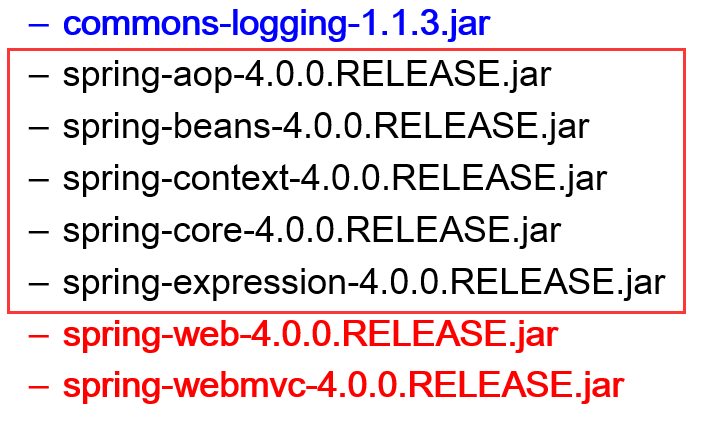
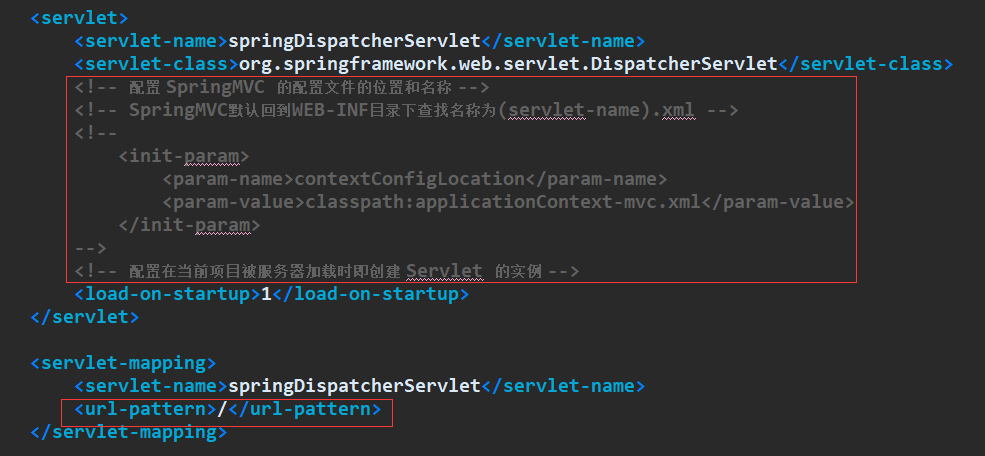
最后一页的图很重要

SpringMVC是基于Servlet的，核心处理器DispatcherServlet

导入包：Spring+SpringMVC

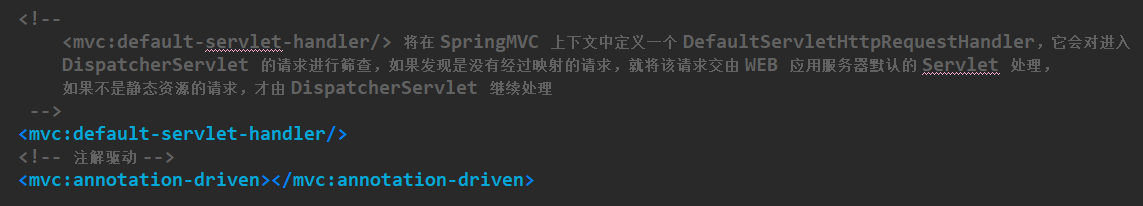


配置web.xml



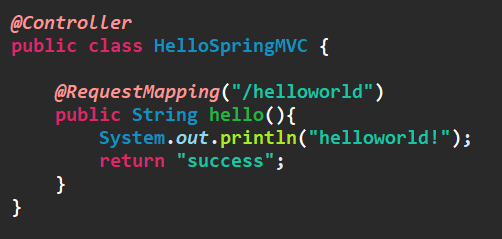
由于SpringMVC对任何请求都会拦截，在请求静态资源的时候(JS，图片)，SpringMVC会因为未找到指向的方法，而不去执行此请求，所以获取静态资源就会失败

需要在SpringMVC的配置文件中如下配置



这两个也是进行SpringMVC开发的两个标准配置

RequestMapping：



@Controller用于修饰类，表示此类是请求的处理器类

@RequestMapping表示请求的路径，可以定义在类和方法，定义类时相当于WEB应用的根目录

RequestMapping除了映射请求外，还有其他作用



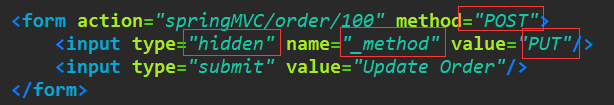
Value：表示请求的路径，支持Ant风格，也可以使用正则表达式

/user/\*/select：\*表示任意多个字符

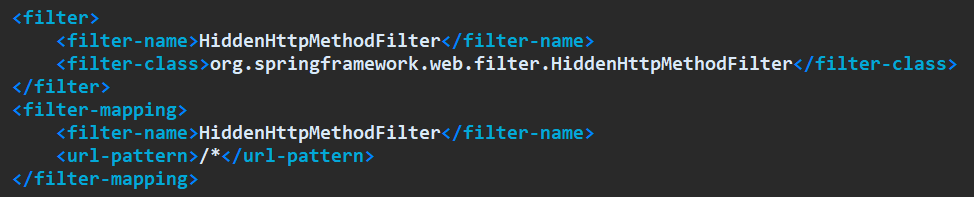
/user/\*\*/select：\*\*表示多层路径

/user/select？：？表示一个字符

由于浏览器只会发生get和post请求，想要提交其他请求，如Delete，put请求，需要添加一个请求参数”***\_method***”



这里只能是post提交from表单(post提交更安全)



在web.xml作如上配置，SpringMVC才会去过滤请求，然后执行对应的请求方式(PUT)

HiddenHttpMethodFilter是一个Filter，在执行Servlet之前，获取到\_method属性对应的值，然后将请求包装后交给DispatcherServlet

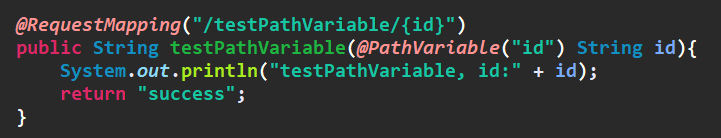
注意点：如果method是PUT或DELETE，且tomcat是8.0的，还要在RequestMapping注解下添加@Response注解，否则会报405 jsps only permit GET POST AND HEAD错误

Params：表示请求必须有age请求参数，!age[不包含]，age=11/age!=11，{“age”,”name”}支持一些简单的表达式(多个属性之间是且关系)

Headers：表示请求头要有一些什么信息，包含一些简单的表达式(和params差不多)

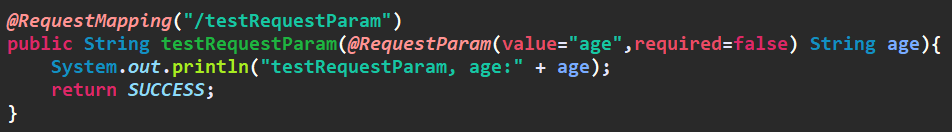
获取请求相关的参数

*PathVariable*：



这里会自动类型转换，PathVariable 主要是支持RestFul风格PathValue只有一个value属性

@RequestParam：



获取请求参数

Value：参数名

Required：是否是必须的，默认为true(如果请求参数中没有，将会报错)

defaultValue：参数为空时的默认值

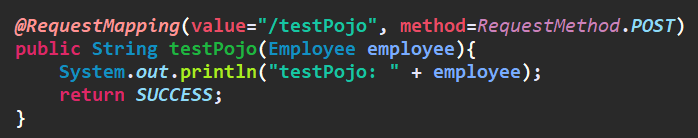
虽然不使用@RequestParam也可以获取到请求的参数(参数名一致)，但@RequestParam的功能更完善

