# 数据的验证

描述：

b/s系统中对http请求数据的校验多数在客户端进行，这也是出于简单及用户体验性上考虑，但是在一些安全性要求高的系统中服务端校验是不可缺少的。

Spring3支持JSR-303验证框架，JSR-303是JAVAEE6中的一项子规范，叫做Bean Validation，官方参考实现是Hibernate Validator（与Hibernate ORM 没有关系），JSR 303 用于对Java Bean 中的字段的值进行验证。

### 开发环境：

加入jar包

### 配置验证框架

1. 在springMVC中配置validator

```
<!-- 校验器 -->
<bean id="validator"
  class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
  <!-- 校验器-->
  <property name="providerClass" value="org.hibernate.validator.HibernateValidator" />
  <!-- 指定校验使用的资源文件，如果不指定则默认使用classpath下的ValidationMessages.properties -->
  <property name="validationMessageSource" ref="messageSource" />
 </bean>
<!-- 校验错误信息配置文件 -->
 <bean id="messageSource"
  class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
  <!-- 资源文件名-->
  <property name="basenames">
         <list>
   <--错误信息文件CustomValidationMessages.propertis-->
           <value>classpath:CustomValidationMessages</value>
         </list>
     </property>
  <!-- 资源文件编码格式 -->
  <property name="fileEncodings" value="utf-8" />
  <!-- 对资源文件内容缓存时间，单位秒 -->
  <property name="cacheSeconds" value="120" />
 </bean>
```

将验证器添加到适配器中

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

书写配置信息文件CustomValidationMessages.propertis

```
item.name.length.error=名称在1到20个字符之间
pic.is.null=请上传文件
```

在pojo中添加验证规则

```
public class Items {
    private Integer id;
    @Size(min=1,max=30,message="{item.name.length.error}")
    private String name;

    @NotEmpty(message="{pic.is.null}")
    private String pic;
```

捕获错误

修改Controller方法：

```
// 商品修改提交
 @RequestMapping("/editItemSubmit")
 public String editItemSubmit(@Validated  Items items,BindingResult result,
   @RequestParam("pictureFile") MultipartFile[] pictureFile,Model model)
   throws Exception {
 //如果存在校验错误则转到商品修改页面
  if(bindingResult.hasErrors()){
   model.addAttribute("errors", bindingResult);
}
   return "item/editItem";
  }
```

页面显示错误信息

```
<c:forEach items="${errors.allErrors}" var="error">
 ${error.defaultMessage }<br/>
</c:forEach>
```

注意：添加@Validated表示在对items参数绑定时进行校验，校验信息写入BindingResult中，在要校验的pojo后边添加BingdingResult， 一个BindingResult对应一个pojo，且BingdingResult放在pojo的后边。

## 分组校验

场景：

当有些时候，我们不需要将pojo中全部去进行校验，就可以使用分组校验去完成

定义分组：

```
//分组就是一个标识，这里定义一个接口：
public interface ValidGroup1 {

}
public interface ValidGroup2 {

}
```

指定分组校验：

```
public class Items {
    private Integer id;
//这里指定分组ValidGroup1，此@Size校验只适用ValidGroup1校验
    @Size(min=1,max=30,message="{item.name.length.error}",groups={ValidGroup1.class})
    private String name;
```

重写controller

```
// 商品修改提交
 @RequestMapping("/editItemSubmit")
 public String editItemSubmit(@Validated(value={ValidGroup1.class})
  Items items,BindingResult result,@RequestParam("pictureFile") MultipartFile[] pictureFile,Model model)
  throws Exception {
 //同上代码
 }
```

补充：

在@Validated中添加value={ValidGroup1.class}表示商品修改使用了ValidGroup1分组校验规则，也可以指定多个分组中间用逗号分隔，@Validated(value={ValidGroup1.class，ValidGroup2.class })

**校验注解**

@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

---

完