29 Spring MVC之类型转换Converter

更新时间：2020-08-06 13:41:07

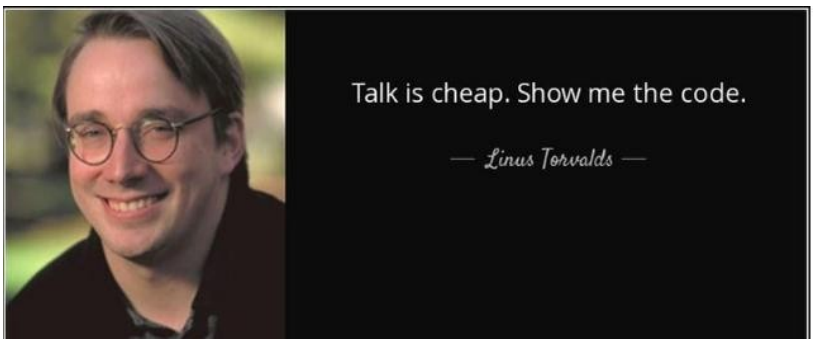
> 加紧学习，抓住中心，宁精勿杂，宁专勿多。 —— 周恩来

背景

在以往我们需要 SpringMVC 为我们自动进行类型转换的时候都是用的 PropertyEditor。通过 PropertyEditor 的 setAsText() 方法我们可以实现字符串向特定类型的转换。但是这里有一个限制是它只支持从 String 类型转为其他类型。在 Spring3 中引入了一个 Converter 接口，它支持从一个 Object 转为另一个 Object。除了 Converter 接口之外，实现 ConverterFactory 接口和 GenericConverter 接口也可以实现我们自己的类型转换逻辑。那么在 spring mvc 中如何实现自定义的 Converter 来实现表单数据转换到特定的类型呢？

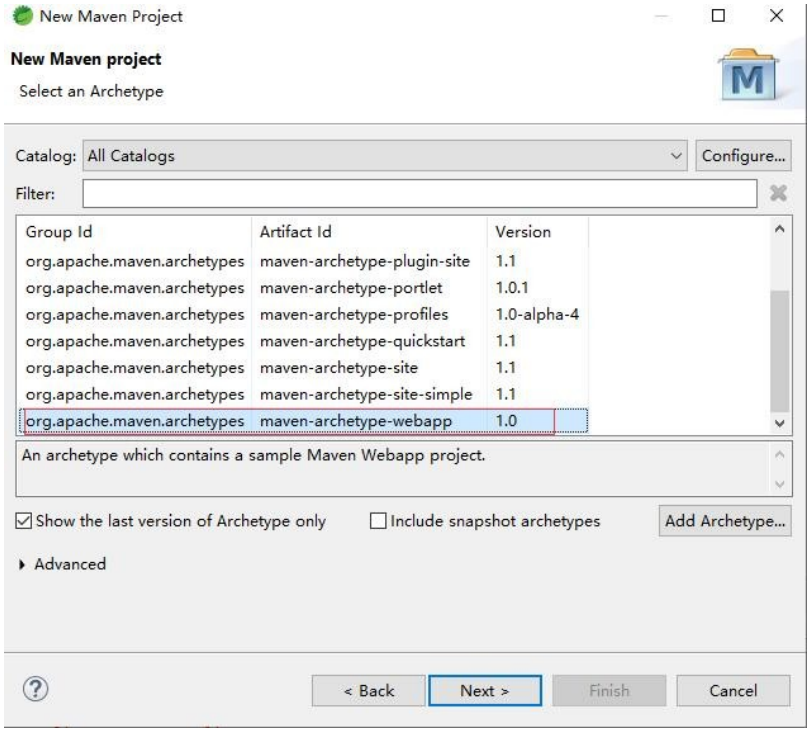
SpringMVC 实现自定义 Converter 类型转换器实例

Talk is cheap. Show me the code.



准备一个普通的 MVC 项目

1. 创建 **Maven** 项目，选择 **Web** 项目：



2. 添加依赖：

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <!-- Spring dependencies -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>

            <!-- Servlet Dependency -->
    <dependency>
         <groupId>javax.servlet</groupId>
         <artifactId>javax.servlet-api</artifactId>
         <version>3.1.0</version>
         <scope>provided</scope>
    </dependency>
</dependencies>
```

3. 修改或者添加配置文件：

web.xml 替换类：

