22 Spring Core参数校验之validator

更新时间：2020-08-04 18:05:10

> 学习这件事不在乎有没有人教你，最重要的是在于你自己有没有觉悟和恒心。 —— 法布尔

背景

参数校验是我们程序开发中必不可少的过程。用户在前端要校验参数的合法性，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。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？

参数校验常用方式：

1. Validator 接口校验：Validator 接口校验是 Spring 的校验机制。可以用它来验证自己定义的实体对象。注意：这个接口是上下文无关的，也就是说不仅可以在 Web 层验证对象，也可以在 dao 层，或者其它任意层。

2. Bean校验：Bean Validation 标准化了 Java 平台的约束定义、描述、和验证。它现在一个有两个规范：Bean Validation 1.0（JSR-303）和 Bean Validation 1.1（JSR-349）和 Bean Validation 2.0（JSR-380）。hibernate validator 提供了一套比较完善、便捷的验证实现方式。

不积跬步无以至千里，不积小流无以成江海。本次我们以实现 Validator 接口为例进行验证，并了解其背后的原理。

Validator接口校验实例

要验证的 Java Bean：

```java
import java.math.BigDecimal;
import java.time.LocalDate;

import lombok.Data;
@Data
public class Order {
    private LocalDate date;
    private BigDecimal price;

    @Override
    public String toString () {
        return "Order{'date='" + date + "', price=" + price + '}';
    }
}
```

验证类实现了 Validator 接口，重写了supports 方法和 validate 方法：

supports 方法的作用： 判断当前的 Validator 实现类是否支持校验需要校验的实体类，该方法返回 true 时才可以调用 valida 方法来对需要校验的实体类进行校验。

validate 方法的作用： 编写具体的校验逻辑，并根据不同的校验结果，将错误放入错误对象 Errors 中。其中 Errors 是存储和暴露数据绑定错误和验证错误相关信息的接口，其提供了存储和获取错误消息的方法。

```java
import java.math.BigDecimal;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

public class OrderValidator implements Validator {

    @Override
    public boolean supports (Class<?> clazz) {
        return Order.class == clazz;
    }

    @Override
    public void validate (Object target, Errors errors) {
        ValidationUtils.rejectIfEmpty(errors, "date", "date.empty");
        ValidationUtils.rejectIfEmpty(errors, "price", "price.empty");
        Order order = (Order) target;
        if (order.getPrice() != null &&
                        order.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
            errors.rejectValue("price", "price.invalid");
        }
    }
}
```

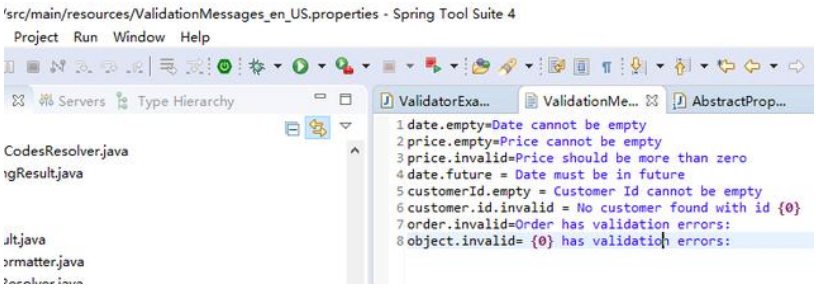
测试类：

```java
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.validation.BeanPropertyBindingResult;
import org.springframework.validation.ValidationUtils;
import java.math.BigDecimal;
import java.util.Locale;

public class ValidatorExample {

    public static void main (String[] args) {

        ResourceBundleMessageSource messageSource =
                        new ResourceBundleMessageSource();
        messageSource.setBasename("ValidationMessages");

        Order order = new Order();
        order.setPrice(BigDecimal.ZERO);
        BeanPropertyBindingResult result = new BeanPropertyBindingResult(
                        order, "order");
        ValidationUtils.invokeValidator(new OrderValidator(), order, result);
        result.getAllErrors().stream().
                        forEach(e -> System.out.println(messageSource
                                        .getMessage(e, Locale.US)));
    }
}
```

