1. Converter 将一种类型转换成另外一种类型。

创建Converter必须实现Converter接口

public interface Converter<S, T> 从S类型转换成T类型

实现T convert(S source)方法

示例: 使请求参数里的字符串的日期对象都转换成Date对象

public class StringToDateConverter implements Converter<String, Date> {  
  
 private String datePattern;  
  
 public StringToDateConverter(String datePattern) {  
 this.datePattern = datePattern;  
 }  
  
 @Override  
 public Date convert(String s) {  
 try {  
 SimpleDateFormat dateFormat = new SimpleDateFormat(datePattern);  
 dateFormat.setLenient(false);  
 return dateFormat.parse(s);  
 } catch (ParseException e) {  
 throw new IllegalArgumentException(e);  
 }  
 }  
}

spring mvc配置文件里的配置：

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

和

<bean id="conversionService" class="org.springframework.context.support.ConversionServiceFactoryBean">  
 <property name="converters">  
 <set>  
 <bean class="com.converter.StringToDateConverter">  
 <constructor-arg type="java.lang.String" value="MM-dd-yyyy" />  
 </bean>  
 </set>  
 </property>  
</bean>

1. Formatter 把String转换成另外一种类型（Formatter更适合Spring MVC应用）

要实现 public interface Formatter<T>

实现T parse(String text, Locale locale) //利用指定的Locale将一个String解析成目标类型

String print(T object, Locate locate) //返回目标对象的字符串表示

示例：使请求参数里的字符串的日期对象都转换成Date对象

Spring mvc配置文件里的配置：

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

<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">  
 <property name="formatters">  
 <set>  
 <bean class="com.converter.DateFormatter">  
 <constructor-arg type="java.lang.String" value="MM-dd-yyyy" />  
 </bean>  
 </set>  
 </property>  
</bean>

Formatter实现类

public class DateFormatter implements Formatter<Date> {  
 private String datePattern;  
 private SimpleDateFormat dateFormat;  
  
 public DateFormatter(String datePattern) {  
 this.datePattern = datePattern;  
 dateFormat = new SimpleDateFormat(datePattern);  
 dateFormat.setLenient(false);  
 }  
  
 @Override  
 public Date parse(String s, Locale locale) throws ParseException {  
 return dateFormat.parse(s);  
 }  
  
 @Override  
 public String print(Date date, Locale locale) {  
 return dateFormat.format(date);  
 }  
}

1. FormatterRegistrar：注册多个Formatter，不再需要一个个在spirng mvc配置文件里配置。

示例： 需要放在driven前面

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

<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">  
 <property name="formatterRegistrars">  
 <set>  
 <bean class="com.converter.MyFormatterRegistrar">  
 <constructor-arg type="java.lang.String" value="MM-dd-yyyy" />  
 </bean>  
 </set>  
 </property>  
</bean>

public class MyFormatterRegistrar implements FormatterRegistrar {  
  
 private String datePattern;  
 public MyFormatterRegistrar(String datePattern) {  
 this.datePattern = datePattern;  
 }  
  
 @Override  
 public void registerFormatters(FormatterRegistry formatterRegistry) {  
 formatterRegistry.addFormatter(new DateFormatter(datePattern));  
 //add more formatter  
 }  
}

1. Validator 验证器：检验传入参数的属性是否符合要求。

Public interface Validator{

Boolean supports(Class<?> clazz); //如果验证器可以处理指定的class，此方法会返回true

Void validate(Object target, Errors errors); //会验证目标对象，并将验证错误填入Errors对象

}

Errors对象里面包含FieldError对象(被验证对象中的某个属性相关的一个错误)和ObjectError对象（）

给Errors对象添加错误的最容易的方法是：在Errors对象上调用一个reject或则rejectValue方法。

void reject(String errorCode); //往FieldError中添加一个ObjectError和rejectValue

void reject(String errorCode, String defaultMessage);

void rejectValue(String field, String errorCode);

void rejectVaalue(String field, String errorCode, String defaultMessage);