```java
/**
 * Abstract base for exceptions related to media types. Adds a list of supported
 * {@link MediaType MediaTypes}.
 *
 * @author Arjen Poutsma
 * @since 3.0
 */
public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class<?>[] { AppConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

dispatcher-servlet.xml 替换类：

```java
package org.framework.davidwang456.controller;

import org.springframework.context.MessageSource;

/**
 * Abstract base for exceptions related to media types. Adds a list of supported {@link MediaType MediaTypes}.
 *
 * @author Arjen Poutsma
 * @since 3.0
 */
@SuppressWarnings("deprecation")
@Configuration
@EnableWebMvc
@ComponentScan
public class AppConfig extends WebMvcConfigurerAdapter {

    @Bean
    public MessageSource messageSource () {
        ResourceBundleMessageSource messageSource =
                new ResourceBundleMessageSource();
        messageSource.setBasenames("locale/messages");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
    }

    @Override
    public void addFormatters (FormatterRegistry registry) {
        registry.addConverter(new StringToDateTimeConverter());
    }

    @Bean
    public ViewResolver beanNameViewResolver() {
        BeanNameViewResolver resolver = new BeanNameViewResolver();
        return resolver;
    }
}
```

4. 控制器 **Controller**：

自定义 converter:StringToDateTimeConverter：

```java
package org.framework.davidwang456.controller;

import java.text.SimpleDateFormat;
import java.util.Date;

import org.springframework.core.convert.converter.Converter;
import org.springframework.expression.ParseException;

public class StringToDateTimeConverter implements Converter<String, Date>{
    private final static String datePattern="MM-dd-yyyy";
    @Override
    public Date convert(String source)  {
        try {
            SimpleDateFormat dateFormat = new SimpleDateFormat(datePattern);
            dateFormat.setLenient(false);
            return dateFormat.parse(source);
        } catch (ParseException | java.text.ParseException e) {
            throw new IllegalArgumentException("invalid date format. Please use this pattern\"" + datePattern + "\
        }
    }

}
```

配置自定义 Converter：

```
package org.framework.davidwang456.controller;

import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.format.FormatterRegistry;
import org.springframework.http.MediaType;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

/**
 * Abstract base for exceptions related to media types. Adds a list of supported {@link MediaType MediaTypes}.
 *
 * @author Arjen Poutsma
 * @since 3.0
 */
@SuppressWarnings("deprecation")
@Configuration
@EnableWebMvc
@ComponentScan
public class AppConfig extends WebMvcConfigurerAdapter {

      @Bean
      public MessageSource messageSource () {
          ResourceBundleMessageSource messageSource =
                    new ResourceBundleMessageSource();
          messageSource.setBasenames("locale/messages");
          messageSource.setDefaultEncoding("UTF-8");
          return messageSource;
      }

       @Override
       public void addFormatters (FormatterRegistry registry) {
           registry.addConverter(new StringToDateTimeConverter());
       }
}
```

测试类：

```
@RequestMapping(value="/test3")
@ResponseBody
public String handlePostRequest (@Valid @ModelAttribute("user") User user, BindingResult bindingResult,
                                 Model model) {


    if (bindingResult.hasErrors()) {
        populateError("name", model, bindingResult);
        populateError("email", model, bindingResult);
        populateError("password", model, bindingResult);
        populateError("dateOfBirth", model, bindingResult);
        throw new NullPointerException();
    }
    return "registration-done";
}
```

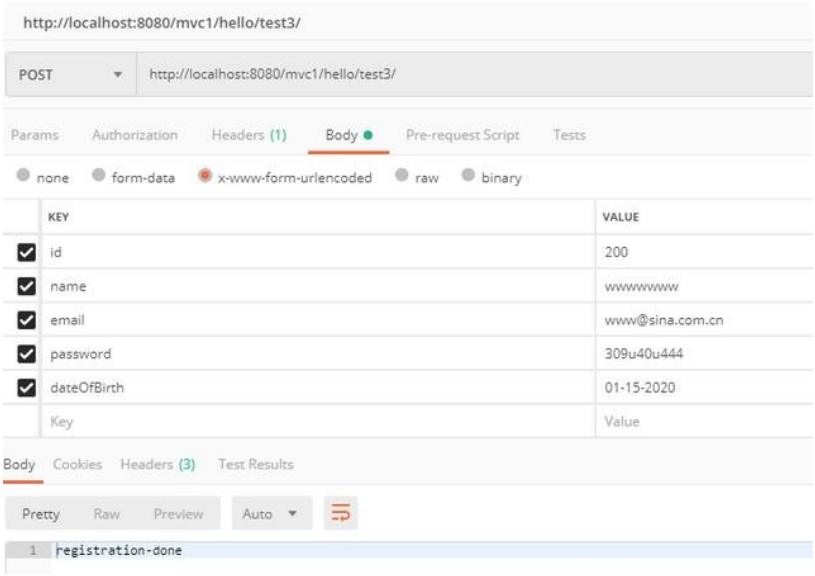
注意，没有使用 InitBinder 注解。

Postman 测试

启动 Web 项目，可以添加 Jetty 或者 Tomcat 插件来启动，本文以 Jetty 为例，添加 Jetty 依赖：

```
<build>
    <plugins>
      <plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>9.4.8.v20171121</version>
      </plugin>
    </plugins>
  </build>
```