不使用@RequestParam也可以在方法是获取到age，只要请求参数的名称一样(其实这里默认使用了@ModelAttribute注解)

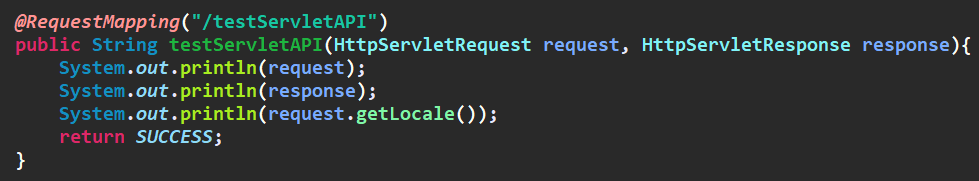
@RequestHeader：

和RequestParam的用法一样，只有这个用来获取头信息

@CookieValue：获取cookie中的信息，用法一样



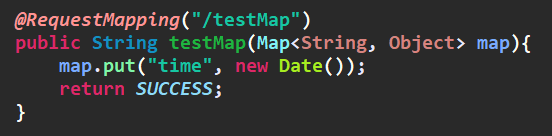
根据参数名，指定将属性写入到POJO中，支持联级属性



支持传入原生的ServletAPI，SpringMVC会自动将原生的ServletAPI作为方法参数传入

SpringMVC中保存数据的方式

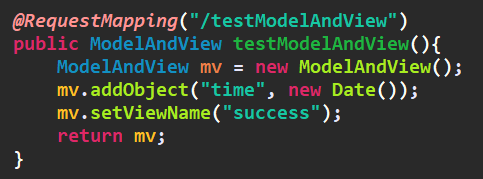
Map/Model/ModelMap:



当方法参数中有Map，Molde，ModelMap时，可以向Map中添加参数，可以在响应的页面中获取到Map中的参数

在执行方法时，SpringMVC会将隐含模型的引用传递给入参(Map)，这个隐含模型是BindingAwareModelMap，而Map，Model，ModelMap是隐含对象的父类

ModelAndView：

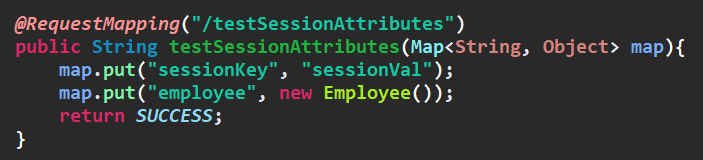


ModelAndView包含了Model和View，Model是数据模型，保存数据，响应页面可以获取这里面的数据，View是视图，，即方法响应的页面，一般返回的方法为String，SpringMVC会自动调用ModelAndView的setViewName，将响应页面加入到View中

@SessionAttribute：



当一个属性需要在多个模块之间共享时，可以使用SessionAttributes注解，SessionAttributes注解只能使用在Controller类上，不能使用在方法上

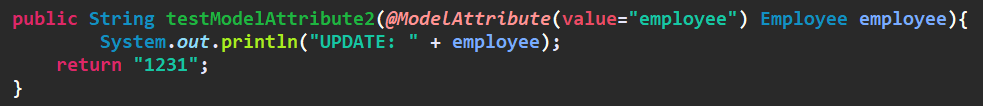


当模块中的某个方法执行时，将属性put到隐含模型中时，如果属性名和@SessionAttribute的value相同或属性类型和@SessionAttribute的types类型相同，则将此属性放入到HttpSession中

如何在方法中获取到Session中的属性：前面说到，在执行方法时，会给方法赋一个隐含模型(BindingAwareModelMap)，这个隐含模型中就有Session中的数据，所以只要在方法入参中声明Map，通过Map就可以引用BindingAwareModelMap

如何区分Session中数据还是其他域中的数据

@ModelAttribute：



如果方法参数为JavaBean，SpringMVC会自动为JavaBean添加ModelAttribute注解，当执行方法时，SpringMVC会先到隐含模型BindingAwareModelMap中去寻找是否有对应的JavaBean(名称为value属性，默认类名第一个字母小写)，因为方法参数中有JavaBean，肯定要先要有JavaBean传递给方法，所以先到隐含对象中获取，如果没有，则到HttpSession中获取(只有当方法所在的类实现了SessionAttribute注解才会到Session中获取，否则不会)，如果没有获取到，则会抛出异常；如果没有声明SessionAttribute注解，隐含模型中没有找到，SpringMVC则会通过反射创建一个JavaBean(如果SpringMVC通过反射创建JavaBean了，那么说明该类肯定没有实现SessionAttribute注解，在Session中要么获取到，要么抛出异常)，JavaBean创建好后，请求参数会对JavaBean中属性进行赋值或覆盖(是否从session中获取，根据方法所在类是否使用SessionAttribute注解)

防止SessionAttribute抛出异常的方法就是让他在隐含模型中就获取到

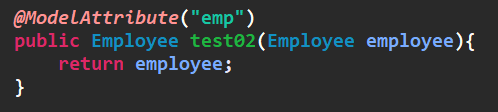
获取到JavaBean后，还会将此JavaBean添加到隐含模型中，可以在页面中获取

方法参数的类型：JavaBean和基本类型/String

方法参数是JavaBean时，会自动添加到隐含模型中

方法参数是基本类型/String时，请求参数会自动赋值到方法的参数上(JavaBean需要创建对象)，可以在方法中使用，但是基本数据类型/String不会添加到隐含模型中，需要手动添加，使用@RequestParam可以对请求参数进行限制(是否是必须的等)，不是使用@RequestParam请求参数也可以在方法中获取到

如果ModelAttribute修饰的参数是String类型(不是JavaBean)，此String只会到隐含模型中去找，不会接收请求参数的属性



@ModelAttribute修饰方法时：此方法会在请求方法被调用之前调用，如果方法有返回值，会将返回值放入到模型对象中

隐含模型(BindingAwareModelMap)和ModelAndView的区别：

隐含对象是用来存储属性的，而ModelAndView是方法实现完后的返回值，用于视图解析，Model保存的其实就是一个隐含对象

ModelAndView中包含了：对View的set/getView()，set/getViewName()操作，对Model的getModel()，addObject()等操作这里的Model就是ModelMap(ModelAndView的父类)

视图：

执行完方法后，会返回一个ModelAndView，SpringMVC会通过ViewResolver去解析View

View和ViewResolver：

View表示视图，常见的有JSP，Excel，JFreeChart等，在SpringMVC中常见的视图有：

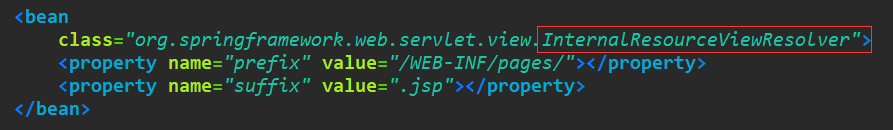


ViewResolver是用于解析View的，将解析的结果发送给浏览器进行渲染，每类View对应一个ViewResolver，常见的ViewResolver

有：(ViewResolver的作用就是根据ModelAndView中View名称找到对应实际的View)



