# 佟刚Spring MVC知识讲解：

## 一、SpringMVC概述：

### 1、Spring为展现层提供的基于MVC设计理念的优秀的web框架，是目前主流的MVC框架之一。

### 2、Spring3.0后，SpringMVC已经全面超越Struts2，成为最优秀的MVC框架。

### 3、SpringMVC通过一套MVC注解，让POJO成为处理请求的控制器，而无须实现任何接口。

### 4、采用了松散耦合可插拔组件结构，比其他MVC框架更具拓展性和灵活性

## 二、入门HelloWorld【参考springMVCDemo1\handler】

### 1、步骤：

1）加入jar包

2）在web.xml中配置DispatcherServlet

3）加入SpringMVC的配置文件

4）编写处理请求的处理器，并表示为处理器

5）编写视图

### 2、注意点：

1）从index.jsp的请求到了HelloWorld Controller 的method1()方法，然后再跳转到其他页面，注意此时不能跳转到WEB-INF目录及其子目录

2）实际上,也可以不通过 contextConfigLocation 来配置SpringMVC的配置文件,而使用默认的默认的配置文件为 : /WEB-INF/<servlet-name>-servlet.xml。比如我我们在web.xml文件中配置的DispatcherServlet的<servlet-name>为chenming，那么在/WEB-INF/下有一个默认的 chenming-servlet.xml的SpringMVC的核心配置文件

## 三、常用注解：

### 1、@RequestMapping：

1）SpringMVC使用 @RequestMapping 注解为控制器指定可以处理哪些URL请求。

2）在控制器的类定义即方法定义处都可标注 @RequestMapping：

【1】类定义处：提供初步的请求映射信息。相对于WEB应用的根目录

【2】方法处：提供进一步的细分映射信息。相对于类定义处的URL。

【3】若类定义处未标注 @RequestMapping，则方法处标记的URL相当于WEB应用的根目录

【参考springMVCDemo2/handler/requestMapping、testRequestMappingOfClass\_index.jsp、testRequestMappingOfMethod\_index.jsp】

3）DispatcherServlet截获请求后，就通过控制器上 @RequestMapping 提供的映射信息确定请求所对应的处理方法。

4）@RequestMapping除了可以使用请求URL映射请求外，还可以使用请求方法、请求参数、请求头映射请求。

5）@RequestMapping的value、method、params及headers分别表示请求URL、请求方法、请求参数、请求头的映射条件，他们之间是与的关系，联合使用多个条件可以让请求映射更加精确化。

【参考SpringMVCDemo2/requestMethod、testMethodRequest\_index.jsp】

6）params和headers支持简单的表达式：

【1】param1：表示请求必须包含名为param1的请求参数

【2】！param1：表示请求不能包含名为param1的请求参数

【3】param1 ！= value1：表示请求包含名为param1的请求参数，但其值不能为value1

【4】{“param1=value1”，“param2”}：表示请求包含param1和param2两个参数，且param1参数的值必须为value1

【参考SpringMVCDemo2/paramRequest、testParamRequest\_index.jsp】

7）@RequestMapping支持Ant风格的URL：

【1】Ant风格资源地址支持3种匹配符：

① ？：匹配文件名中的一个字符

② \*：匹配文件名中的任意字符

③ \*\*：匹配多层路径

### 2、@PathVariable：

1）@PathVariable映射URL绑定的占位符

2）通过@PathVariable可以将URL中占位符参数绑定到控制器处理方法的入参中，即：URL中的 {XXX} 占位符可以通过 @PathVariable（“XXX”）绑定到操作方法的入参中。

【参考SpringMVCDemo2/pathVariable】

### 3、@RequestParam：

1）使用@RequestParam可以把请求参数传递给请求方法

【1】value：参数名

【2】required：是否必须，默认为true，表示请求参数中必须包含对应的参数，若不存在，将抛出异常

【参考SpringMVCDemo2/requestParam】

### @RequestHeader：

1. @RequestHeader：绑定请求报头的属性值
2. 请求头包含了若干个属性，服务器可以据此获取客户端的信息

【参考SpringMVCDemo2/requestHeader】

### @CookieValue

1. 绑定请求中的cookie值

【参考springMVCDemo2/cookieValue】

### 6、@InitBinder

1）由 @initBinder标识的方法可以对WebDataBinder对象进行初始化。

2）WebDataBinder是DataBinder的子类，用于完成由表单字段到JavaBean属性的绑定。

3）@InitBinder标注的方法不能有返回值，该方法必须声明为void

4）@InitBinder标注的方法传递的参数通常是WebDataBinder

## 四、使用POJO对象绑定请求参数：

### 1、使用POJO对象绑定请求参数：

1）SpringMVC会按照请求参数名和POJO属性名进行自动匹配，自动为该对象填充属性值。支持级联属性。如：dept\_deptId、dept\_address\_tel