使用 jetty:run 启动，然后利用 Postman 进行测试。



深入自定义 **Converter** 实现原理

1. 初始化：

web 容器启动时，抛出异常，查看调用链：

异常如下：

```
Caused by: java.lang.NullPointerException
        at org.framework.davidwang456.controller.AppConfig.addFormatters(AppConfig.java:53)
        at org.springframework.web.servlet.config.annotation.WebMvcConfigurerComposite.addFormatters(WebMvcConfigurerComposite.java:81)
        at org.springframework.web.servlet.config.annotation.DelegatingWebMvcConfiguration.addFormatters(DelegatingWebMvcConfiguration.java:78)
        at org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport.mvcConversionService(WebMvcConfigurationSupport.java:697)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.springframework.beans.factory.support.SimpleInstantiationStrategy.instantiate(SimpleInstantiationStrategy.java:154)
        ... 39 more
```

2. 去掉异常，追踪源码，查看背后：

TypeConverterDelegate#convertIfNecessary：

```
@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);
                }
            }
        }
    }
```

当没有定义PropertyEditor，但是有自定义的conversionService时，执行converter的数据转换方法convert

GenericConversionService#convert 封装了 converter 的实现：

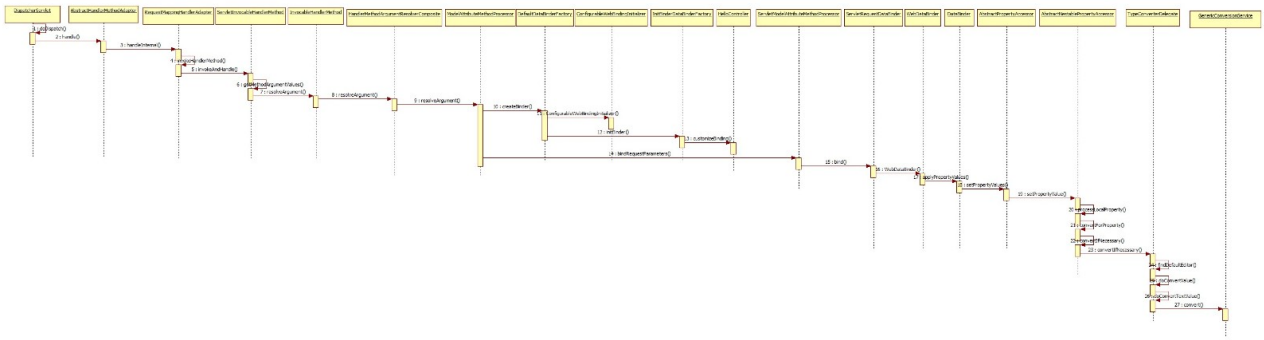
```
@Override
@Nullable
public Object convert(@Nullable Object source, @Nullable TypeDescriptor sourceType, TypeDescriptor targetType) {
    Assert.notNull(targetType, "Target type to convert to cannot be null");
    if (sourceType == null) {
        Assert.isTrue(source == null, "Source must be [null] if source type == [null]");
        return handleResult(null, targetType, convertNullSource(null, targetType));
    }
    if (source != null && !sourceType.getObjectType().isInstance(source)) {
        throw new IllegalArgumentException("Source to convert from must be an instance of [" +
                sourceType + "]; instead it was a [" + source.getClass().getName() + "]");
    }
    GenericConverter converter = getConverter(sourceType, targetType);
    if (converter != null) {
        Object result = ConversionUtils.invokeConverter(converter, source, sourceType, targetType);
        return handleResult(sourceType, targetType, result);
    }
    return handleConverterNotFound(source, sourceType, targetType);
}
```

