27 Spring MVC数据绑定InitBinder揭秘

更新时间：2020-08-04 18:53:37

> 人生的旅途，前途很远，也很暗。然而不要怕，不怕的人的面前才有路。—— 鲁 迅

背景

在使用 SpingMVC 框架的项目中，经常会遇到页面某些数据要转换成类型是 Date、Integer、Double 等的数据绑定到控制器的实体。

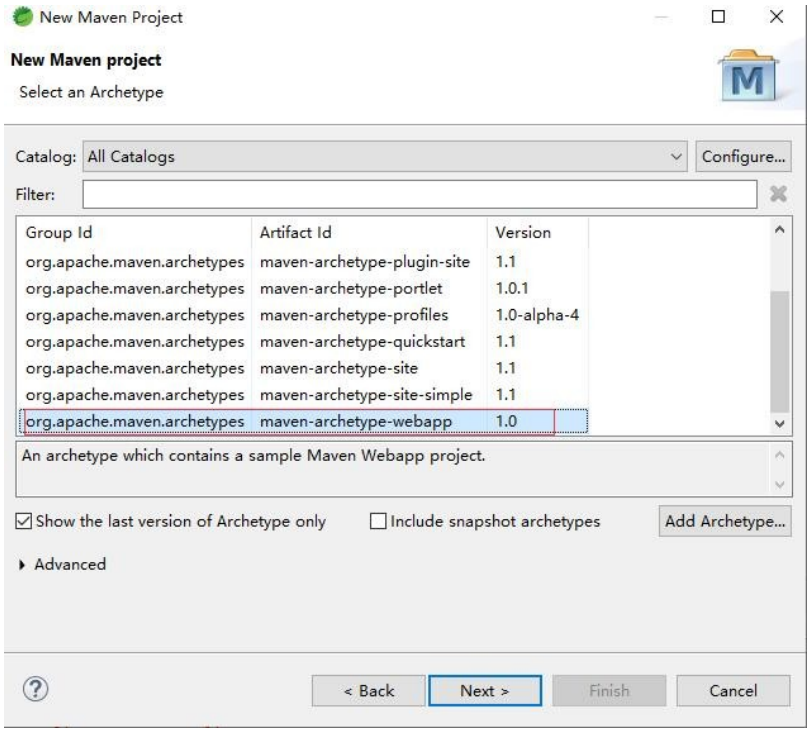
Spring 提供了很多的将表单数据转换到特定类型的 PropertyEditor 实现类，如 CustomDateEditor、CustomBooleanEditor、CustomNumberEditor 等将页面数据转换到特定的数据类型，在绑定表单之前，都会先注册这些编辑器，SpringMVC 提供了使用注解 `@InitBinder` 来解决注册这些类型转换器，然后通过例如 @ModelAttribute 类似的注解来绑定表单数据到 JavaBean。

一般会将这些方法些在 BaseController 中，需要进行这类转换的控制器只需继承 BaseController 即可。PropertyEditor 实现类很多，大部分场景是够用的。如果业务比较特殊，可以继承 PropertyEditorSupport 自定义一个类型转换器。

实例

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**：

```java
    @InitBinder("user")
    public void customizeBinding (WebDataBinder binder) {
        SimpleDateFormat dateFormatter = new SimpleDateFormat("yyyy-MM-dd");
        dateFormatter.setLenient(false);
        binder.registerCustomEditor(Date.class, "dateOfBirth",
                                    new CustomDateEditor(dateFormatter, true));
        //throw new NullPointerException();
    }

    @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";
    }

    private void populateError (String field, Model model, BindingResult bindingResult) {
        if (bindingResult.hasFieldErrors(field)) {
            model.addAttribute(field + "Error", bindingResult.getFieldError(field)
                                                             .getDefaultMessage());
        }
    }
```

其中 User 定义如下：

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

import java.util.Date;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

import lombok.Data;

@Data
public class User {
    private Long id;

    @Size(min = 5, max = 20)
    private String name;


    @Size(min = 6, max = 15)
    @Pattern(regexp = "\\S+", message = "Spaces are not allowed")
    private String password;

    @NotEmpty
    @Email
    private String email;