【参考springMVCDemo3/pojo、Address.java、User.java、TestPojo.java】

### 2、使用Servlet原生API作为方法参数：

1）SpringMVC的Handler方法可以接受的Servlet API的类型

【1】HttpServletRequest、HTTPServletResponse

【2】HttpSession

【3】java.security.Principal

【4】Locale

【5】InPutStream、OutputStream

【6】Reader、Writer

【具体查看 ServletHandlerMethodInvoker.class】

### 3、处理模型数据：

1）SpringMVC提供了以下几种途径输出模型数据：

【1】ModelAndView：处理方法返回值类型为ModeAndView时，方法体即可通过该对象添加模型数据

①控制器处理方法的返回值如果为ModelAndView，则其既包含视图信息，也包含模型数据信息。

【参考springMVCDemo3/dealModel/TestModelAndView.java】

【2】Map及Model：入参为org.springframework.ui.Model、org.springframework.ui.ModelMap或java.util.Map时，处理方法返回时，Map中的数据会自动添加到模型中

①SpringMVC在内部使用了一个org.springframework.ui.Model接口存储数据。

②具体步骤：

--》SpringMVC在调用方法之前会创建一个隐含的模型对象作为模型数据的存储容器。

--》如果方法的入参为Map或者Model类型，SpringMVC会将隐含模型的引用传递给这些入参。在方法体内，开发者可以通过这个入参对象访问到模型中的所有数据，也可以向模型中添加新的属性数据

【参考SpringMVCDemo3/dealModel/mapOrModel】

【3】@SessionAttributes：将模型中的某个属性暂存到HttpSession中，以便多个请求之间可以共享这个属性

①向之前的ModelAndView、map、ModelMap等都只是将数据存放到request域对象之中，这样只适合转发的情况，那怎么样将数据存放到session域对象呢？

【参考springMVCDemo3/sessionAttributes】

【4】@ModelAttribute：方法入参标注该注解后，入参的对象就会放到数据模型中【@ModelAttribute注解是重难点】

1. 有 @ModelAttribute 标记的方法，会在每个目标方法执行之前被SpringMVC调用【很重要】
2. @ModelAttribute 注解也可以用来修饰目标POJO类型的入参，其value属性值有如下的作用：

①SpringMVC会把value属性值在implicatModel中查找对应的对象，若存在则会直接传入到目标方法的入参中。

②SpringMVC会以value（@ModelAttribute注解的value）为key，POJO类型的对象为value，存入到request中

3）运行流程：

--》执行 @ModelAttribute 注解修饰的方法：从数据库中取出对象，把对象放入到Map中，键为”user”

--》SpringMVC从Map中取出User，并把表单提交的请求参数赋给该User对象的属性

--》SpringMVC把上述对象传入目标方法的参数

### 4、SpringMVC确定目标方法POJO类型入参的过程：

1）确定一个key。

【1】若目标方法的POJO类型的参数没有使用 @ModelAttribute 作为修饰，则 key 为 POJO类名的第一个字母的小写。

【2】若使用了 @ModelAttribute 注解，则 key 为 @ModelAttribute 注解的 value 属性值。

1. 在implicatModel中key对应的类型，若存在，则作为入参传入。

【1】若在 @ModelAttribute 标记的方法中存在Map，且 key 和 1）确定的 key一种，则会获取到。

3）若在implicatModel中不存在key对应的对象，则检查当前的 Handler 是否使用 @SessionAttributes 注解修饰。若使用了该注解，且 @SessionAttributes 注解的value属性值中包含了key，则会从 HttpSession 中获取key所对应的value值。若所对应的value值存在，则直接传入到目标方法的入参中；若不存在，则将抛出异常。

4）若Handler 没有标识 @SessionAttributes 注解或 @SessionAttributes 注解的value值中不包含key，则会通过反射来创建POJO类型的参数，传入为目标方法的参数。

