25 Spring Core 数据绑定之BeanWrapper实现示例及背后原理探究

更新时间：2020-08-04 18:27:28

> 理想必须要人们去实现它，它不但需要决心和勇敢而且需要知识。——吴玉章

背景

在 Spring 配置文件中使用 <value.../> 元素来为属性指定属性值，如下面的例子所示：

```
<bean id="pojoBean" class="com.davidwang456.test.POJOBean">
    <property name="name" value="davidwang456"></property>
    <property name="age" value="10"></property>
    <property name="name" value="2019-12-24"></property>
</bean>
```

上面的 XML 是如何实现的呢？

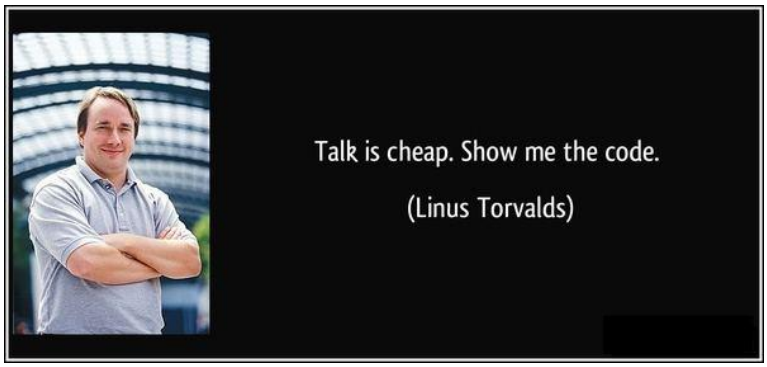
Spring MVC 数据绑定示例

为了探究上面的原理，我们暂且忘掉上面 XML 配置方式，如何使用编程方式来做到对属性指定相应的值呢？

有两种方式可以做到：

（1）使用 BeanWrapper 实现属性指定；

（2）使用 DataBinder 来实现属性指定（下篇）。



本章节我们探索一下 BeanWrapper 实现属性指定的过程：

示例程序，Java Bean 如下：

```
package com.davidwang456.test;

import java.util.Date;

import lombok.Data;
@Data
public class POJOBean {
    private int age;
    private String name;
    private Date birthday;
    @Override
    public String toString () {
        return "POJOBean{name=" + name+",age="+age+",birthday="+birthday.toGMTString()+ '}';
    }
}
```

测试类：

```
package com.davidwang456.test;

import java.util.Date;

import org.springframework.beans.BeanWrapper;
import org.springframework.beans.BeanWrapperImpl;
import org.springframework.beans.MutablePropertyValues;

public class BeanWrapperWithError {
    public static void main (String[] args) {
        MutablePropertyValues mpv = new MutablePropertyValues();
        mpv.add("name", "davidwang456");
        mpv.add("age", "10");
        mpv.add("birthday", new Date());

        BeanWrapper bw = new BeanWrapperImpl(new POJOBean());
        bw.setPropertyValues(mpv);
        System.out.println(bw.getWrappedInstance());
    }
}
```

MutablePropertyValues是PropertyValues的默认实现，可以对属性进行简单的操作，同时也提供了支持深度复制和从Map中复制的构造方法。

BeanWrapperImpl是BeanWrapper接口的默认实现，BeanWrapper是spring 底层JavaBean最基础的工具，它提供了分析和操作标准JavaBean的各种操作。

现在我们可以开始 debug 了。

Spring-Core数据绑定原理

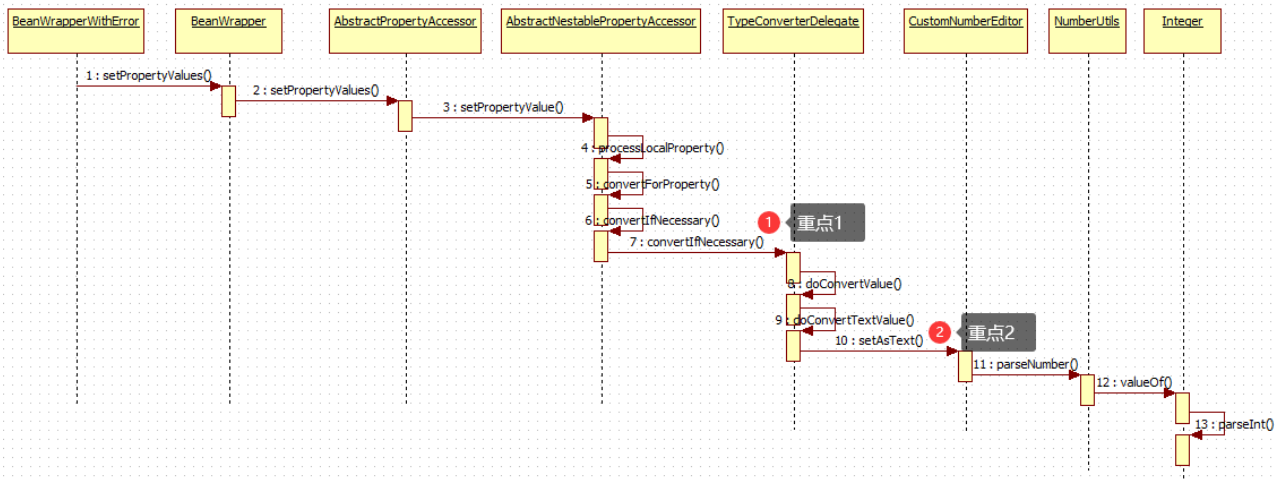
通过一层层 debug，我们发现 BeanWrapper 的底层是通过 PropertyEditor 的 setAsText 方法来完成数据绑定的。如下图所示：

