## 7. Validation, Data Binding, and Type Conversion

### 7.1 Introduction

### 7.2 Validation using Spring’s Validator interface

Spring提供Validation接口，进行验证Object，在Errors对象上，把验证报告显示错误信息

例子

**public** **class** Person {

**private** String name;

**private** **int** age;

*// the usual getters and setters...*

}

我们将提供验证行为，靠org.springframework.validation.Validatorin接口

* Supports（Class）
* validate(Object, org.springframework.validation.Errors)

当你知晓到ValidationUtils 帮助类后，你能够很直接实现一个Validator

**public** **class** PersonValidator **implements** Validator {

**/\*\***

**\* This Validator validates \*just\* Person instances**

**\*/**

**public** **boolean** supports(Class clazz) {

**return** Person.**class**.equals(clazz);

}

**public** **void** validate(Object obj, Errors e) {

ValidationUtils.rejectIfEmpty(e, "name", "name.empty");

Person p = (Person) obj;

**if** (p.getAge() < 0) {

e.rejectValue("age", "negativevalue");

} **else** **if** (p.getAge() > 110) {

e.rejectValue("age", "too.darn.old");

}

}

}

### 7.5 Spring Type Conversion

#### 7.5.1 Converter SPI

具强大的转换工具，提供的api

**package** org.springframework.core.convert.converter;

**public** **interface** Converter<S, T> {

T convert(S source);

}

自定义的转换工具

**package** org.springframework.core.convert.support;

**final** **class** StringToInteger **implements** Converter<String, Integer> {

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

**return** Integer.valueOf(source);

}

}

#### 7.5.2 ConverterFactory

当你抽象出来，转换的逻辑，例如你想把String转换成Enum

**package** org.springframework.core.convert.converter;

**public** **interface** ConverterFactory<S, R> {

<T **extends** R> Converter<S, T> getConverter(Class<T> targetType);

}

考虑下StringToEnumConverterFactory

**package** org.springframework.core.convert.support;

**final** **class** StringToEnumConverterFactory **implements** ConverterFactory<String, Enum> {

**public** <T **extends** Enum> Converter<String, T> getConverter(Class<T> targetType) {

**return** **new** StringToEnumConverter(targetType);

}

**private** **final** **class** StringToEnumConverter<T **extends** Enum> **implements** Converter<String, T> {

**private** Class<T> enumType;

**public** StringToEnumConverter(Class<T> enumType) {

**this**.enumType = enumType;

}

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

**return** (T) Enum.valueOf(**this**.enumType, source.trim());

}

}

}

#### 7.5.3 GenericConverter

当你想要精密的转换工具，可以考虑下GenericConverter接口

**package** org.springframework.core.convert.converter;

**public** **interface** GenericConverter {

**public** Set<ConvertiblePair> getConvertibleTypes();

Object convert(Object source, TypeDescriptor sourceType, TypeDescriptor targetType);

}

##### ConditionalGenericConverter

当你需要考虑到转换时，需要指定条件下，才进行转换

**public** **interface** ConditionalGenericConverter **extends** GenericConverter {

**boolean** matches(TypeDescriptor sourceType, TypeDescriptor targetType);

}

### 7.5.4 ConversionService API

ConversionService提供了统一标准的API, Converters通过外观模式进行转换

**package** org.springframework.core.convert;

**public** **interface** ConversionService {

**boolean** canConvert(Class<?> sourceType, Class<?> targetType);

<T> T convert(Object source, Class<T> targetType);

**boolean** canConvert(TypeDescriptor sourceType, TypeDescriptor targetType);

Object convert(Object source, TypeDescriptor sourceType, TypeDescriptor targetType);

}

较多ConversionService同样也实现了ConverterRegistry接口，用来注册转换器

### 7.5.5 Configuring a ConversionService

<bean id="conversionService"

class="org.springframework.context.support.ConversionServiceFactoryBean">

<property name="converters">

<set>

<bean class="example.MyCustomConverter"/>

</set>

</property>

</bean>

### 7.5.6 Using a ConversionService programmatically

*@Service*

**public** **class** MyService {

*@Autowired*

**public** MyService(ConversionService conversionService) {

**this**.conversionService = conversionService;

}

**public** **void** doIt() {

**this**.conversionService.convert(...)

}

}

## 7.7 Configuring a global date & time format

默认情况下，date和time类型没有注解，你可以通过org.springframework.format.datetime.joda.JodaTimeFormatterRegistrar or org.springframework.format.datetime.DateFormatterRegistrar

例如，你可以全局的时间格式“yyyyMMdd”

*@Configuration*

**public** **class** AppConfig {

*@Bean*

**public** FormattingConversionService conversionService() {

*// Use the DefaultFormattingConversionService but do not register defaults*

DefaultFormattingConversionService conversionService = **new** DefaultFormattingConversionService(false);

*// Ensure @NumberFormat is still supported*

conversionService.addFormatterForFieldAnnotation(**new** NumberFormatAnnotationFormatterFactory());

*// Register date conversion with a specific global format*

DateFormatterRegistrar registrar = **new** DateFormatterRegistrar();

registrar.setFormatter(**new** DateFormatter("yyyyMMdd"));

registrar.registerFormatters(conversionService);

**return** conversionService;

}

}

如果你更愿意通过XML方式，可以使用FormattingConversionServiceFactoryBean

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd>

<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">

<property name="registerDefaultFormatters" value="false" />

<property name="formatters">

<set>

<bean class="org.springframework.format.number.NumberFormatAnnotationFormatterFactory" />

</set>

</property>

<property name="formatterRegistrars">

<set>

<bean class="org.springframework.format.datetime.joda.JodaTimeFormatterRegistrar">

<property name="dateFormatter">

<bean class="org.springframework.format.datetime.joda.DateTimeFormatterFactoryBean">

<property name="pattern" value="yyyyMMdd"/>

</bean>

</property>

</bean>

</set>

</property>

</bean>

</beans>