="${animalForm.ocount}"**></**div**>  
<**div th:errors="${animalForm.memo}"**></**div**>  
<**form id="iform" th:action="@{/list.html}" th:method="get" th:object="${animalForm}"**>  
 <**table border="1"**>  
 <**tr**>  
 <**th**>动物名称</**th**>  
 <**th**>数量</**th**>  
 <**th**>备注</**th**>  
 <**th**>Action</**th**>  
 </**tr**>  
 <**tr**>  
 <**td**><**input type="text" name="oname" value="" th:value="\*{oname}"**/></**td**>  
 <**td**><**input type="text" name="ocount" value="" th:value="\*{ocount}"**/></**td**>  
 <**td**><**input type="text" name="memo" value="" th:value="\*{memo}"**/></**td**>  
 <**td**><**input type="submit" value="添加"**/></**td**>  
 </**tr**>  
 </**table**>  
</**form**>

# 4.测试





# 5.微小总结

<http://www.cnblogs.com/yangzhilong/p/3724967.html> 讲的非常好，

1. @Null   被注释的元素必须为 **null**
2. @NotNull    被注释的元素必须不为 **null**
3. @AssertTrue     被注释的元素必须为 **true**
4. @AssertFalse    被注释的元素必须为 **false**
5. @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
6. @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
7. @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
8. @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
9. @Size(max=, min=)   被注释的元素的大小必须在指定的范围内
10. @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
11. @Past   被注释的元素必须是一个过去的日期
12. @Future     被注释的元素必须是一个将来的日期
13. @Pattern(regex=,flag=)  被注释的元素必须符合指定的正则表达式
15. Hibernate Validator 附加的 constraint
16. @NotBlank(message =)   验证字符串非**null**，且长度必须大于0
17. @Email  被注释的元素必须是电子邮箱地址
18. @Length(min=,max=)  被注释的字符串的大小必须在指定的范围内
19. @NotEmpty   被注释的字符串的必须非空
20. @Range(min=,max=,message=)  被注释的元素必须在合适的范围内

@DateTimeFormat(pattern="yyyy-MM-dd")

@Email     验证邮箱

# 6.代码加强

## 1.给数量定义不能为空，并且限定大小

|  |
| --- |
| *// 加强第二种* @NotBlank(message=**"数量: 不能为空"**) @Pattern(regexp=**"[1-9]{1,3}"**, message=**"数量X: 必须为正整数，并且0<X<1000"**) **private** String **ocount**; |

### @NotBlank

### @Pattern



# 7.自定义验证注解：以备注为例进行说明

## 1.注解其实都是借口，先写一个注解的接口性文件

@Retention(***RUNTIME***)  
@Target({ ***FIELD***, ***METHOD*** })  
@Constraint(validatedBy=MemoValidator.**class**)  
**public** @**interface** MyValidateMemo {  
  
 String message() **default "请输入正确的备注"**;  
 Class<?>[] groups() **default** {};  
 Class<? **extends** Payload>[] payload() **default** {};  
}

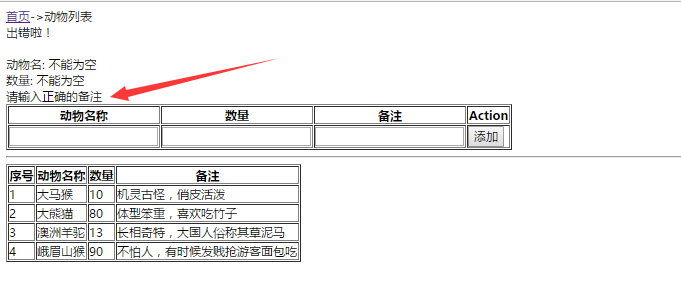
## 2.将自定义注解进行配置，并添加条件（只能输入"圈养，或者散养）

**public class** MemoValidator **implements** ConstraintValidator<MyValidateMemo, String> {  
 @Override  
 **public void** initialize(MyValidateMemo arg0) {  
 }  
 @Override  
 **public boolean** isValid(String arg0, ConstraintValidatorContext arg1) {  
 HashSet<String> memoSet = **new** HashSet<String>();  
 memoSet.add(**"圈养"**);  
 memoSet.add(**"散养"**);  
 **return** memoSet.contains(arg0);  
 }  
  
}

## 3.给备注进行注解自定义名称

@MyValidateMemo(message = **"备注不能为空，且只能填写\"圈养\"，或者\"散养\""**)  
**private** String **memo**;

#### 3.1解释：如果上面的message去掉的话，或自动显示我们的注解中默认的名称



## 4.测试

# 8.  BindingResult 中后台查看错误信息

1. **public** @ResponseBody User doRegister(@Valid User user, BindingResult result, Model model) {
2. **if** (result.hasErrors()) {
3. List<ObjectError> list = result.getAllErrors();
4. **for** (ObjectError error : list) {
5. System.out.println(error.getDefaultMessage());
6. }
7. **return** **null**;
8. }
9. System.out.println("注册..");
10. **return** user;
11. }