CustomNumberEditor.java#setAsText()：

```
/**
 * Parse the Number from the given text, using the specified NumberFormat.
 */
@Override
public void setAsText(String text) throws IllegalArgumentException {
    if (this.allowEmpty && !StringUtils.hasText(text)) {
        // Treat empty String as null value.
        setValue(null);
    }
    else if (this.numberFormat != null) {
        // Use given NumberFormat for parsing text.
        setValue(NumberUtils.parseNumber(text, this.numberClass, this.numberFormat));
    }
    else {
        // Use default valueOf methods for parsing text.
        setValue(NumberUtils.parseNumber(text, this.numberClass));
    }
}
```

1 重点

其完整调用链如下图所示：



从上图的可以看出：重中之重在 **TypeConverterDelegate.java#convertIfNecessary**，它来决定是使用那种 PropertyEditor 来完整最终的赋值。代码如下：

```
/**
 * Convert the value to the required type (if necessary from a String),
 * for the specified property.
 * @param propertyName name of the property
 * @param oldValue the previous value, if available (may be {@code null})
 * @param newValue the proposed new value
 * @param requiredType the type we must convert to
 * (or {@code null} if not known, for example in case of a collection element)
 * @param typeDescriptor the descriptor for the target property or field
 * @return the new value, possibly the result of type conversion
 * @throws IllegalArgumentException if type conversion failed
 */
@SuppressWarnings("unchecked")
@Nullable
public <T> T convertIfNecessary(@Nullable String propertyName, @Nullable Object oldValue, @Nullable Object newValue,
        @Nullable Class<T> requiredType, @Nullable TypeDescriptor typeDescriptor) throws IllegalArgumentException {

    // Custom editor for this type?
    PropertyEditor editor = this.propertyEditorRegistry.findCustomEditor(requiredType, propertyName);

    ConversionFailedException conversionAttemptEx = null;

    // No custom editor but custom ConversionService specified?
    ConversionService conversionService = this.propertyEditorRegistry.getConversionService();
    if (editor == null && conversionService != null && newValue != null && typeDescriptor != null) {
        TypeDescriptor sourceTypeDesc = TypeDescriptor.forObject(newValue);
        if (conversionService.canConvert(sourceTypeDesc, typeDescriptor)) {
            try {
                return (T) conversionService.convert(newValue, sourceTypeDesc, typeDescriptor);
            }
            catch (ConversionFailedException ex) {
                // fallback to default conversion logic below
                conversionAttemptEx = ex;
            }
        }
    }

    Object convertedValue = newValue;

    // Value not of required type?
    if (editor != null || (requiredType != null && !ClassUtils.isAssignableValue(requiredType, convertedValue))) {
        if (typeDescriptor != null && requiredType != null && Collection.class.isAssignableFrom(requiredType) &&
                convertedValue instanceof String) {
            TypeDescriptor elementTypeDesc = typeDescriptor.getElementTypeDescriptor();
            if (elementTypeDesc != null) {
                Class<?> elementType = elementTypeDesc.getType();
                if (Class.class == elementType || Enum.class.isAssignableFrom(elementType)) {
                    convertedValue = StringUtils.commaDelimitedListToStringArray((String) convertedValue);
                }
            }
        }
        if (editor == null) {
            editor = findDefaultEditor(requiredType);
        }
        convertedValue = doConvertValue(oldValue, convertedValue, requiredType, editor);
    }
```

如果指定 PropertyEditor，则使用指定的 PropertyEditor，否则使用默认的 PropertyEditor。

总结

通过调试代码示例，我们可以看到，指定属性最终是要使用实现了 PropertyEditorSupport 的各种 PropertyEditor 来实现类型的转换，TypeConverterDelegate 使用了代理模式，代理了各种 PropertyEditor 的实现类。PropertyEditor 的 setAsText 方法完成了转换逻辑，各种 PropertyEditor 实现可以参照下图所示。

}

24 Spring Core vs Spring MVC参数校验

26 Spring Core 数据绑定之DataBinder实现示例及背后原理探究