    @NotNull
    private Date dateOfBirth;
}
```

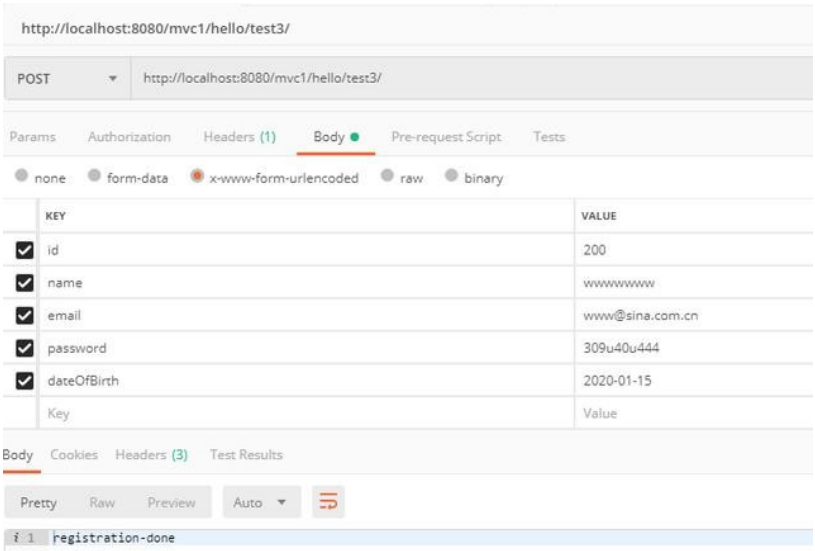
Postman 测试

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：

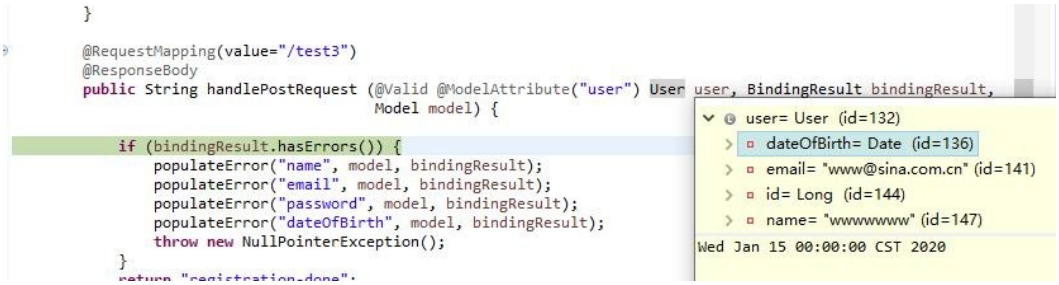
```xml
<build>
    <plugins>
      <plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>9.4.8.v20171121</version>
      </plugin>
    </plugins>
  </build>
```

使用 `jetty:run` 启动，然后利用 Postman 进行测试。

Header ： `Content-Type application/x-www-form-urlencoded` 。



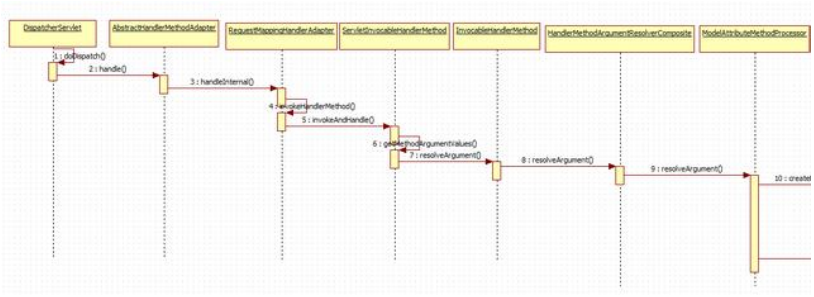
结果：



深入原理

从 DispatcherServlet 开始，所有的请求统一追踪到 InvocableHandlerMethod#invokeForRequest 获取参数。

![



此时，调用 HandlerMethodArgumentResolverComposite 包装的 resolveArgument 方法，真正执行的是:

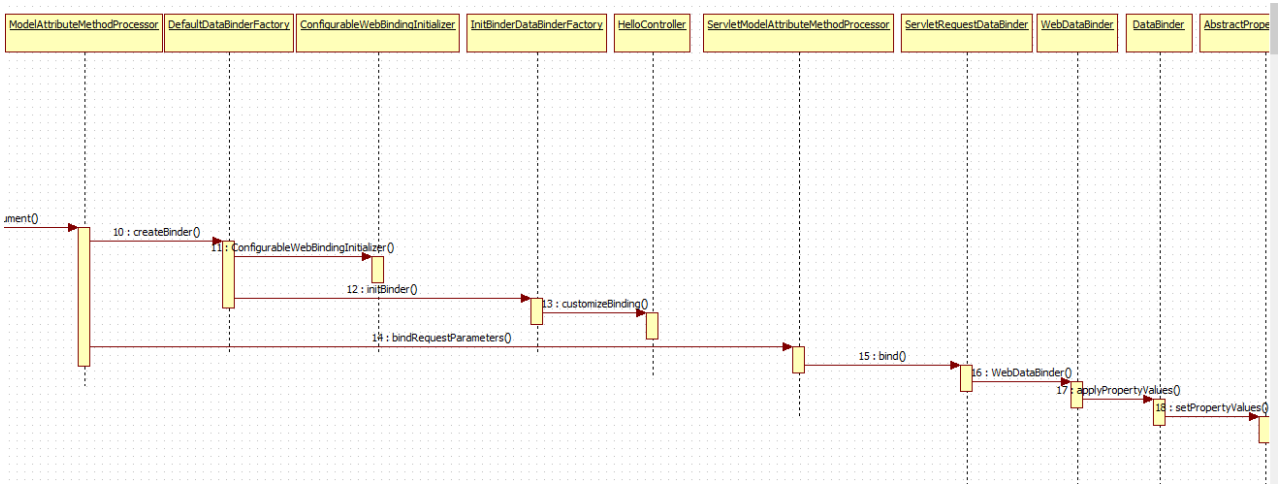
ModelAttributeMethodProcessor.java#resolveArgument 方法，Argument 解析注解 @ModelAttribute("user") 。