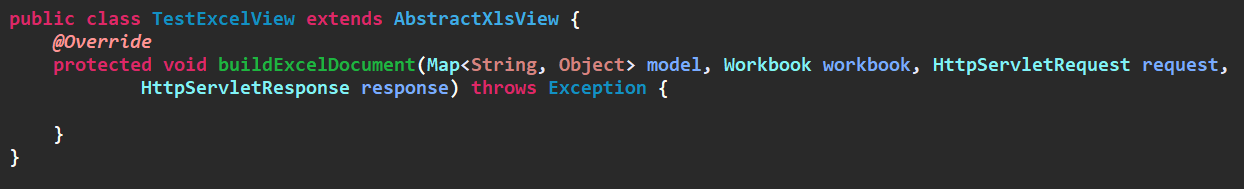
比如常见的解析jsp，首先会声明一个ViewResolver



当方法执行完后，返回一个ModelAndView，然后执行ViewResolver，通过ViewResolver找到真正的视图，然后将真正的视图发送给浏览器，这里的JSP就是一个View

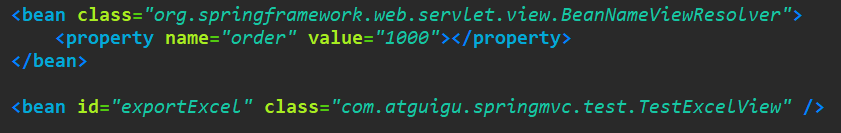
ViewResolver的作用就是根据逻辑视图名称找到物理视图

又比如返回的是一个Excel，首先要定义一个关于Excel的视图，创建一个关于Excel的视图



这里的AbstractXlsView是AbstractExcelView的子类，创建好视图后，可以在视图里定义逻辑，最后通过HttpServletResponse输出给浏览器即可(这里返回的其实就是一个Excel文件)

创建AbstractExcelView对应的视图解析器



ViewResolver的作用就是用来找视图的，创建好视图后，应该告诉SpringMVC容器，而BeanNameViewResolver的作用是根据ModelAndView中的View的视图名称找到对应的视图

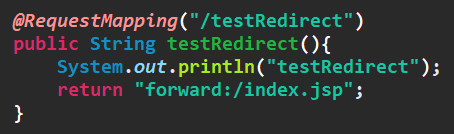
而解析JSP视图的*InternalResourceViewResolver*是根据View的名称到对应路径下寻找指定JSP

当与多个ViewResolver时，可以通过order属性指定ViewResolver的优先级，order越小，优先级越高，默认order最大(Integer.MAX\_VAL)

如果需要自定义可以继承AbstractView，自定义视图解析器可以实现ViewResolver接口

倒过去再看看View和ViewResolver的两张图

重定向和转发：



在SpringMVC中，只要在返回的参数中直接写forward/redirect即可，可以重定向/转发到一个JSP页面，也可以是一个Action请求

RESTful的CRUD：

在RESTful中，GET请求表示获取信息，POST请求表示添加信息，DELETE请求表示删除信息，PUT请求表示修改信息

SpringMVC标签

通过 SpringMVC 的表单标签可以实现将模型数据中的属性和 HTML 表单元素相绑定，以实现表单数据更便捷编辑和表单值的回显

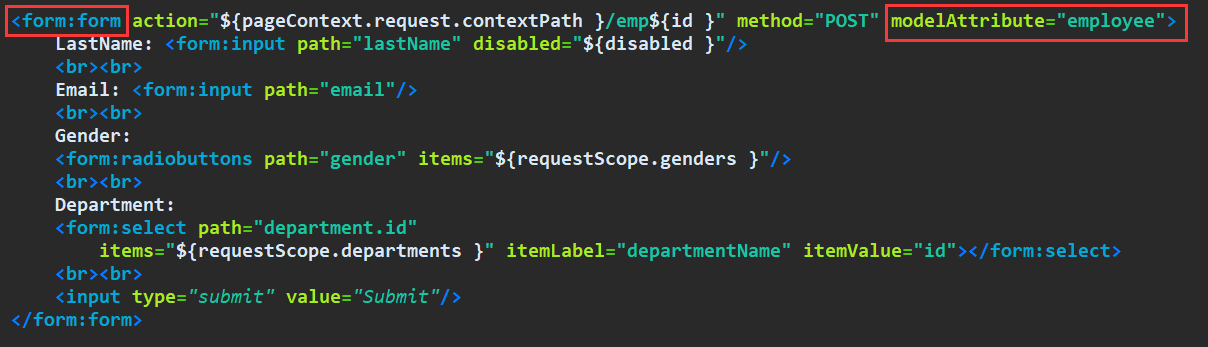
<form:form>、<form:input/>、<form:select/>、form:input、form:password、form:hidden、form:textarea、form:errors、form:checkbox、form:radiobutton等标签

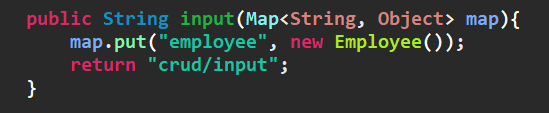
标签中的path属性对应html中的name属性，支持级联属性

HtmlEscape属性表示是否对表单值的HTML特殊字符进行转换，默认TRUE

cssClass对应的CSS样式

cssErrorClass表单组件的数据存在错误时，采取的CSS样式





SpringMVC在是使用form标签时，需要指定一个modelAttribute，而form标签中的input标签的path就是modelAttribute对应JavaBean的属性名，SpringMVC会到隐含模型中寻找对应的JavaBean，如果没有会报错，所以需要手动的添加一个new Employee对象(form标签是SpringMVC自己的标签)，SpringMVC就会自动会显隐含模型中的JavaBean，如果未指定modelAttribute属性，SpringMVC会自动到隐含对象中获取名为command的Bean，如果此bean也不存在，则会抛异常

SpringMVC中数据绑定流程：

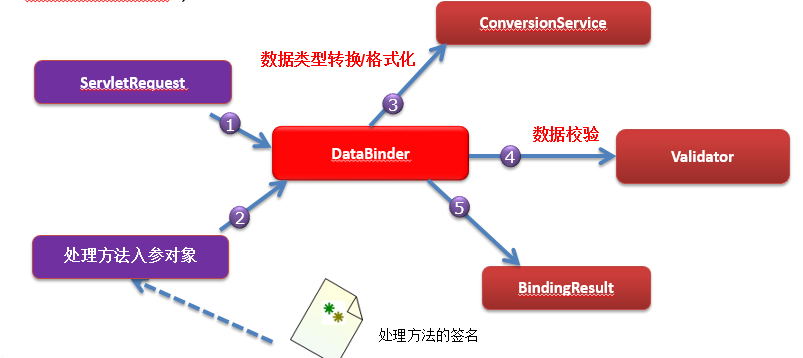
SpringMVC会将request对应以及目标方法(@RequestMapping对应的方法)的入参实例传递给WebDataBinderFactory实例，

通过WebDataBinderFactory返回一个DataBinder对象，

DataBinder通过调用SpringMVC上下文中ConversionService组件进行数据类型转换，数据格式化操作，并将Servlet中的请求信息填充到方法入参的对象中(类型转换在传入到方法参数之前执行)

调用Validator组件对绑定请求消息的入参对象(需要校验的对象)进行合法性校验，并生成数据绑定结果对象BindingData

Spring MVC 抽取 BindingResult 中的入参对象和校验错误对象，将它们赋给处理方法的响应入参

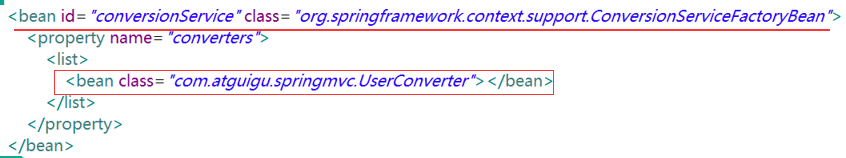


数据的转换：

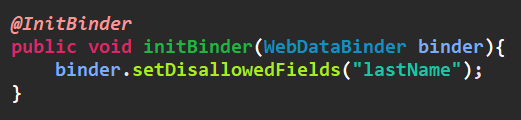
类型转换：SpringMVC中有很多内置的转换器可以将String转化为基本数据类型

