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：

<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:

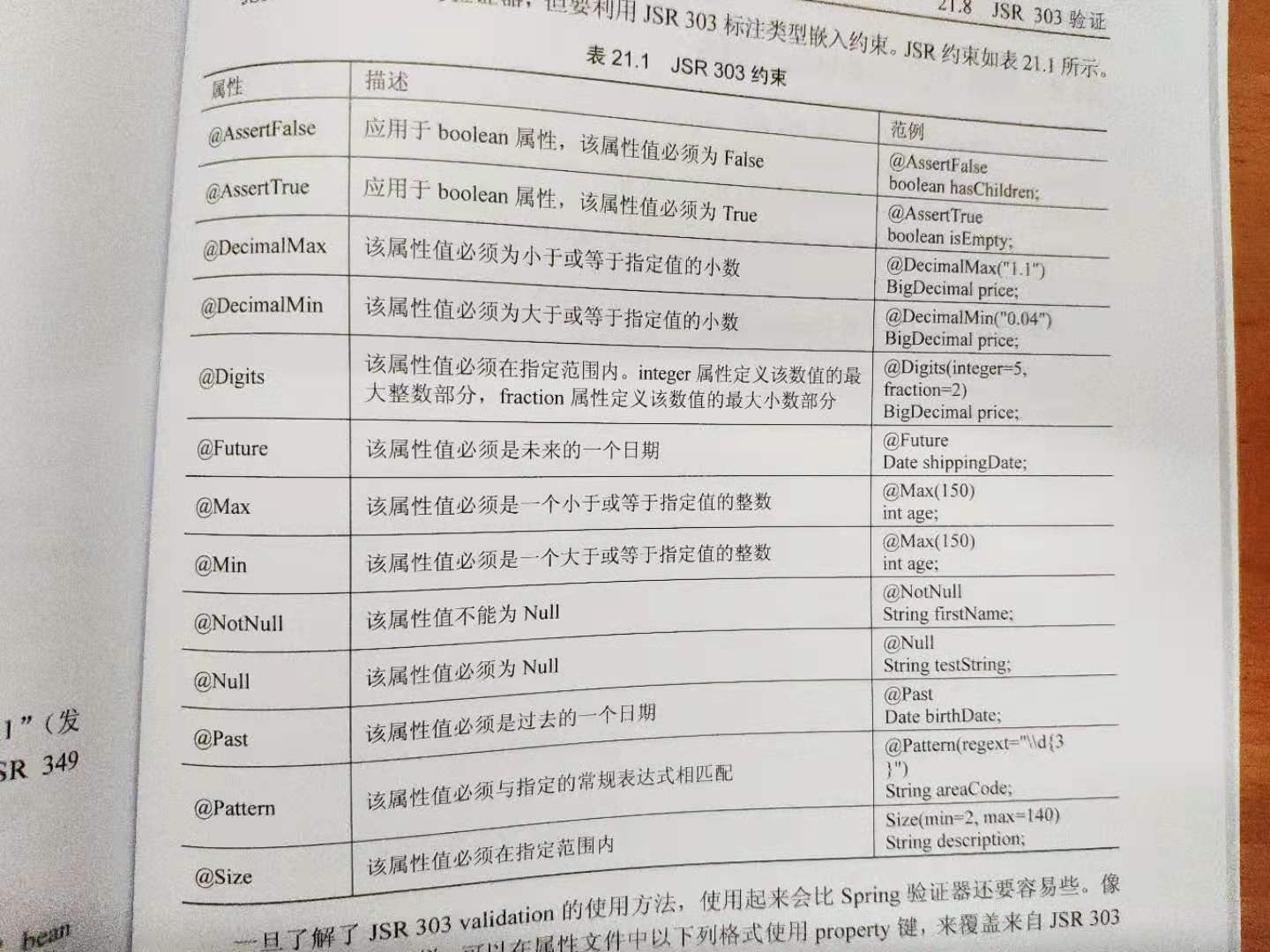
@RequestMapping("date\_test")  
@ResponseBody  
public String datetest(Product product, BindingResult bindingResult) { //为了验证有没有产生错误信息，必须绑定bingdingResult  
 ProductValidator productValidator = new ProductValidator();  
 productValidator.validate(product, bindingResult); //bingdingResult, 也就是Errors对象的一个实例  
  
 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";  
}