Spring MVC

-数据转换和校验

课程目标

- 掌握SpringMVC的数据转换
- 了解SpringMVC的数据校验

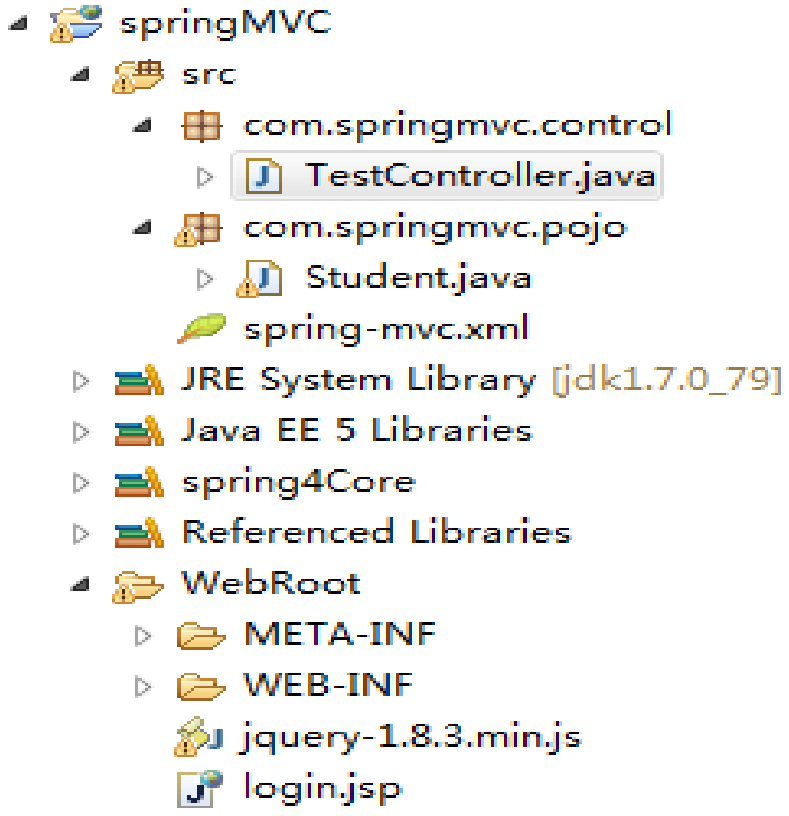
SpringMVC的数据转换

SpringMVC的数据转换

- 在SpringMVC开发中，可能遇到比较多的问题就是前台与后台实体类之间日期转换处理的问题了，有时很令人头疼，对问题进行跟踪后，会发现是日期类型转换失败“映射”不到对应的持久类的日期属性上造成的；同理，国内外对数字的处理方式不同同样可能造成同样的错误。
- 这时，我们可以使用spring内建的转换机制进行处理。

实例

□项目结构


springMVC
src
com.springmvc.control
TestController.java
com.springmvc.pojo
Student.java
spring-mvc.xml
JRE System Library [jdk1.7.0_79]
Java EE 5 Libraries
spring4Core
Referenced Libraries
WebRoot
META-INF
WEB-INF
jquery-1.8.3.min.js
login.jsp

实例

View：

```
<html>

  <body>
    <form action="testConvert" method="post">
        输入姓名：<input name="name">  <br/><br/>
        输入年龄：<input name="age">  <br/><br/>
        输入生日：<input name="birthday"> <br/><br/>
        输入缴费金额<input name="money"> <br/><br/>
    <input type="submit" value="提交">
    </form>
    <br /><br />
  </body>
</html>
```

实例

- Pojo类：

```
public class Student {
    private String name;
    private Integer age;
    private Date birthday;
    private Double  money;

    //省略getter/setter方法
```

实例

Controller

```
@Controller
public class TestController{


    @RequestMapping(value="/testConvert")
    @ResponseBody
    public String testConvert(Student student){
        return student.toString();
    }

}
```

正确的输入

- Spring能够自动将符合格式的日期字符串转换为对应的日期类型，默认格式模板为“yyyy/mm/dd”.
- 数字的默认格式模板为“###.#”.