自定义类型转换器

通过实现Converter<S,T>/ConverterFactory/GenericConverter接口实现自定义类型转换器



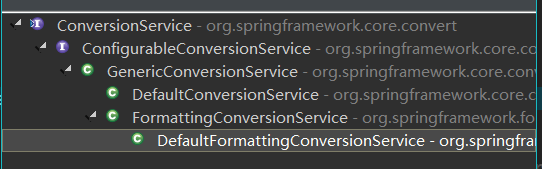
实现自定义转换器后，还要告知SpringMVC转换的存在



@InitBinder用于标志方法，方法参数是WebDataBinder(DataBinder的子类)，方法无返回值为void，用于声明哪些属性不会被JavaBean绑定，比如请求参数包含lastName=jack age=123 role=dep(Object的属性)，则在输出Object的结尾为[lastName=null,age=123,role=dep]

数据的格式化：

ConversionService提供了对数据转换和格式化操作

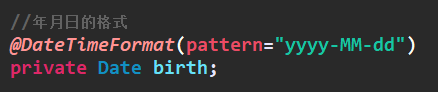


FormattingConversionService实现了对数据的转换

DefaultFormattingConversionService实现了对数据的格式化

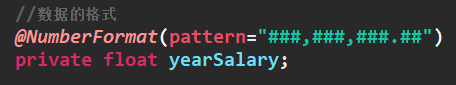
数据格式化有NumberFormat和DateTimeForMat





@DateTimeFormat可以对java.util.Date、java.util.Calendar、java.long.Long时间类型进行处理

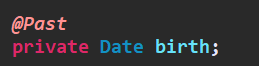
Pattern转换的格式



@NumberFormat对数字进行格式化

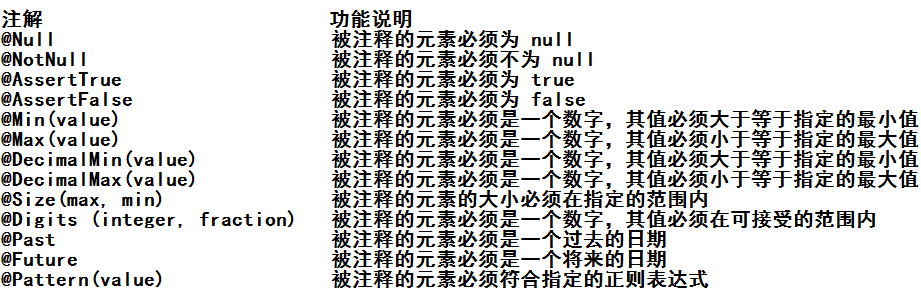
Pattern表示格式化的样式[Style用于执行样式类似[Style.NUMBER（正常数字类型）、 Style.CURRENCY（货币类型）、 Style.PERCENT（百分数类型）]

合法性校验：

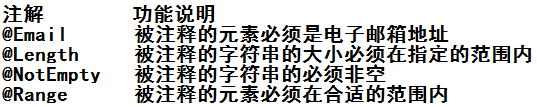


要求birth的日期必须是以前的时间

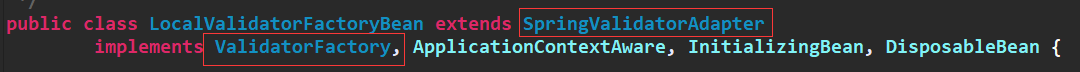
可以通过JSR 303对数据进行合法性校验，JSR是java内置校验框架



还可以使用Hibernate Validator(是JSR 303的一个参考实现)，除了支持标准校验，还支持以下扩展注解

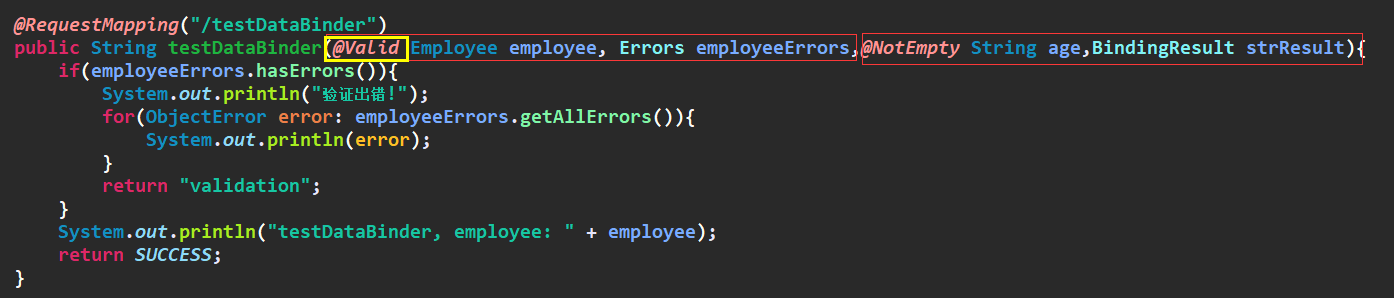


SpringMVC拥有自己独立的数据校验框架，同时支持JSR 303



LocalValidatorFactoryBean既实现了Spring的Validator，还实现了JSR 303的Validator

<mvc:annotation-driven/>会自动装配LocalValidatorFactoryBean



如果对象中的属性需要进行校验，使用@Valid

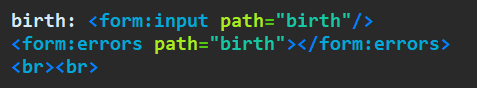
校验结果封装到BindingResult或Errors类中

当参数中有多个需要校验时，需要和校验结果成对出现，但是最终所有的校验错误都会被添加到隐含模型中

方法参数中有没有Error都无所谓，如果有可以获取错，且成对出现，如果没有，错误信息会自动保存到隐含模型中

隐含模型中的所有数据最总都会通过HttpServletRequest暴露给JSP

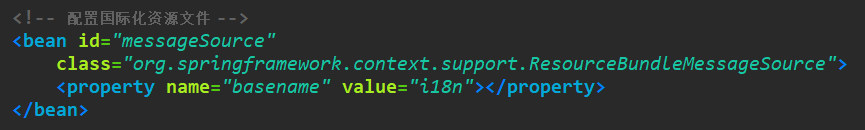
获取错误信息：



错误信息：

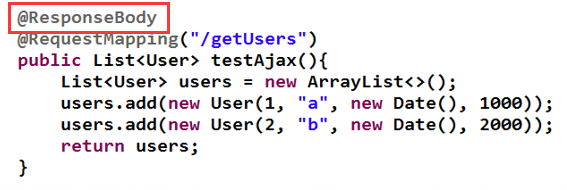
当数据校验失败或绑定失败后，都会生成一个FieldError对象，比如在验证User类的password属性不能为空时(@NotEmpty)，会生成4个错误代码：NotEmpty.user.password NotEmpty.apssword NotEmpty.java.lang.String NotEmpty在获取错消息的时候可以通过NotEmpty.user.password，而其他三个用于模糊获取

获取规则：验证规则.表单modelAttribute值.属性名(对于类型不匹配而言, 其验证规则为 typeMismatch)



配置资源文件

返回JSON对象：

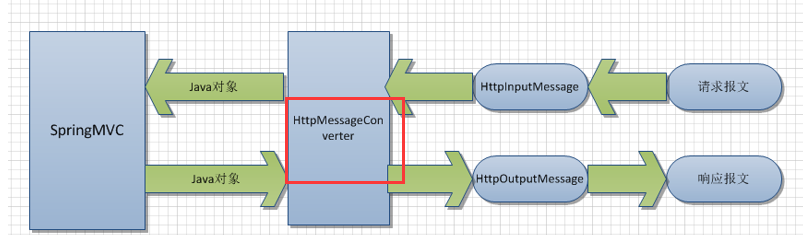
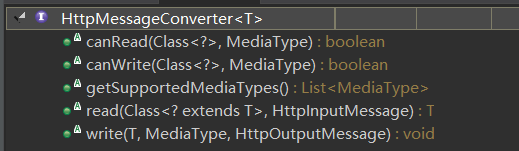