5）SpringMVC会把 key 和 POJO 类型的对象 保存到implicatModel 中，进而保存到 request域对象中。

### 5、总结：

|  |
| --- |
| 情况一：没有使用 @ModelAttribute 注解： |
| 情况二：使用 @ModelAttribute 注解，且定义了value值为“abc” |
| 情况三：使用了 @ModelAttribute 注解，但是没有定义value值 |

1. 对于POJO入参，不论是否使用了 @ModelAttribute 注解，SpringMVC 都会把POJO放入到 request域中。
2. 若是没有使用 @ModelAttribute 注解，request域的键（key）就是POJO类的类名小写【在这里，request域默认的key 就是 user，value就是User对象】【第一种情况】
3. 若是使用了@ModelAttribute 注解，但是没有定义value值，request域的key还是POJO类的类名小写【在这里，request域默认的key 就是 user，value就是User对象】【第三种情况】
4. 若是使用了 @ModelAttribute 注解，且定义了value值，那么request域的key就是定义的value值【在这里request域的key就是abc，value就是User对象】【第二种情况】

## 五、自定义:

### 1、自定义视图：

1）我们自定的视图要implements View接口

2）我们自定义的视图类要配置到扫描的包里面

【参考SpringMVCDemo3/viewResolver、TestHelloView.java】

### 2、自定义类型转换器：

1）ConversionService是Spring类型转换体系的核心接口。

2）可以利用ConversionServiceFactoryBean在Spring的IOC容器中定义一个ConversionService。Spring将自动识别出IOC容器中的ConversionService，并在Bean属性配置及SpringMVC处理方法入参绑定等场合使用它进行数据的转换。

3）可通过ConversionServiceFactoryBean的converters属性注册自定义的类型转换器。

4）Spring定义了三种类型的转换器接口，实现任意一个转换器接口都可以作为自定义转换器注册到ConversionServiceFactoryBean中：

【1】Converter<S,T>：将S类型对象转换成T类型对象

【2】ConverterFactory：将相同系列多个“同质”Converter封装在一起。如果希望将一种类型的对象转换成另一种类型及其子类的对象（例如将String转换成Number及Number子类对象），可使用该转换器工厂类。

【3】GenericConverter：会根据源类对象及目标类对象所在的宿主类中的上下文信息进行全类型转换。

【参考springMVCDemo4/testConversionServiceConverter\_index.jsp、TestConversionServiceConverter.java、UserConverter.java】

### 3、自定义错误消息（提示消息的国际化）：

1）每个属性在数据绑定和数据校验发生错误时，都会生成一个对应的FieldError对象

2）当一个属性校验失败后，校验框架会为该属性生成4个消息代码，这些代码以校验注解类名为前缀，结合modelAttribute、属性名及属性类型名生成多个对应的消息代码：例如User类中的password属性标注了一个@Pattern注解，当该属性值不满足@Pattern所定义的规则时，就会产生以下4个错误代码。

--》Pattern.user.password

--》Pattern.password

--》Pattern.java.lang.String

--》Pattern

3）当使用SpringMVC标签显示错误消息的时候，SpringMVC会查看WEB上下文是否装配了对应的国际化消息，如果没有，则会显示默认的错误消息，否则使用国际化消息

### 4、自定义拦截器：

1）SpringMVC也可以使用拦截器对请求进行拦截处理，用户可以自定义拦截器来实现特定的功能，自定义拦截器必须实现HandlerInterceptor接口

【1】preHandler()：这个方法在业务处理器处理请求之前被调用，在该方法中对用户请求request进行处理。如果程序员决定该拦截器对请求进行拦截处理后还需要调用其他拦截器，或者是业务处理器去进行处理，则返回true；如果程序员决定不需要再调用其他组件去处理请求，则返回false。

【2】postHandler()：这个方法在业务处理器处理完请求之后，但是DispatcherServlet向客户端返回响应前被调用，在该方法中对用户请求request进行处理。