实例

- **当需求变更为：**
  - 日期类型模板：“yyyy-mm-dd”。
  - 数字类型模板：“###.#”。

- Spring提供了多种方式进行转换

方法1：使用格式化注解

- Spring提供了内建的格式化注解供使用。
- **@DateTimeFormat和@NumberFormat**
- 1.需配置**<mvc:annotation-driven />**
- 2.使用在需要转换类型的字段上方。

@DateTimeFormat

- @DateTimeFormat 注解有3个可选的属性：
  - style
  - Pattern
  - iso。

@DateTimeFormat

- 属性style允许我们使用两个字符的字符串来表明怎样格式化日期和时间。第一个字符表明了 日期的格式，第二个字符表明了时间的格式。
- 例：

```
public class Student {
        private String name;
        private Integer age;
        @DateTimeFormat(style="SS")
        private Date birthday;
        private Double  money;

        //省略getter/setter方法
```

@DateTimeFormat

描述	字符串值	示例输出
短格式（这是缺省值）	SS	8/30/64 11:24 AM
中等格式	MM	Aug 30, 1964 11:24:41 AM
长格式	LL	August 30, 1964 11:24:41 AM CDT
完整格式	FF	Sunday, August 30, 1964 11:24:41 AM CDT
使用短横线省略日期或时间	M-	Aug 30, 1964

@DateTimeFormat

- Pattern 属性允许我们使用自定义的日期/时间格式。该属性的值遵循java标准的date/time格式规范。缺省的该属性的值为空，也就是不进行特殊的格式化。该属性较为常用。推荐

- 例：

```
public class Student {
    private String name;
    private Integer age;
    @DateTimeFormat(pattern="yyyy-mm-dd")
    private Date birthday;
    private Double  money;


    //省略getter/setter方法
```

@DateTimeFormat

- .ISO属性允许使用org.springframework.format.annotation.DateTimeFormat.ISO枚举值来使用ISO标准的日期/时间格式来格式化。

- 例：

```
public class Student {
    private String name;
    private Integer age;
    @DateTimeFormat(iso=ISO.DATE)
    private Date birthday;
    private Double  money;
```

@NumberFormat

- @NumberFormat注解有两个可选的属性：
  - style
  - pattern。

@NumberFormat

- style属性是一个NumberFormat.Style枚举值，可以是以下的三个值之一：

NumberFormat.Style 枚举值	是否缺省值
NUMBER	是
CURRENCY	否
PERCENT	否

@NumberFormat

- Pattern 属性允许我们使用自定义的模板。

- 针对上例：

```
public class Student {
    private String name;
    private Integer age;
    @DateTimeFormat(pattern="yyyy-mm-dd")
    private Date birthday;
    @NumberFormat(style= Style.CURRENCY,pattern="###,#")
    private Double  money;


    //省略getter/setter方法
```

方法二：控制器中加入数据绑定代码

- **使用WebDataBinder进行控制器级别注册PropertyEditor（控制器独享）**
- **@InitBinder的作用：**
- 使用注解@InitBinder标注，那么spring mvc在绑定表单之前，都会先注册这些编辑器

方法二：控制器中加入数据绑定代码

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    dateFormat.setLenient(false);
    binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, true));
    NumberFormat numberFormat = NumberFormat.getInstance();
    numberFormat.setGroupingUsed(true);
    binder.registerCustomEditor(Double.class,
            new CustomNumberEditor(Double.class,numberFormat,true));
}
```

方法三：自定义类型转换器

- **自定义类型转换器开发步骤：**
- **一.自定义类型转换器（实现Converter 接口）**
- **二.配置spring配置文件**
  - 1.注册ConversionService
  - 2. 使用ConfigurableWebBindingInitializer注册conversionService
  - 3.注册ConfigurableWebBindingInitializer到RequestMappingHandlerAdapter

自定义类型转换器

一、实现Converter<T,K>接口，重写convert方法

- T为convert方法的入参泛型类型
- K为convert方法的返回泛型类型