@ResponseBody返回的是一个JSON格式的字符串

需要导入Jackson-Annotations.jar，Jackson-core.jar,Jackson-databind.jar，才能将Object转化为字符串

如何返回的是JSON格式字符串

HttpMessageConverter<T>：将请求转化为对象，将对象转化为输出

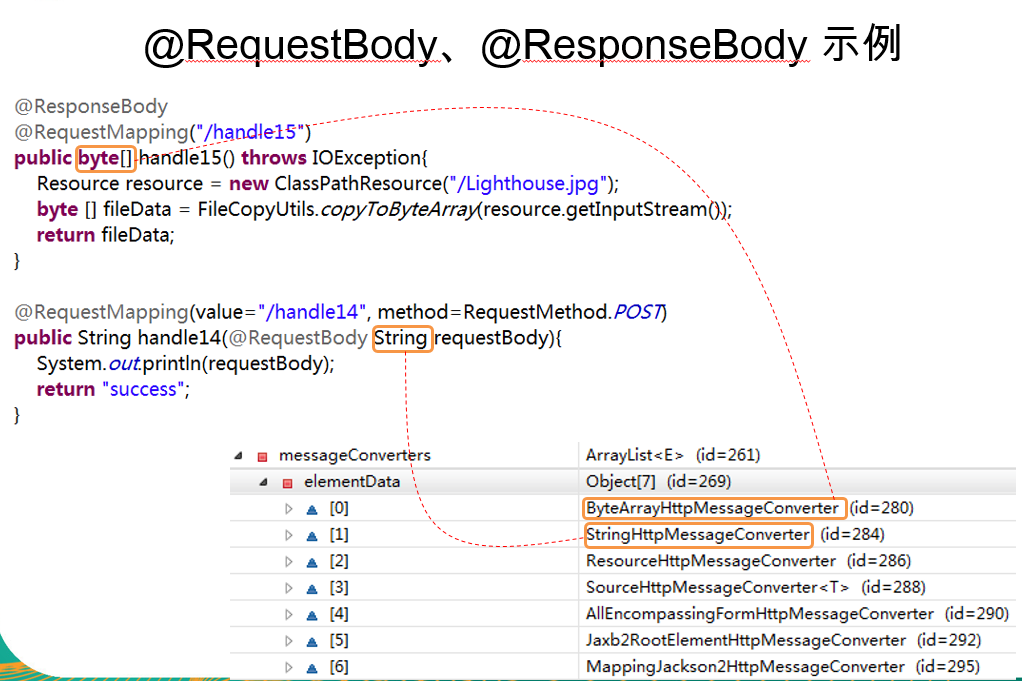
HttpMessageConverter<T>实现类：



实现将请求信息转化为Object，将Object作为响应输出

@RequestBody/@ResponseBody对方法进行标注

HttpEntity<T>/ResponseEntity<T>作为处理方法的入参或返回值

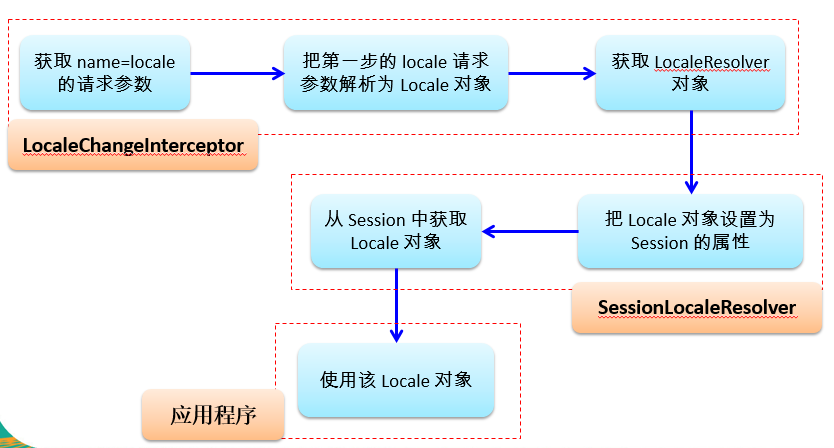


当方法参数声明为RequestBody时，SpringMVC会将请求参数转化为对应的类，当通过POST表单提交name=jack、age=12时，@RequestBody修饰的是一个String属性，所以将输入的参数转为String，本来是要映射到JavaBean中的，但是执行了StringHttpMessageConverter，所以requestbody的结果为name=jack&age=12

当方法被@ResponseBody修饰时，会根据方法的返回值，找到对应的HttpMessageConverter



国际化：



本地化解析器：

AcceptHeaderLocaleResolver：根据 HTTP 请求头的 Accept-Language 参数确定本地化类型

CookieLocaleResolver：根据指定的 Cookie 值确定本地化类型

SessionLocaleResolver：根据 Session 中特定的属性确定本地化类型

本地化拦截器：

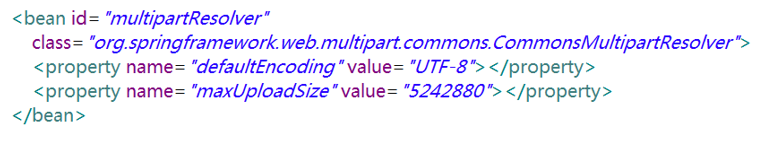
LocaleChangeInterceptor：从请求参数中获取本次请求对应的本地化类型



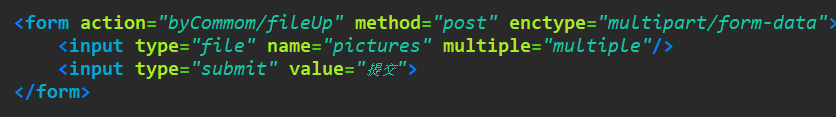
显示国际化对应的资源

文件上传和下载：

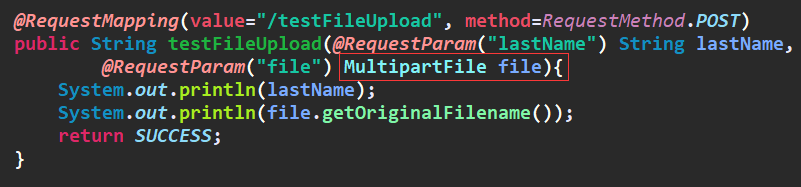
SpringMVC为文件的上传提供了直接支持，通过MultipartResolver实现，具体实现类CommonsMultipartResovler



SpringMVC默认没有装配MultipartResolver



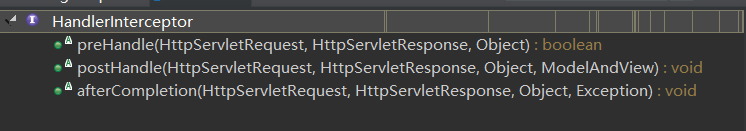
上传文件的实现需要指定enctype，这样SpringMVC才知道这是文件的上传操作(导入commons-fileupload，commons-io包)



下载可以通过ResponseEntity<byte[]>实现(HttpMessageConverter)，以流的方式输出

SpringMVC的拦截器(HandlerInterceptor)