```
if (bindingResult == null) {
    // Bean property binding and validation;
    // skipped in case of binding failure on construction.
    WebDataBinder binder = binderFactory.createBinder(webRequest, attribute, name);
    if (binder.getTarget() != null) {
        if (!mavContainer.isBindingDisabled(name)) {
            bindRequestParameters(binder, webRequest);
        }
        validateIfApplicable(binder, parameter);
        if (binder.getBindingResult().hasErrors() && isBindExceptionRequired(binder, parameter)) {
            throw new BindException(binder.getBindingResult());
        }
    }
    // Value type adaptation, also covering java.util.Optional
    if (!parameter.getParameterType().isInstance(attribute)) {
        attribute = binder.convertIfNecessary(binder.getTarget(), parameter.getParameterType(), parameter);
    }
    bindingResult = binder.getBindingResult();
}

// Add resolved attribute and BindingResult at the end of the model
Map<String, Object> bindingResultModel = bindingResult.getModel();
mavContainer.removeAttributes(bindingResultModel);
mavContainer.addAllAttributes(bindingResultModel);

return attribute;
}
```


绑定的详细的流程如下：



1. 初始化 **HelloController** 自定义的 **PropertyEditor**

InitBinderDataBinderFactory.java#initBinder：

```
/**
 * Initialize a WebDataBinder with {@code @InitBinder} methods.
 * <p>If the {@code @InitBinder} annotation specifies attributes names,
 * it is invoked only if the names include the target object name.
 * @throws Exception if one of the invoked @{@link InitBinder} methods fails
 * @see #isBinderMethodApplicable
 */
@Override
public void initBinder(WebDataBinder dataBinder, NativeWebRequest request) throws Exception {
    for (InvocableHandlerMethod binderMethod : this.binderMethods) {
        if (isBinderMethodApplicable(binderMethod, dataBinder)) {
            Object returnValue = binderMethod.invokeForRequest(request, null, dataBinder);
            if (returnValue != null) {
                throw new IllegalStateException(
                        "@InitBinder methods must not return a value (should be void): " + binderMethod);
            }
        }
    }
}

/**
 * Determine whether the given {@code @InitBinder} method should be used
 * to initialize the given {@link WebDataBinder} instance. By default we
 * check the specified attribute names in the annotation value, if any.
 */
protected boolean isBinderMethodApplicable(HandlerMethod initBinderMethod, WebDataBinder dataBinder) {
    InitBinder ann = initBinderMethod.getMethodAnnotation(InitBinder.class);
    Assert.state(ann != null, "No InitBinder annotation");
    String[] names = ann.value();
    return (ObjectUtils.isEmpty(names) || ObjectUtils.containsElement(names, dataBinder.getObjectName()));
}
```

2. 属性绑定

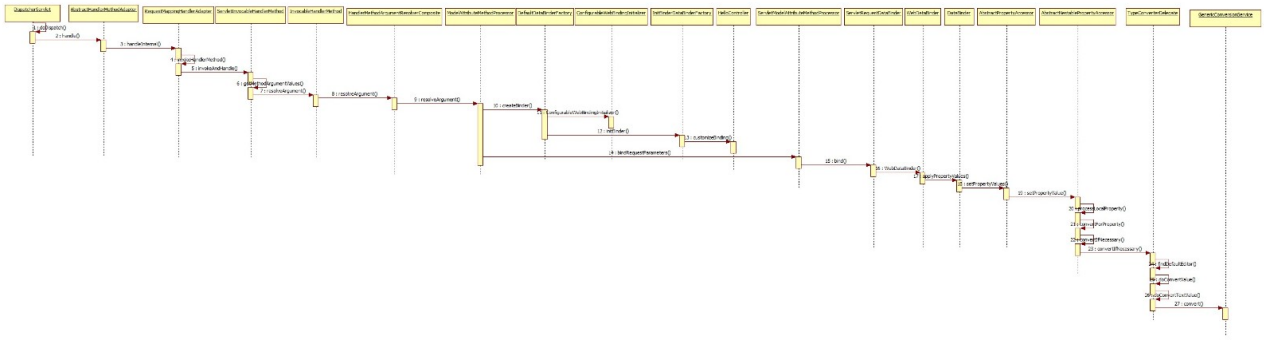
ServletRequestDataBinder#bind：

