# 表单标签—了解

通过 SpringMVC 的表单标签可以实现将模型数据中的属性和 HTML 表单元素相绑定，以实现表单数据更便捷编辑和表单值的回显

<%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"*%>

一般情况下，通过 GET 请求获取表单页面，而通过POST 请求提交表单页面，因此获取表单页面和提交表单页面的 URL 是相同的。只要满足该最佳条件的契约，<form:form> 标签就无需通过 action 属性指定表单提交的 URL

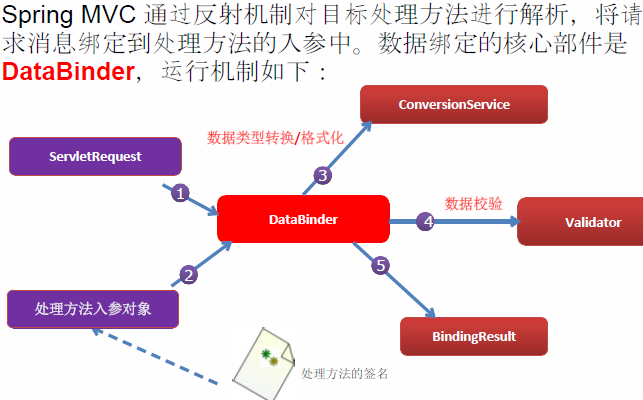
可以通过 modelAttribute 属性指定绑定的模型属性，若没有指定该属性，则默认从 request 域对象中读取command 的表单 bean，如果该属性值也不存在，则会发生错误。



# 数据转换

### (1)数据绑定流程





### (2)内置转换



### (3)自定义类型转换器—了解

ConversionService 是 Spring 类型转换体系的核心接口。

可以利用 ConversionServiceFactoryBean 在 Spring 的 IOC容器中定义一个 ConversionService. Spring 将自动识别出IOC 容器中的 ConversionService，并在 Bean 属性配置及Spring MVC 处理方法入参绑定等场合使用它进行数据的转换

可通过 ConversionServiceFactoryBean 的 converters 属性注册自定义的类型转换器

<form action=*"testConversionServiceConverer"* method=*"POST"*>

<!-- lastname-email-gender-department.id 例如: GG-gg@qq.com-0-105 -->

自定义转换器：Employee: <input type=*"text"* name=*"employee"*/>

<input type=*"submit"* value=*"Submit"*/>

</form>

@Controller

**public** **class** SpringMVCTest {

@Autowired

**private** EmployeeDao employeeDao;

@RequestMapping("/testConversionServiceConverer")

**public** String testConverter(@RequestParam("employee") Employee employee){

System.*out*.println("save" + employee);

employeeDao.save(employee);

**return** "redirect:/emps";

}

}

Spring 定义了 3 种类型的转换器接口，实现任意一个转换器接口都可以作为自定义转换器注册到ConversionServiceFactroyBean 中：

Converter<S,T>：将 S 类型对象转为 T 类型对象

ConverterFactory：将相同系列多个 “同质” Converter 封装在一起。如果希望将一种类型的对象转换为另一种类型及其子类的对象（例如将 String 转换为 Number 及 Number 子类

（Integer、Long、Double 等）对象）可使用该转换器工厂类

GenericConverter：会根据源类对象及目标类对象所在的宿主类中的上下文信息进行类型转换

/\*\*

\* 自定义类型转换器

\* **@author** Administrator

\*

\*/

@Component

**public** **class** EmployeeConverter **implements** Converter<String, Employee>{

@Override

**public** Employee convert(String source) {

**if**(source != **null**){

String [] vals = source.split("-");

//GG-gg@qq.com-0-105

**if**(vals != **null** && vals.length == 4){

String lastName = vals[0];

String email = vals[1];

Integer gender = Integer.*parseInt*(vals[2]);

Department department = **new** Department();

department.setId(Integer.*parseInt*(vals[3]));

Employee employee = **new** Employee(**null**, lastName, email, gender, department);

System.*out*.println(source + "--convert--" + employee);

**return** employee;

}

}

**return** **null**;

}

}

<mvc:annotation-driven conversion-service=*"conversionService"*></mvc:annotation-driven>