拦截器的作用就是做一些预处理和后处理



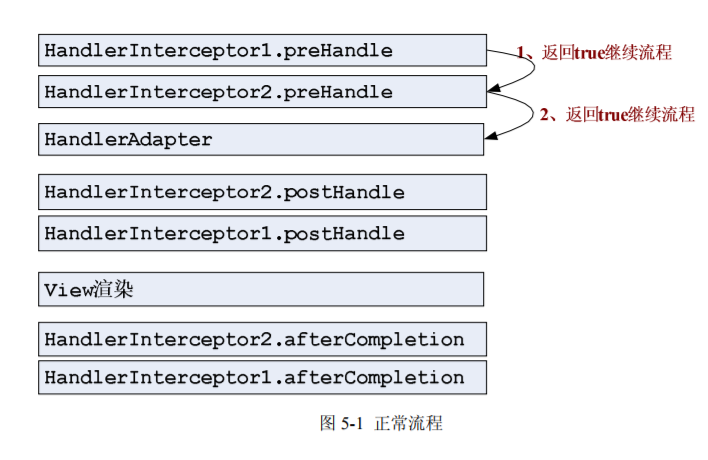
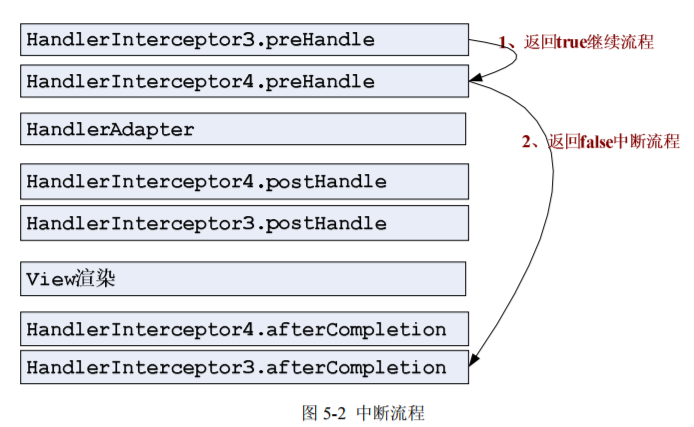
SpringMVC的拦截器共有三个方法：

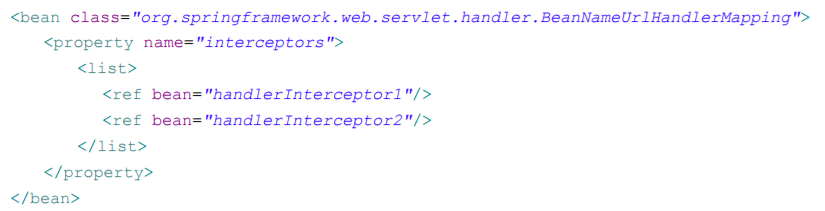
PreHandler：预处理，在执行Controller方法前调用，返回值为true执行下一个拦截器或处理器(或者Controller的方法)，返回值为false，中断处理，通过response返回响应

PostHandle：后处理回调方法，处理器执行完后，在渲染页面之前执行

AfterCompletion：在页面渲染完毕后执行

继承HandlerInterceptor需要实现3个方法，可以实现HandlerInterceptorAdapter，可以只实现一个方法



声明需要执行的拦截器，因为拦截器的获取是在执行HandlerMapping时添加到HandlerExecutionChain中的

Spring常用组件：

·DispatcherServlet：前端控制器

·HandlerMapping：请求映射到处理器，找谁来处理

·HandlerAdapter：把处理器包装为适配器，用来支持多种类型的处理器

·ViewResolver：视图解析器，找谁来处理返回的页面

·Controller：页面控制器、动作处理器

·LocalResolver：本地化、国际化

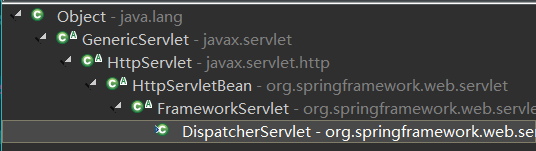
·MultipartResolver：文件上传解析器

·HandlerExceptionResolver：异常处理器

当SpringMVC启动后，会去加载SpringMVC的配置文件，主要是将请求路径和对象的方法找到，并将结果保存到HandlerMapping中

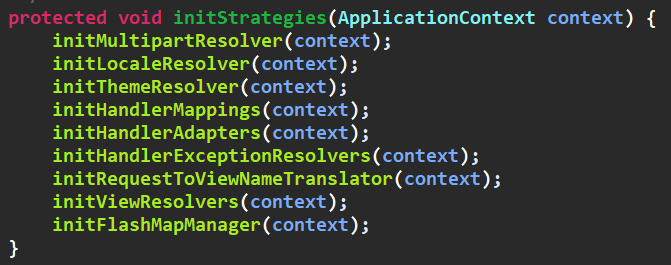
DispatcherServlet是一个Servlet，做些初始化和调度作用

初始化操作：



HttpServletBean对init()方法进行重写主要是用于获取DispatcherServlet的初始化参数(contextConfigLocation等)

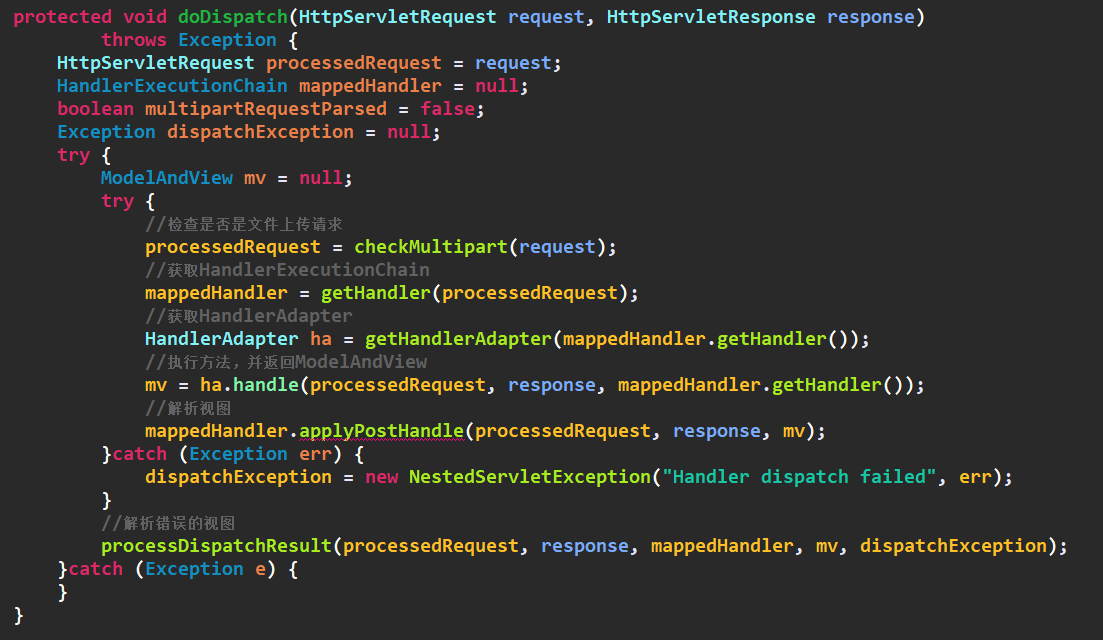
FrameworkServlet主要的作用是初始化上下文(加载SpringMVC的配置文件)