```
/**
 * Bind the parameters of the given request to this binder's target,
 * also binding multipart files in case of a multipart request.
 * <p>This call can create field errors, representing basic binding
 * errors like a required field (code "required"), or type mismatch
 * between value and bean property (code "typeMismatch").
 * <p>Multipart files are bound via their parameter name, just like normal
 * HTTP parameters: i.e. "uploadedFile" to an "uploadedFile" bean property,
 * invoking a "setUploadedFile" setter method.
 * <p>The type of the target property for a multipart file can be MultipartFile,
 * byte[], or String. The latter two receive the contents of the uploaded file;
 * all metadata like original file name, content type, etc are lost in those cases.
 * @param request the request with parameters to bind (can be multipart)
 * @see org.springframework.web.multipart.MultipartHttpServletRequest
 * @see org.springframework.web.multipart.MultipartFile
 * @see #bind(org.springframework.beans.PropertyValues)
 */
public void bind(ServletRequest request) {
    MutablePropertyValues mpvs = new ServletRequestParameterPropertyValues(request);
    MultipartRequest multipartRequest = WebUtils.getNativeRequest(request, MultipartRequest.class);
    if (multipartRequest != null) {
        bindMultipart(multipartRequest.getMultiFileMap(), mpvs);
    }
    addBindValues(mpvs, request);
    doBind(mpvs);
}
```

最终使用 DataBinder#applyPropertyValues 来执行：

```
/**
 * Apply given property values to the target object.
 * <p>Default implementation applies all of the supplied property
 * values as bean property values. By default, unknown fields will
 * be ignored.
 * @param mpvs the property values to be bound (can be modified)
 * @see #getTarget
 * @see #getPropertyAccessor
 * @see #isIgnoreUnknownFields
 * @see #getBindingErrorProcessor
 * @see BindingErrorProcessor#processPropertyAccessException
 */
protected void applyPropertyValues(MutablePropertyValues mpvs) {
    try {
        // Bind request parameters onto target object.
        getPropertyAccessor().setPropertyValues(mpvs, isIgnoreUnknownFields(), isIgnoreInvalidFields());
    }
    catch (PropertyBatchUpdateException ex) {
        // Use bind error processor to create FieldErrors.
        for (PropertyAccessException pae : ex.getPropertyAccessExceptions()) {
            getBindingErrorProcessor().processPropertyAccessException(pae, getInternalBindingResult());
        }
    }
}
```

完整流程如下：



扩展

上面的 initBinder 针对的是单个 Controller，那么如何设置全局的 initBinder 呢？

可以通过 `@ControllerAdvice` 注解来定义一个全局的 initBinder，使用 `@ControllerAdvice`，可以实现三个方面的功能：

1. 全局异常处理；
2. 全局数据绑定；
3. 全局数据预处理。

如下所示：

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

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

import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.InitBinder;

@ControllerAdvice
public class GlobalWebBinderAdvice {

    @InitBinder
    protected void initBinder(WebDataBinder binder) {
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    dateFormat.setLenient(false);
    // 注册
    binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, false));
    }
}
```

总结

从上面的流程可以看出，整个流程的关键点是 ModelAttributeMethodProcessor.java#resolveArgument ，它处理请求的数据绑定和数据验证。

1. 初始化，使用 WebDataBinderFactory 来创建 createBinder 一个 DataBinder;
2. 使用 DataBinder 将请求绑定到变量中;
3. 验证 `@Valid` 定义的变量。

}

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

28 百变星君Converter实现示例及背后原理探究