<!-- 配置 ConversionService -->

<bean id=*"conversionService"*

class=*"org.springframework.format.support.FormattingConversionServiceFactoryBean"*>

<property name=*"converters"*>

<set>

<ref bean=*"employeeConverter"*/>

</set>

</property>

</bean>

### (4) 关于 mvc:annotation-driven

<mvc:annotation-driven /> 会自动注册**RequestMappingHandlerMapping**

、**RequestMappingHandlerAdapter** 与**ExceptionHandlerExceptionResolver** 三个bean。

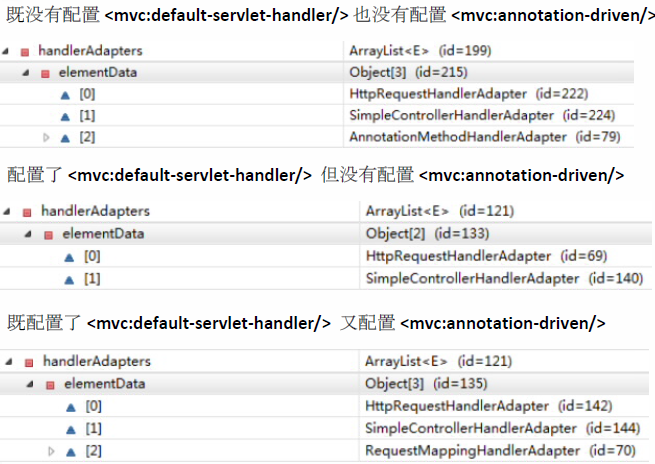
还将提供以下支持：

支持使用 **ConversionService** 实例对表单参数进行类型转换

支持使用 **@NumberFormat annotation**、**@DateTimeFormat**注解完成数据类型的格式化

支持使用 **@Valid** 注解对 JavaBean 实例进行 JSR 303 验证

支持使用 **@RequestBody** 和 **@ResponseBody** 注解



### (5) @InitBinder

由 **@InitBinder** 标识的方法**•** ，可以对 **WebDataBinder** 对象进行初始化。WebDataBinder 是 DataBinder 的子类，用于完成由表单字段到 JavaBean 属性的绑定

**@InitBinder**方法不能有返回值，它必须声明为void• 。

**@InitBinder**方法的参数通常是是 **WebDataBinder**



# 数据格式化

首先创建：**<mvc:annotation-driven/>**

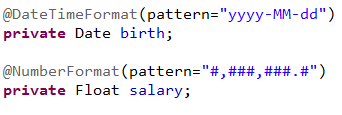
<mvc:annotation-driven></mvc:annotation-driven>

## 日期格式化



## 数值格式化



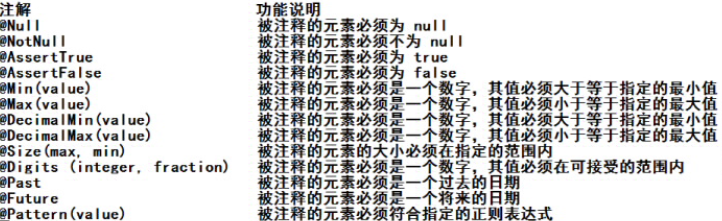


# 数据检验

### (1) JSR 303

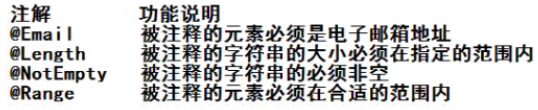
**JSR 303** 是 Java 为 Bean 数据合法性校验提供的标准框架，它已经包含在 JavaEE 6.0 中 .

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



### (2) Hibernate Validator 扩展注解

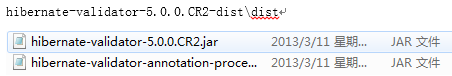
Hibernate Validator 是 JSR 303 的一个参考实现，除支持所有标准的校验注解外，它还支持以下的扩展注解