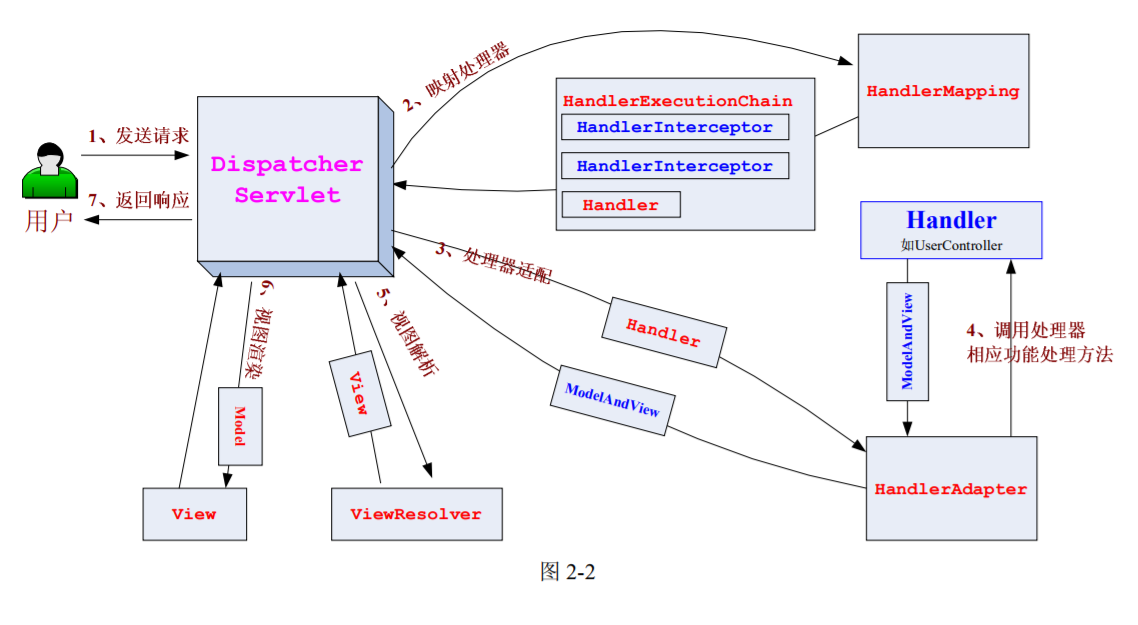
DispatcherServlet初始化化一些默认的配置，并将结果存放到ApplicationContext(IOC)中

这些默认配置都可以在spring-webmvc org.springframework.web.servlet DispatcherServlet.properties下查看

DispatcherServlet的调度作用：





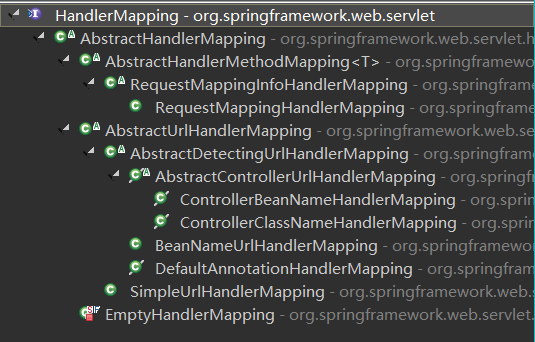


执行getHandler(processedRequest)返回一个HandlerExecuteChain

getHandler首先会到HandlerMapping中寻找HandlerExecutionChain



HandlerMapping找到对应的处理器(Controller)，HandlerMapping的种类很多，默认使用RequestMappingHandlerMapping



找到对应的HandlerMapping后就是获取HandlerExecutionChain，执行的是AbstractHandlerMapping的getHandler方法，但是在getHandler方法中调用getHandlerInternal方法获取handler(在SpringMVC中，很多地方都是Xxx只是声明，但是实际的处理方法是XxxInternal )

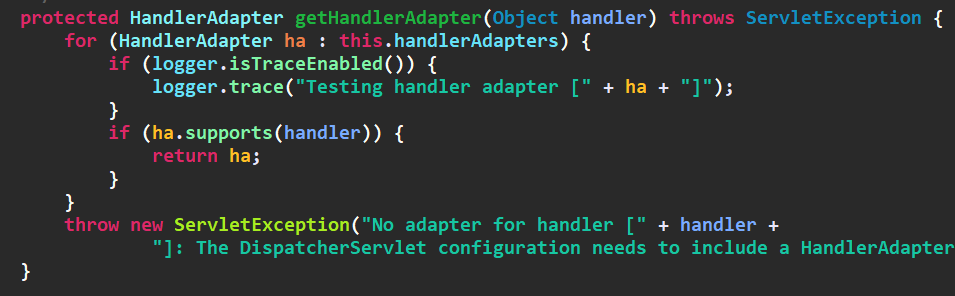


getHandlerInternal获取到handler(实际就是处理器的类路径和方法名)，getHandlerExecutionChain获取的是HandlerInterceptor

