

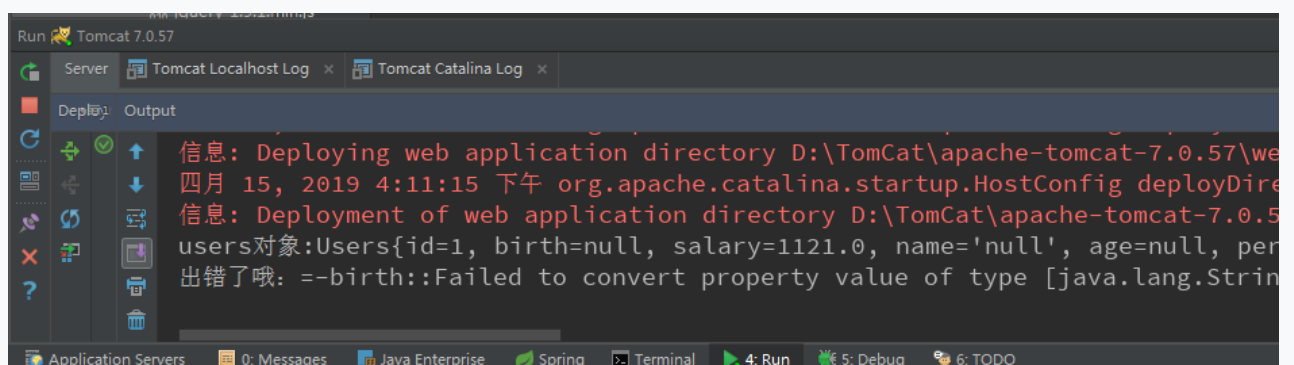
# 11 SpringMVC-类型转换出错异常 & JSR303数据校验 & Json数据

SpringMVC

## 一、类型转换出错

- 1、如需要的数字，但是输入的文字
- 2、将错误信息进行打印出来

```
1. @RequestMapping("/testDateTimeFormat")
2.     public String testDateTimeFormat(Users users, BindingResult result) {
3.         System.out.println("users对象:"+users);
4.         if (result.getErrorCount() > 0) {
5.             for (FieldError error : result.getFieldErrors()) {
6.                 System.out.println("出错了哦:==" + error.getField() + "::" + error.getDefaultMessage() + "\t");
7.             }
8.         }
9.         //调用用Service层
10.        return "success";
11.    }
```



## 二、JSR303数据校验

1、JSR 303 是 Java 为 Bean 数据合法性校验提供的标准框架，已经包含在 JavaEE 6.0

2、JSR 303 通过在 Bean 属性上标注类似于 @NotNull、@Max 等标准的注解指定校验规则，并通过标准的验证接口对 Bean 进行验证

3、如下的注解描述

Constraint	详细信息
@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(value)	被注释的元素必须符合指定的正则表达式

## 1、Hibernate Validator 扩展注解

1、Hibernate Validator 是 JSR 303 的一个参考实现，除支持所有标准的校验注解外，它还支持以下的扩展注解

Constraint	详细信息
@Email	被注释的元素必须是电子邮箱地址
@Length	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	被注释的元素必须在合适的范围内

## 2、验证






1、Spring 的 LocalValidatorFactoryBean 既实现了 Spring 的 Validator 接口，也实现了 JSR 303 的 Validator 接口。只要在 Spring 容器中定义了一个 LocalValidatorFactoryBean，即可将其注入到需要数据校验的 Bean 中

2、Spring 本身并没有提供 JSR303 的实现，所以必须将 JSR303 的实现者的 jar 包放到类路径下

3、`<mvc:annotation-driven/>` 会默认装配好一个 LocalValidatorFactoryBean，通过处理方法的入参上标注 @valid 注解即可让 Spring MVC 在完成数据绑定后执行数据校验的工作，所以要加配置 `<mvc:annotation-driven/>`

4、加入对应的jar包

javax.inject-2.5.0-b61.jar

名称	修改日期
 classmate-0.8.0.jar	2014/12/20 16:12
 hibernate-validator-5.0.0.CR2.jar	2014/12/20 16:12
 hibernate-validator-annotation-processor-5.0.0.CR2.jar	2014/12/20 16:12
 jboss-logging-3.1.1.GA.jar	2014/12/20 16:12
 validation-api-1.1.0.CR1.jar	2014/12/20 16:12