说明：

第一步： 验证消息使用 ResourceBundleMessageSource 来指定相应的位置和名称。

验证消息如下：



第二步： 定义了 Error 消息的存放对象。

BeanPropertyBindingResult 实现了 Error和BindingResult 接口，用来存放验证错误的信息。

第三步： 使用 ValidationUtils 触发对 Bean 的验证，并将验证错误信息结果放入 BeanPropertyBindingResult 中。

第四步： 从 BeanPropertyBindingResult 中获取验证错误的信息，并打印出来。

打印结果：

> Date cannot be empty
>
> Price should be more than zero

在验证类 OrderValidator 的 validate 方法中，规定了 Date 和 prize 不能为空，且价格要大于 0。

在代码中我们定义 order 时，仅仅定义价格为 0，因此会打印上面的结果。

工作原理

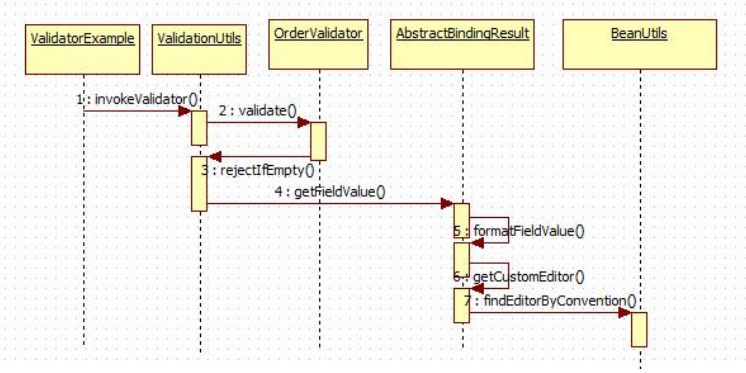
通过 debug 源代码，我们可以了解到流程的主要路径是：

第一步： ValidationUtils.invokeValidator() 方法来触发对order bean的验证，使用实现了 validator 接口的自定义实现类 OrderValidator。

第二步： OrderValidator 的具体逻辑在 validate() 方法，对指定的 bean 进行验证，并将错误放入错误对象 Errors 中。

第三步： 底层调用 PropertyEditor 或者 ConversionService 来获取 Bean 的属性值。

Validator 接口全流程图如下：



重点：

AbstractPropertyBindingResult：

```
/**
 * Formats the field value based on registered PropertyEditors.
 * @see #getCustomEditor
 */
@Override
protected Object formatFieldValue(String field, @Nullable Object value) {
    String fixedField = fixedField(field);
    // Try custom editor...
    PropertyEditor customEditor = getCustomEditor(fixedField);
    if (customEditor != null) {
        customEditor.setValue(value);
        String textValue = customEditor.getAsText();
        // If the PropertyEditor returned null, there is no appropriate
        // text representation for this value: only use it if non-null.
        if (textValue != null) {
            return textValue;
        }
    }
    if (this.conversionService != null) {
        // Try custom converter...
        TypeDescriptor fieldDesc = getPropertyAccessor().getPropertyTypeDescriptor(fixedField);
        TypeDescriptor strDesc = TypeDescriptor.valueOf(String.class);
        if (fieldDesc != null && this.conversionService.canConvert(fieldDesc, strDesc)) {
            return this.conversionService.convert(value, fieldDesc, strDesc);
        }
    }
    return value;
}
```

PropertyEditor

converter

总结

参数校验是我们程序开发中必不可少的过程，它的好坏决定了程序的健壮性，故我们必须打起来十二万的精神加以重视，Spring 提供了 org.springframework.validation.Validator 的方式，同时也支持 JSR 349/303 注解方式，因篇章限制，本次讨论 Validator 接口方式，通过 debug 代码我们可以看到其内部是通过 PropertyEditor 或者 ConversionService 来验证属性的。

}

21 Resource总结

23 Spring Core参数校验之JSR-303_JSR-349注解