1 根据源和目标类型，查找合适的转换器

2 触发最合适的转换器进行工作

3 处理转换结果

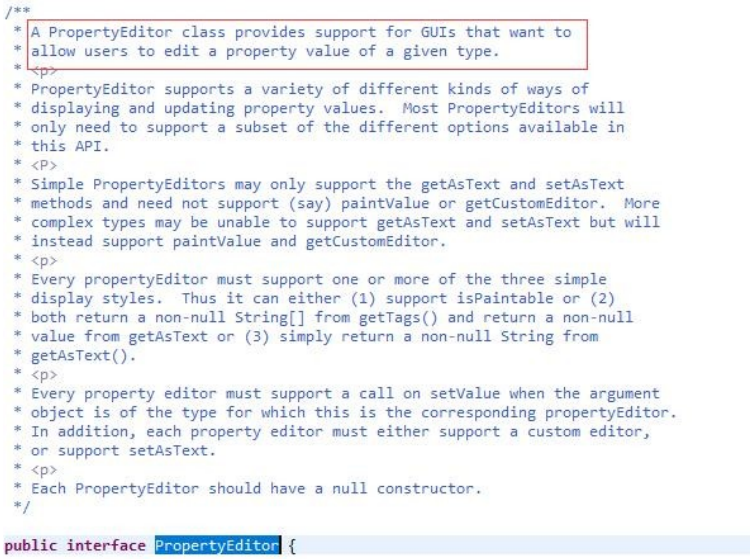
完整的流程如下图：



拓展 **PropertyEditor** 还是 **Converter**

1. PropertyEditor：

定义：



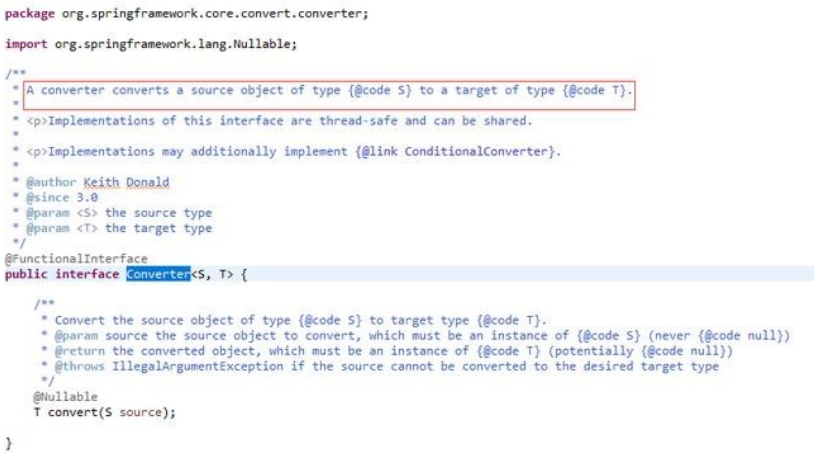
```
/**
 * A PropertyEditor class provides support for GUIs that want to
 * allow users to edit a property value of a given type.
 * <p>
 * PropertyEditor supports a variety of different kinds of ways of
 * displaying and updating property values.  Most PropertyEditors will
 * only need to support a subset of the different options available in
 * this API.
 * <P>
 * Simple PropertyEditors may only support the getAsText and setAsText
 * methods and need not support (say) paintValue or getCustomEditor.  More
 * complex types may be unable to support getAsText and setAsText but will
 * instead support paintValue and getCustomEditor.
 * <p>
 * Every propertyEditor must support one or more of the three simple
 * display styles.  Thus it can either (1) support isPaintable or (2)
 * both return a non-null String[] from getTags() and return a non-null
 * value from getAsText or (3) simply return a non-null String from
 * getAsText().
 * <p>
 * Every property editor must support a call on setValue when the argument
 * object is of the type for which this is the corresponding propertyEditor.
 * In addition, each property editor must either support a custom editor,
 * or support setAsText.
 * <p>
 * Each PropertyEditor should have a null constructor.
 */

public interface PropertyEditor {
```

适用场景 GUI 传递的参数转换，通常是 String–各种对象。

2. Converter：

定义：



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

import org.springframework.lang.Nullable;

/**
 * A converter converts a source object of type {@code S} to a target of type {@code T}.
 *
 * <p>Implementations of this interface are thread-safe and can be shared.
 *
 * <p>Implementations may additionally implement {@link ConditionalConverter}.
 *
 * @author Keith Donald
 * @since 3.0
 * @param <S> the source type
 * @param <T> the target type
 */
@FunctionalInterface
public interface Converter<S, T> {

    /**
     * Convert the source object of type {@code S} to target type {@code T}.
     * @param source the source object to convert, which must be an instance of {@code S} (never {@code null})
     * @return the converted object, which must be an instance of {@code T} (potentially {@code null})
     * @throws IllegalArgumentException if the source cannot be converted to the desired target type
     */
    @Nullable
    T convert(S source);

}
```

PropertyEditor 的扩展，它支持从一个 Object 转为另一个 Object。

总结

通过 PropertyEditor 的 setAsText()方法我们可以实现字符串向特定类型的转换。适用场景为 web 前后之间数据的传递。

Converter 接口，它支持从一个 Object 转为另一个 Object。除了 Converter 接口之外，实现 ConverterFactory 接口和 GenericConverter 接口也可以实现我们自己的类型转换逻辑。适用于业务逻辑内部。可以用 @Autowired 来引入 ConversionService

}