【3】afterCompletion()：这个方法在DispatcherServlet处理完请求后被调用，可以在该方法中进行一些资源清理的操作

【参考：springMVCDemo4/test/TestMyInterceptor1.java】

【参考：springMVCDemo4/test/TestMyInterceptor2.java】

|  |
| --- |
| 拦截器执行顺序：拦截器在配置文件中定义的顺序  多个拦截器方法的执行顺序： |

## 六、知识点：

### 1、关于重定向：

1）一般情况下，控制器返回字符串类型的值会被当成逻辑视图名处理

2）如果返回的字符串中带 forward: 或 redirect: 前缀时，SpringMVC会对他们进行特殊处理：

--》将forward: 和 redirect: 当成指示符，其后的字符串作为URL来处理

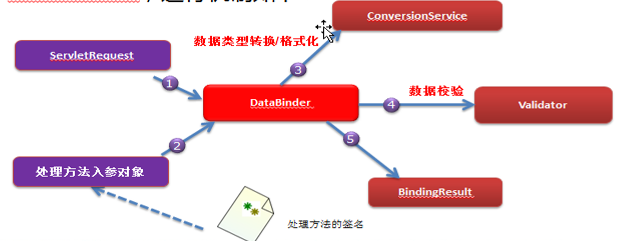
### 2、数据绑定流程：

1）SpringMVC主框架将ServletRequest对象以及目标方法的入参实例传递给WebDataBinderFactory实例，以创建DataBinder实例对象。

2）DataBinder调用装配在SpringMVC上下文中ConversionService组件进行数据类型转换、数据格式化工作。将Servlet中的请求信息填充到入参对象中。

3）调用Validator组件对已经绑定了请求消息的入参对象进行数据合法性校验，并最终生成数据绑定结果BindingData对象

4）SpringMCM抽取BindingResult中的入参对象和校验错误对象，将他们赋给处理方法的响应入参。



### 3、<mvc:annotation-driven />配置：

1）一般开发的时候都要显示的声明 <mvc:annotation-driven>

2) <mvc:annotation-driven>会自动注册RequestMappingHandlerMapping、RequestMappingHandlerAdapter、ExceptionHandlerExceptionResolver三个bean

3）<mvc:annotation-driven>还提供以下支持：

【1】支持使用 ConversionService 实例对表单参数进行类型转换

【2】支持使用 @NumberFormatannotation、@DateTimeFormat注解完成数据类型的格式化

【3】支持使用 @Valid 注解对 JavaBean 实例进行JSR 303验证

【4】支持使用 @RequestBody 和 @ResponseBody注解

### 4、数据的格式化：

1）对于数据的格式化一般是有Date和Number两种类型

2）只需要在相应的Date、Number类型的属性上加上注解 @DateTimeFormat、@NumberFormat了。

|  |
| --- |
|  |

在提交的表单页面，就要使用相应的格式：

|  |
| --- |
|  |

1. 对属性对象的输入/输出进行格式化，从本质上来将依然属于“类型转化”的范畴。
2. Spring在格式化模块中定义了一个实现 ConversionService 接口的 FormattingConversionService实现类，该实现类拓展了GenericConversionService，因此它即具有类型转换的功能，又具有格式化的功能
3. FormattingConversionService拥有一个FormattingConversionServiceFactoryBean工厂类，后者用于在Spring上下文中构造前者
4. FormattingConversionServiceFactoryBean内部已经注册了：

①NumberFormatAnnotationFormatterFactory：支持对数字类型的属性使用 @NumberFormat注解

②JodaDateTimeFormatAnnotationFormatterFactory：支持日期时间类型属性使用 @DateTimeFormat注解

7）装配了FormattingConversionServiceFactoryBean后，就可以在SpringMVC入参绑定及模型数据输出时使用注解驱动了

### 5、JSR303数据校验：

1）JSR 303是java为Bean数据合法性校验提供的标准框架，它已经包含在javaEE6.0中

2）JSR 303通过在Bean属性上标注类似于 @NotNull、@Max等标准的注解指定校验规则，并通过标准的验证接口对Bean进行校验

|  |
| --- |
|  |

1. Hibernate Validator是JSR303的一个参考实现，除支持所有标准的校验注解外，它还支持一下的拓展注解。

|  |
| --- |
|  |

### SpringMVC数据校验：

1. Spring4.0拥有自己独立的数据校验框架，同时支持JSR303标准的校验框架
2. Spring在进行数据绑定时，可同时调用校验框架完成数据校验工作。在SpringMVC中，可直接通过注解驱动的方法进行数据校验
3. Spring的LocalValidatorFactoryBean既实现了Spring的Validator接口，也实现了JSR303的Validator接口。只要在Spring容器中定义了一个LocalValidatorFactoryBean，即可将其注入到需要数据校验的Bean中。
4. <mvc:annotation-driven>会默认装配一个LocalValidatorFactoryBean，通过在处理方法的入参上标注 @valid 注解即可让SpringMVC在完成数据绑定后执行数据校验的工作。
5. 在已经标注了 JSR303 注解的表单/命令对象前标注一个 @valid，SpringMVC框架在将请求参数绑定到该入参对象后，就会调用校验框架根据注解声明的校验规则实施校验。
6. SpringMVC是通过对处理方法签名的约束来保存校验结果的：前一个表单/命令对象的校验结果保存到随后的入参中，这个保存校验结果的入参必须是BindingResult 或 Errors 类型