1. ValidationUtils 有助于编写Spring验证器

有以下方法：

这些方法是从Errors对象获取目标对象，从而判断目标对象的值是否为空

public static void rejectIfEmpty(Errors errors, String field, String errorCode) {}  
  
public static void rejectIfEmpty(Errors errors, String field, String errorCode, String defaultMessage) {}  
  
public static void rejectIfEmpty(Errors errors, String field, String errorCode, Object[] errorArgs) {}  
  
public static void rejectIfEmpty(Errors errors, String field, String errorCode, Object[] errorArgs, String defaultMessage) {  
}  
  
public static void rejectIfEmptyOrWhitespace(Errors errors, String field, String errorCode) {}  
  
public static void rejectIfEmptyOrWhitespace(Errors errors, String field, String errorCode, String defaultMessage) {}  
  
public static void rejectIfEmptyOrWhitespace(Errors errors, String field, String errorCode, Object[] errorArgs) {}  
  
public static void rejectIfEmptyOrWhitespace(Errors errors, String field, String errorCode, Object[] errorArgs, String defaultMessage) {}  
}

1. Validator示例：

Validator实现类

public class ProductValidator implements Validator {  
 @Override  
 public boolean supports(Class<?> aClass) {  
 return Product.class.isAssignableFrom(aClass);  
 }  
  
 @Override  
 public void validate(Object o, Errors errors) {  
 Product product = (Product) o;  
 ValidationUtils.*rejectIfEmpty*(errors, "name", "productname.required");  
 ValidationUtils.*rejectIfEmpty*(errors, "date", "productdate.required");  
  
 Date date = product.getDate();  
 if (date != null) {  
 if (date.after(new Date())) {  
 System.*out*.println("error date");  
 errors.rejectValue("date", "productdate.invalid");  
 }  
 }  
 }  
}

controller调用validator:

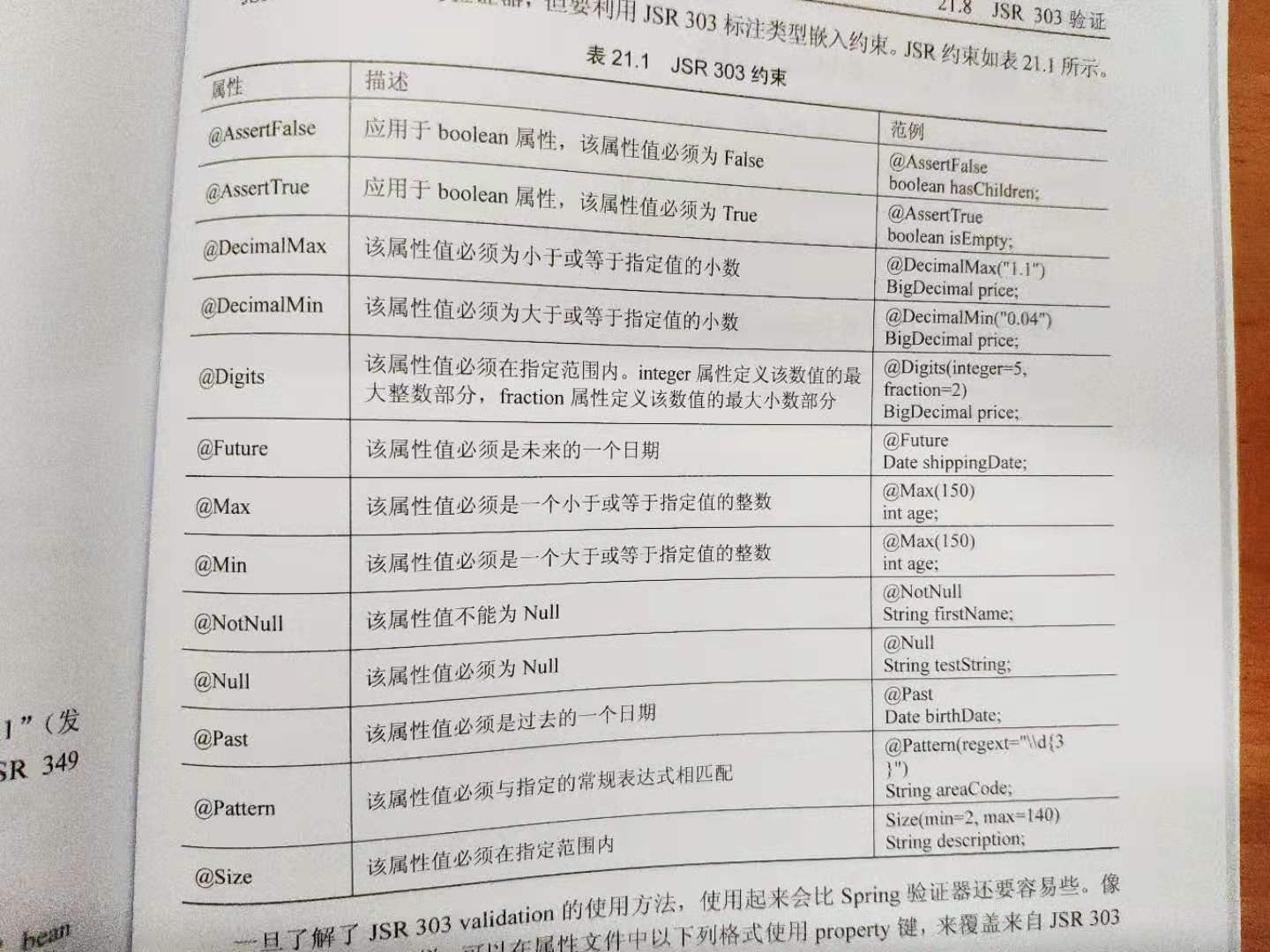
@RequestMapping("date\_test")  
@ResponseBody  
public String datetest(Product product, BindingResult bindingResult) { //为了验证有没有产生错误信息，必须绑定bingdingResult  
 ProductValidator productValidator = new ProductValidator();  
 productValidator.validate(product, bindingResult); //bingdingResult, 也就是Errors对象的一个实例  
  