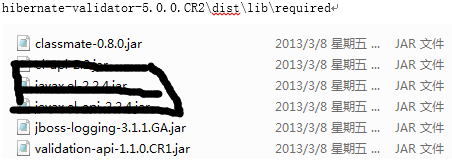


### (3) 如何校验 ? 注解 ?

①. 使用 JSR 303 验证标准

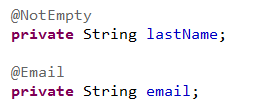
②. 加入 hibernate validator 验证框架的 jar 包





③. 在 SpringMVC 配置文件中添加 <mvc:annotation-driven />

④. 需要在 bean 的属性上添加对应的注解



⑤. 在目标方法 bean 类型的前面添加 @Valid 注解



### (4) 验证出错

注意: 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

Errors 接口提供了获取错误信息的方法，如 getErrorCount() 或getFieldErrors(String field) BindingResult 扩展了 Errors 接口

@RequestMapping(value="emp" , method=RequestMethod.*POST*)

**public** String save(@Valid Employee employee ,BindingResult result

,Map<String, Object> map){



**if**(result.getErrorCount() > 0){

System.*out*.println("出错了!");

**for**(FieldError error:result.getFieldErrors()){

System.*out*.println(error.getField() + ":" + error.getDefaultMessage());

}

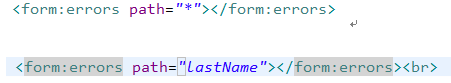
//若验证出错, 则转向定制的页面

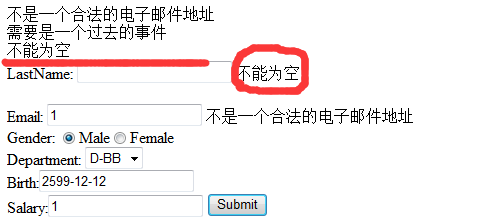
map.put("departments", departmentDao.getDepartments());

**return** "input";

}

### (5) 错误消息 ? 如何显示





### (6) 提示消息的国际化—了解

每个属性在数据绑定和数据校验发生错误时，都会生成一个对应的 FieldError 对象。

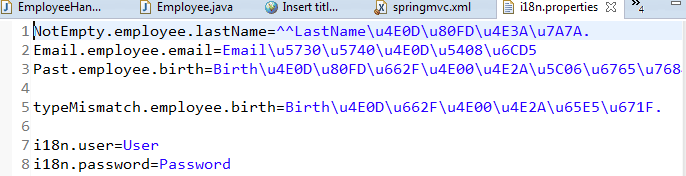
当一个属性校验失败后，校验框架会为该属性生成 4 个消息代码，这些代码以校验注解类名为前缀，结合modleAttribute、属性名及属性类型名生成多个对应的消息代码：例如 User 类中的 password 属性标准了一个 @Pattern 注解，当该属性值不满足 @Pattern 所定义的规则时, 就会产生以下 4个错误代码：

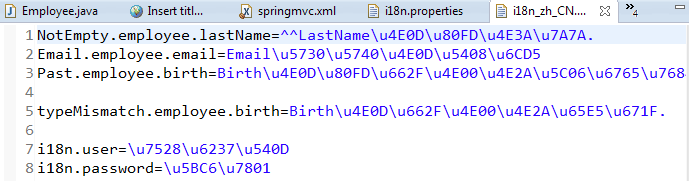
Pattern.user.password ；Pattern.password ；

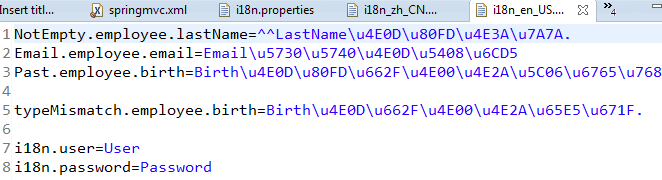
Pattern.java.lang.String ；Pattern



当使用 Spring MVC 标签显示错误• 消息时， Spring MVC 会查看WEB 上下文是否装配了对应的国际化消息，如果没有，则显示默认的错误消息，否则使用国际化消息。







若数据类型转换或数据格式转换时发生错误，或该有的参数不存在，或调用处理方法时发生错误，都会在隐含模型中创建错误消息。其错误代码前缀说明如下：

注册国际化资源文件  
 <!-- 配置国际化资源文件 -->

<bean id=*"messageSource"*

class=*"org.springframework.context.support.ResourceBundleMessageSource"*>

<property name=*"basename"* value=*"i18n"*></property>

</bean>