### 7、关于国际化：

1）问题：

【1】在页面上能够根据浏览器语言设置的情况对文本、时间、数值进行本地化处理

【2】可以在bean中获取国际化资源文件Locale对应的消息

【3】可以通过超链接切换Locale，而不再依赖于浏览器语言设置情况

1. 解决：

【1】使用JSTL标签的ftm标签

【2】在bean中注入ResourceBundleMessageSource的实例，使用其对应的getMessage()方法即可

【3】配置LocaleResolver和LocaleChangeInterceptor

### 8、文件上传：

1）SpringMVC为文件上传提供了直接的支持，这种支持是通过即插即用的MultipartResolver实现的。Spring用Jakarta Commons Fileupload技术实现了一个MultipartResolver实现类：CommonsMultipartResolver

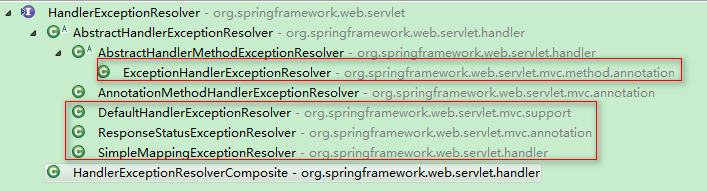
2）SpringMVC上下文中默认没有装配MultipartResolver，因此默认情况下不能处理文件的上传工作，如果想使用Spring的文件上传功能，需先在上下文中配置MultipartResolver

【参考springMVCDemo4/fileupload】

### 9、异常处理：

1）SpringMVC通过HandlerExceptionResolver处理程序的异常，包括Handler映射、数据绑定以及目标方法执行时发生的异常。

2）SpringMVC提供的HandlerExceptionResolver 的实现类（主要是下面4个）



1. DispatcherServlet默认装配的HandlerExceptionResolver：

【1】没有使用 <mvc:annotation-driven/>配置：

①~~AnnotationMethodHandlerExceptionResolver~~（已过时）

②ResponseStatusExceptionResolver

③DefaultHandlerExceptionResolver

【2】使用了 <mvc:annotation-driven/>配置：

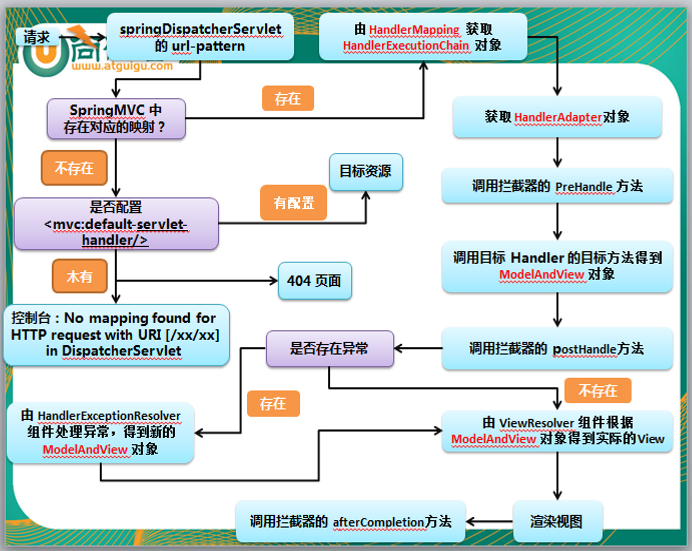
①ExceptionHandlerExceptionResolver

②ResponseStatusExceptionResolver

③DefaultHandlerExceptionResolver

【参考：springMVCDemo4/exception】

## 七、SpringMVC运行流程：



## 八、Spring与SpringMVC整合：

### 1、需要整合吗？

1） SpringMVC本来就是Spring的一个模块，还需要整合Spring与SpringMVC吗？即是否需要加入Spring的IOC容器？也即是否需要在web.xml文件中配置启动Spring IOC容器的ContextLoaderListener？