 if (bindingResult.hasErrors()) {  
 FieldError fieldError = bindingResult.getFieldError();  
 System.*out*.println("Code:" + fieldError.getCode() + ", field:" + fieldError.getField());  
 return "Error";  
 }  
 return "datetest";  
}

1. @InitBinder注解

在controller类里面实现该方法，该方法绑定的方法会作用于该controller的所有方法。

@InitBinder  
public void initBinder(WebDataBinder binder) {  
 binder.setValidator(new ProductValidator());  
 binder.validate();  
}

1. JSR303验证



可以在配置文件里配置错误信息：

如：Size.product.name 、Past.product.productionDate

在pojo里使用@xxx注解

public class Product {  
 private Date date;  
 @Size(min = 1, max = 10)  
 private String name;  
}

在controller类方法里使用@valid注解进行验证，使用BindingResult获取错误信息

@RequestMapping("date\_test")  
@ResponseBody  
public String datetest(@Valid Product product, BindingResult bindingResult) { //为了验证有没有产生错误信息，必须绑定bingdingResult  
 if (bindingResult.hasErrors()) {  
 FieldError fieldError = bindingResult.getFieldError();  
 System.*out*.println(fieldError.getCode() + fieldError.getObjectName() + fieldError.getField() + fieldError.getDefaultMessage());  
 return "Error";  
 }  
 return "datetest";  
}

示例:

[**SpringMVC Date数据类型绑定**](https://www.cnblogs.com/JNUX/p/6848311.html)

数据绑定应该算是[spring](http://lib.csdn.net/base/javaee) MVC的特点之一吧~简单易用且功能强大，极大地简化了我们编程人员对于用户输入数据的接收及转换。

早先版本的Spring中的数据绑定完全都是基于PropertyEditor的。而Spring3中引入了Converter，用来替代PropertyEditor完成类型转换。

那么我们也依照这个顺序，先来讲讲基于传统的PropertyEditor来实现日期类型的数据绑定。

在Spring MVC中，我们可以通过WebDataBinder来注册自定义的PropertyEditor，从而添加对应的请求参数绑定。有两种方式：

1、使用@InitBinder注解@Controller中的方法 2、自定义WebBindingInitializer来提供一个全局的数据绑定规则。

**1、使用@InitBinder注解**

@InitBinder

public void initBinder(WebDataBinder binder){

binder.registerCustomEditor(Date.class, new DateEditor());

}

[复制代码](javascript:void(0);)

public class DateEditor extends PropertyEditorSupport {

@Override

public void setAsText(String text) throws IllegalArgumentException {

SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

Date date = null;

try {

date = format.parse(text);

} catch (ParseException e) {

format = new SimpleDateFormat("yyyy-MM-dd");

try {

date = format.parse(text);

} catch (ParseException e1) {

e1.printStackTrace();

}

}

setValue(date);

}

}

[复制代码](javascript:void(0);)