5、实体类加上了注解

```

1. import org.springframework.format.annotation.DateTimeFormat;
2. import org.springframework.format.annotation.NumberFormat;

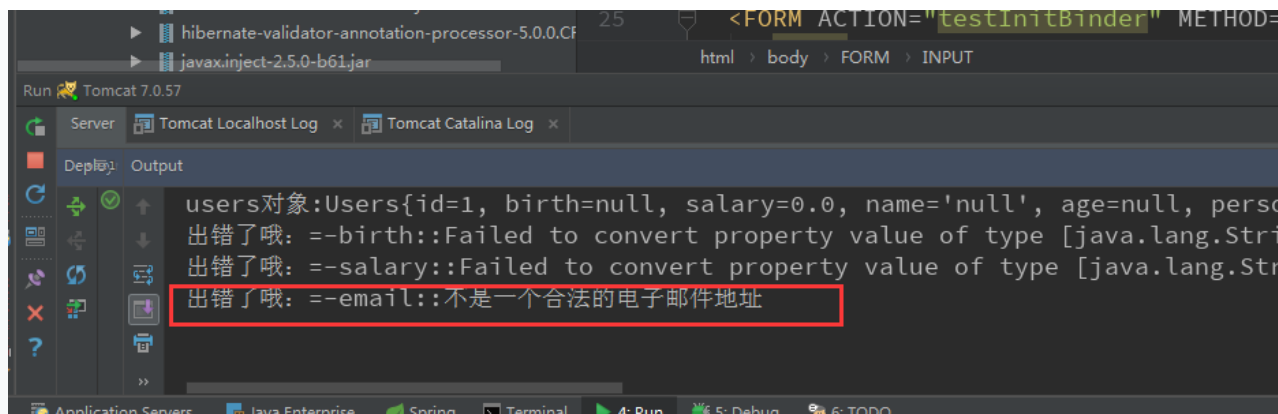
```

```

3.  import javax.validation.constraints.NotNull;
4.  import javax.validation.constraints.Past;
5.  import java.util.Date;
6.  public class Users {
7.      private Integer id;
8.      @Past
9.      @DateTimeFormat(pattern = "yyyy-MM-dd")
10.     private Date birth;
11.     @NotNull
12.     @NumberFormat(pattern = "#,###,###")
13.     private float salary;
14.
15.     @Email
16.     private String email;
17.     //对应的get set toString
18.
19.     =====Controller层=====
20.     @RequestMapping("/testDateTimeFormat")
21.     public String testDateTimeFormat(@Valid Users users, BindingResult
result){
22.         System.out.println("users对象:"+users);
23.         if(result.getErrorCount()>0){
24.             for (FieldError error : result.getFieldErrors()) {
25.                 System.out.println("出错了哦:=-"+error.getField()+"::"+e
rror.getDefaultMessage()+"\t");
26.             }
27.         }
28.         //调用用Service层
29.         return "success";
30.     }
31.     =====Controller层=====
32.     <FORM ACTION="testDateTimeFormat" METHOD="post">
33.         <INPUT TYPE="text" VALUE="1" NAME="id"><br/>
34.         <INPUT TYPE="text" VALUE="2019-2-1" NAME="birth"><br/>
35.         <INPUT TYPE="text" VALUE="23.43" NAME="salary"><br/>
36.         email<INPUT TYPE="text" VALUE="啊实打实" NAME="email"><br/>
37.         <input type="submit" value="submit">
38.     </FORM>

```

## 6、测试



7、需校验的 Bean 对象和其绑定结果对象或错误对象时成对出现的，它们之间不允许声明其他的入参

### 三、请求Json数据

1、加入对应的jar包

jackson-annotations.jar

jackson-core-2.8.1.jar

jackson-databind-2.8.1.jar

2、controller层编写

```

1.     @RequestMapping("/testJsom")
2.     @ResponseBody
3.     public      ArrayList<Animal> testJsom(Animal animal){
4.         System.out.println("对象:"+animal);
5.         //调用用Service层
6.         ArrayList<Animal> animals = new ArrayList<>();
7.         animals.add(animal);
8.         return animals;
9.     }
10.    =====页面请求=====
11.    <script type="text/javascript" src="static/jquery-1.9.1.min.js"
12.    ></script>
13.    <script type="text/javascript">
14.        $(function(){
15.            $("#testJsom").click(function(){
16.                var url = this.href;
17.                var args = {
18.                    'id':1,