```
public class MyPhoneConverter implements Converter<String,Integer> {
    Pattern pattern = Pattern.compile("^(\\d{3,4})-(\\d{7,8})$");
    public Integer convert(String source) {

        if(!StringUtils.hasLength(source)) {
            //①如果source为空 返回null
            return null;
        }
        Matcher matcher = pattern.matcher(source);
        if(matcher.matches()){
            //②如果匹配 进行转换
            String[] temp = source.split("-");
            String str = temp[0]+temp[1];
            Integer phoneNumber = Integer.parseInt(str);
            return phoneNumber;
        } else {
            //③如果不匹配 转换失败
            throw
            new IllegalArgumentException(String.format("类型转换失败，需要格式[010-12345678]
        }
```

自定义类型转换器

二.配置spring配置文件

- 1.注册ConversionService

```
<!-- 1、注册ConversionService -->
<mvc:annotation-driven conversion-service="conversionService" />
<bean id="conversionService"
    class="org.springframework.context.support.ConversionServiceFactoryBean">
    <property name="converters">
        <list>
            <bean class="com.springmvc.pojo.MyPhoneConverter" />
        </list>
    </property>
</bean>
```

自定义类型转换器

- 2. 使用ConfigurableWebBindingInitializer注册conversionService

```
<!-- 2、使用ConfigurableWebBindingInitializer注册conversionService -->
<bean id="webBindingInitializer"
    class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer">
    <property name="conversionService" ref="conversionService" />
</bean>
```

自定义类型转换器

- 3.注册ConfigurableWebBindingInitializer到RequestMappingHandlerAdapter

```
<!-- 3、注册ConfigurableWebBindingInitializer到RequestMappingHandlerAdapter -->
<bean
    class="org.springframework.web.servlet.mvc.method.
    annotation.RequestMappingHandlerAdapter">
    <property name="webBindingInitializer" ref="webBindingInitializer" />
</bean>
```

Spring数据校验

数据校验

- 参数校验是我们程序开发中必不可少的过程。
- 用户在前端页面上填写表单时，前端js程序会校验参数的合法性，当数据到了后端，为了防止恶意操作，保持程序的健壮性，后端同样需要对数据进行校验。
- 后端参数校验最简单的做法是直接在业务方法里面进行判断，当判断成功之后再继续往下执行。但这样带给我们的是代码的耦合，冗余。当我们多个地方需要校验时，我们就需要在每一个地方调用校验程序,导致代码很冗余，且不美观。

- 那么如何优雅的对参数进行校验呢？JSR303就是为了解决这个问题出现的，下面主要是介绍 JSR303，Hibernate Validator 等校验工具的使用，以及自定义校验注解的使用。

什么是JSR

- JSR是Java Specification Requests的缩写，意思是Java 规范提案。是指向JCP(Java Community Process)提出新增一个标准化技术规范的正式请求。任何人都可以提交JSR，以向Java平台增添新的API和服务。JSR已成为Java界的一个重要标准。

什么是JSR303- Bean Validation

- JSR303 是一套JavaBean参数校验的标准，它定义了很多常用的校验注解，我们可以直接将这些注解加在我们JavaBean的属性上面，就可以在需要校验的时候进行校验了。

Bean Validation 中内置的 constraint

- @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

Bean Validation 中内置的 constraint

- @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

Hibernate Validator

- Hibernate Validator 是 Bean Validation 的参考实现 . Hibernate Validator 提供了 JSR 303 规范中所有内置 constraint 的实现，除此之外还有一些附加的 constraint。

Hibernate Validator 附加的 constraint

- @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

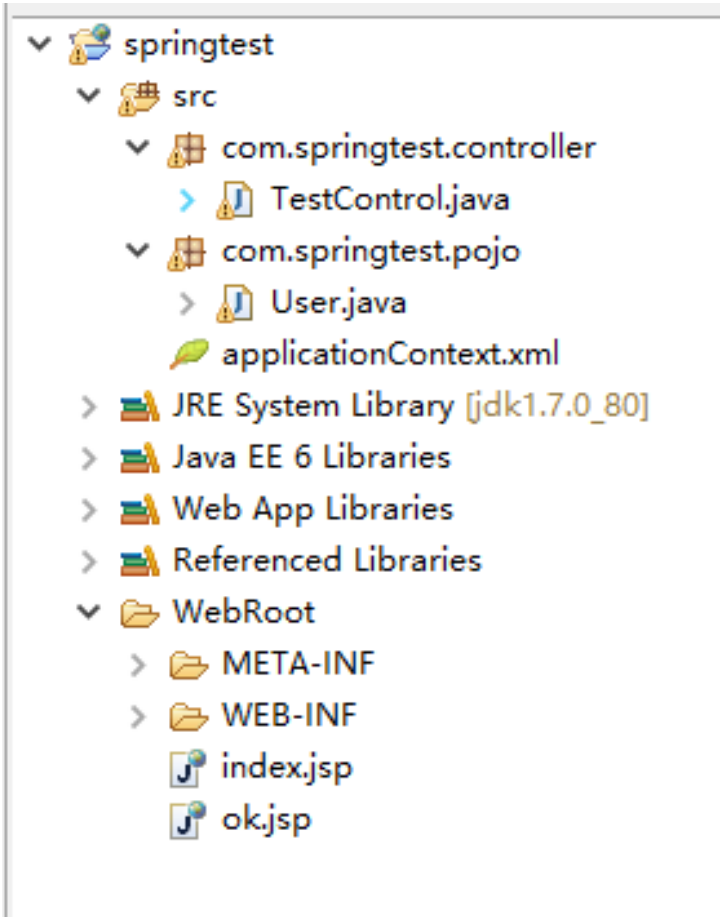
实例

- 所需jar

  - validation-api-1.1.0.final.jar
  - hibernate-validator-5.1.3.Final.jar
  - jboss-logging-3.1.3.ga.jar
  - classmate-1.1.0.jar

实例

□项目结构


springtest
src
com.springtest.controller
TestControl.java
com.springtest.pojo
User.java
applicationContext.xml
JRE System Library [jdk1.7.0_80]
Java EE 6 Libraries
Web App Libraries
Referenced Libraries
WebRoot
META-INF
WEB-INF
index.jsp
ok.jsp

实例

Controller:

```
@Controller
@RequestMapping("/test1")
public class TestControl {
    @RequestMapping(value="/test",method=RequestMethod.POST)
    //1.在需要验证的pojo前加上@Valid注解
    /*2.BindingResult必须和@Valid配合使用，且在@Valid后，用于存放校验信息，并可提取错误信息；
     * 如不使用BindingResult，spring只会抛出异常*/
    public String test(@Valid User user, BindingResult result,ModelMap map){
        if (result.hasErrors()){
                List<ObjectError> errorList = result.getAllErrors();
                for(ObjectError error : errorList){
                    map.addAttribute(error.getCodes()[1].substring(6),
                            error.getDefaultMessage());
                }

                return "index";
            }
        System.out.println(user);
        return "ok";
    }
```

实例

- Pojo：

```
ıblic class User {
        @NotNull(message="姓名不能为空")
        private String name;

        @Range(min=18,max=80,message="年龄须在18-80岁之间")
        private Integer age;

        @Range(min=1,max=50,message="工号须在1-50之间")
        private Integer uid;
        //省略getter和setter方法
```

实例

View：index.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>

  </head>

  <body>
  <form action="test1/test" method="post">
    姓名：<input name="name" >${name}<br><br>
    age：<input name="age" >${age}<br><br>
    uid：<input name="uid" >${uid}<br><br>
    <input type="submit" value="提交">
  </form>
   <br>
  </body>
</html>
```

实例

applicationContext.xml

```
    <!--配置MVC注解扫描 -->
<context:component-scan base-package="com.springtest.controller">
</context:component-scan>

<mvc:default-servlet-handler />
<mvc:annotation-driven />
<!-- 注册校验器，spring会自动加载 -->
 <bean id="validator"
    class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean" />

<!--配置请求的前后缀 -->
<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
```

谢谢！