这里我们将DateEditor提出来封装成一个类方便重用。

另外这里有个try...catch的小妙用，就是首先以"yyyy-MM-dd HH:mm:ss"的形式来解析用户输入的参数，若解析失败则转以"yyyy-MM-dd"的形式来解析。这样的逻辑就可以同时处理"yyyy-MM-dd HH:mm:ss"和"yyyy-MM-dd"形式的日期数据，我想在一般的中文系统中，这两种形式应该是最常用的了吧。

添加如上代码之后，@InitBinder所在的Controller就可以自动绑定日期类型的数据了，不过这仅仅是在该Controller中生效，若想在全局范围内生效的话，可以将@InitBinder注解所在的Controller定义为一个BaseController，其余Controller都继承这个Controller。当然还有另外的方法，若你有兴趣的话，请看2。

**2、自定义WebBindingInitializer**

[复制代码](javascript:void(0);)

public class MyWebBindingInitializer implements WebBindingInitializer {

@Override

public void initBinder(WebDataBinder binder, WebRequest request) {

binder.registerCustomEditor(Date.class, new DateEditor());

}

}

[复制代码](javascript:void(0);)

还是前面写的DateEditor，这么快又见面了，只不过注册的位置改变了，在WebBindingInitializer中注册的PropertyEditor是在全局范围内共享的。

不过光这样还不够，还要将WebBindingInitializer注入到AnnotationMethodHandlerAdapter中。

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">

<property name="webBindingInitializer">

<bean class="com.\*\*.\*\*.\*\*.MyWebBindingInitializer" />

</property>

</bean>

完美解决问题提

测试代码

[复制代码](javascript:void(0);)

@Controller

@RequestMapping(value="/")

public class BaseController {

@RequestMapping(value="index")

public String indexJsp(String username,Date date,Date ddate){

System.out.println(username);

System.out.println(date);

System.out.println(ddate);

return "index";

}

}

[复制代码](javascript:void(0);)

测试参数

index?username=fasdf&date=2012-12-03&ddate=2013-03-06

输出结果：

fasdf

Mon Dec 03 00:00:00 CST 2012

Wed Mar 06 00:00:00 CST 2013

参考文档：[http://blog.csdn.net/u012345283/article/details/43268081](http://blog.csdn.net/u012345283/article/details/43268081" \t "_blank)

第二种方式使用Converter

代码如下

[复制代码](javascript:void(0);)

public class MyConverter implements Converter<String, Date> {

@Override

public Date convert(String source) {

// TODO Auto-generated method stub

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(

"yyyy-MM-dd HH:mm:ss");

try {

// 转成直接返回

return simpleDateFormat.parse(source);

} catch (ParseException e) {

simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");

try {

return simpleDateFormat.parse(source);

} catch (ParseException e1) {

System.out.println("日期转换失败->" + this.getClass().getName());

}

}

// 如果参数绑定失败返回null

return null;

}

}

[复制代码](javascript:void(0);)

XML配置如下

[复制代码](javascript:void(0);)

<!-- 日期时间数据绑定 需要放在mvc:annotation-driven 之前 防止未扫描注入容器总- -->

<bean id="conversionService"

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

<property name="converters">

<list>

<bean class="\*\*\*\*.MyConverter" />

</list>

</property>

</bean>

[复制代码](javascript:void(0);)

 很轻松的解决问题，

3，当然Spring是非常强大的还可以使用Formatter，Formatter比Converter更适合Converter是处理任意两类型间的转换，而Formatter是处理字符串和另一类型之间的转换的。

[复制代码](javascript:void(0);)

public class MyDateFormatter implements Formatter<Date> {

@Override

public String print(Date object, Locale locale) {

// TODO Auto-generated method stub

return null;

}

@Override

public Date parse(String text, Locale locale) throws ParseException {

// TODO Auto-generated method stub

SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

Date date = null;

try {

date = format.parse(text);

} catch (Exception e) {

format = new SimpleDateFormat("yyyy-MM-dd");

date = format.parse(text);

}

return date;

}

}

[复制代码](javascript:void(0);)

XML配置如下：

[复制代码](javascript:void(0);)

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

<bean id="conversionService"

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

<property name="formatters">

<set>

<bean class="com.ivlo.crm.extend.MyDateFormatter"></bean>

</set>

</property>

</bean>

[复制代码](javascript:void(0);)

不问先后顺序