```

```

18.         'name':'下午'
19.     };
20.     alert(url);
21.     $.post(url,args,function(data){
22.         alert(data);
23.         for(var i = 0; i < data.length; i++){
24.             var id = data[i].id;
25.             var lastName = data[i].name;
26.             alert(id + ": " + lastName);
27.         }
28.     });
29.     return false;
30. });
31. })
32. </script>
33. </head>
34. <body>
35. <a href="testJsom" id="testJson">Test Json</a>
36. <br><br>

```

### 3、spring.xml文件的编写

```

1.     <!-- 从请求和响应读取/编写字符串 -->
2.     <bean id="stringHttpMessage"
3.         class="org.springframework.http.converter.StringHttpMessageConverter">
4.         <property name="supportedMediaTypes">
5.             <list>
6.                 <value>text/plain;charset=UTF-8</value>
7.             </list>
8.         </property>
9.     </bean>
10.    <!-- 用于将对象转化为JSON -->
11.    <bean id="jsonConverter"
12.        class="org.springframework.http.converter.json.MappingJackson2HttpMessage
13.        Converter"></bean>
14.
15.    <bean
16.        class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHan
17.        dlerAdapter">
18.        <property name="messageConverters">
19.            <list>
20.                <ref bean="stringHttpMessage"/>
21.                <ref bean="jsonConverter"/>

```

```

17.         </list>
18.     </property>
19. </bean>
20.
21. =====也可以在MVC中指定对应的Bean=====
22.     <mvc:annotation-driven conversion-service="conversionService" >
23.         <mvc:message-converters>
24.             <ref bean="stringHttpMessage"/>
25.             <ref bean="jsonConverter"/>
26.         </mvc:message-converters>
27.     </mvc:annotation-driven>

```

## 1、原理

2、 `HttpMessageConverter<T>` 是 Spring3.0 新添加的一个接口，负责将请求信息转换为一个对象（类型为 T），将对象（类型为 T）输出为响应信息

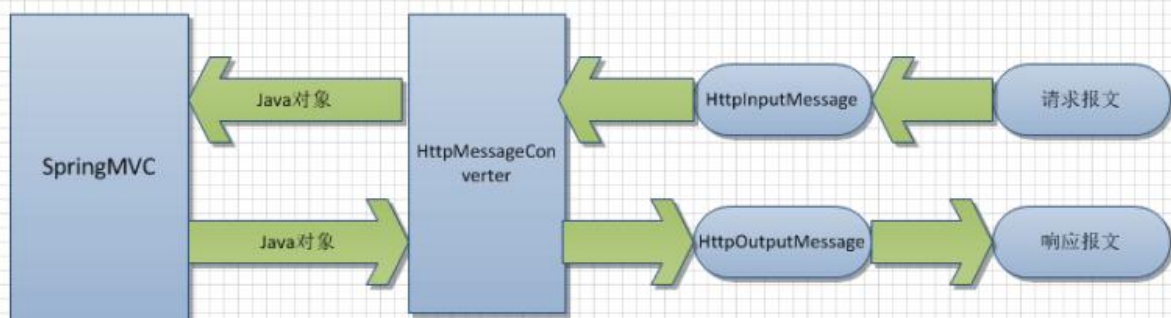
1、使用的 `HttpInputMessage` 与 `HttpOutputMessage` 之间的转换

```

1. public interface HttpOutputMessage extends HttpMessage {
2.     OutputStream getBody() throws IOException;
3. }
4. =====
5. public interface HttpInputMessage extends HttpMessage {
6.     InputStream getBody() throws IOException;
7. }

```

## 2、原理图



# 1、HttpMessageConverter的实现类

## 1、在@ResponseBody的方法返回值上，进行DUG测试