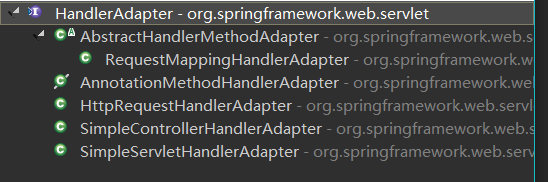


最终返回的是一个HandlerExecutionChain：HandlerInterceptor…. + Handler

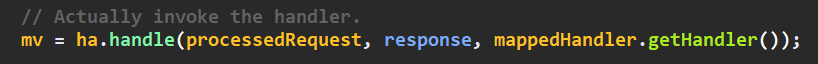
然后就是获取HandlerAdapter



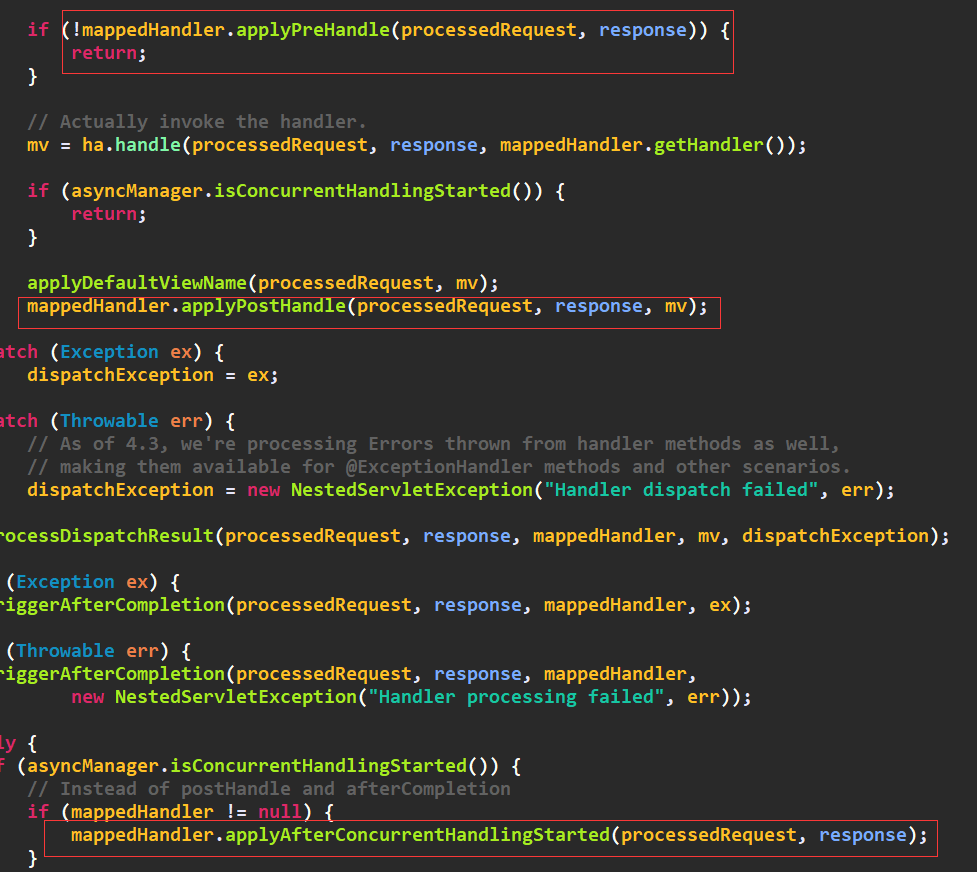
找到合适的HandlerAdapter来执行处理器



获取到HandlerAdapter后，就可以执行处理器，返回ModelAndView



执行HandlerInterceptor的地方



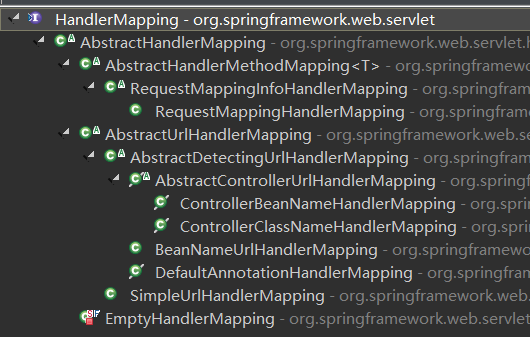
接下来就是解析ModelAndView

隐含模型和Servlet的域对象

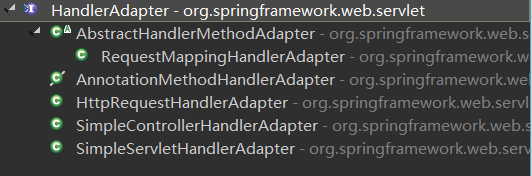
在执行处理器方法之前，SpringMVC会到HttpSession中获取所有的属性保存到隐含模型中，当处理器方法执行完后，会将隐含模型中的属性放入到HttpServletRequest和HttpSession(实现@SessionAttribute)中

HandlerMapping和HandlerAdapter：

HandlerMapping就是找到处理器的位置，如果采用的是实现Controller接口的方式创建处理器，在寻找的时候就得通过BeanNameUrlHandlerMapping，而采用注解的方式实现需要通过RequestMappingHandlerMapping找到



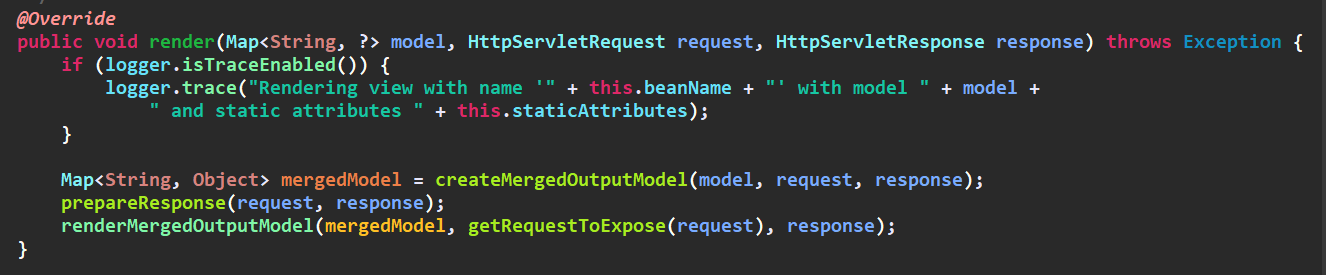
HandlerAdapter是为了找到合适执行处理器的HandlerAdapter，在未使用注解之前是通过RequestMappingHandlerAdapter执行处理器方法，而采用注解后采用SimpleControllerHandlerAdapter



HandlerAdapter采用的适配器模式，适配器模式就是为了解决A不能访问B的问题，A不能直接访问B，但是可以在A和B之间创建一个适配器，这样A就可以访问B

视图(View)

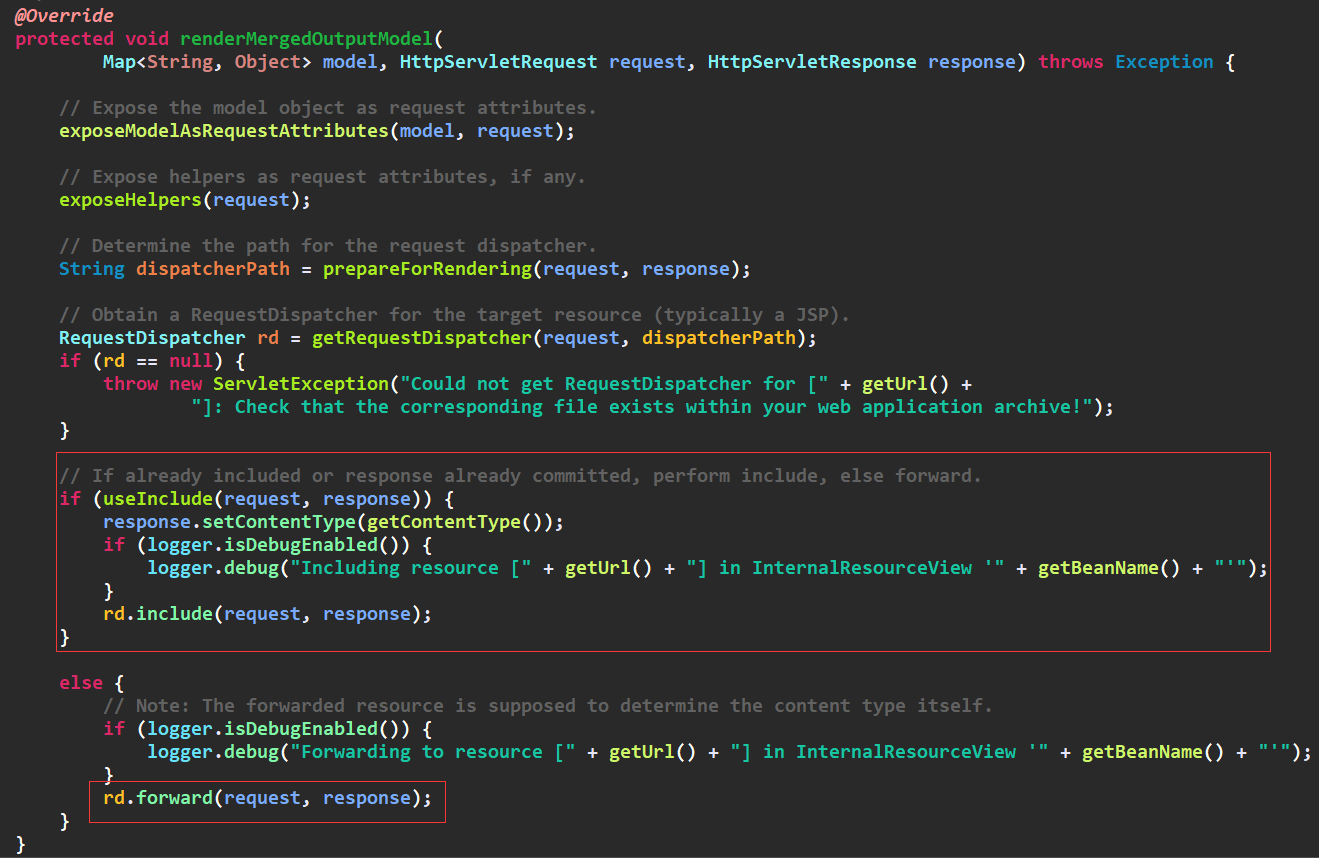
View是接口，对接口中的方法进行了实现，关键方法是render



createMergedOutputModel：将model和request中的参数保存到mergedModel中

prepareResponse：存放头部信息

renderMergedOutputModel：此方法在具体的实现类中实现，普通的JSP是使用InternalResourceView进行实现的



exposeModelAsRequestAttributes：将model(mergedModel)中的数据放入request中

prepareForRendering：检查是否有跳转页面

rd.include(request, response)：加载页面

rd.forward(request, response)：转发操作，将请求响应给用户

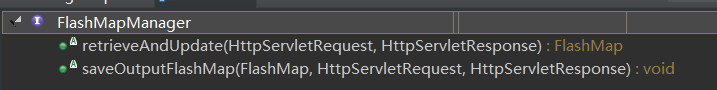
在SpringMVC中，请求转发是不会丢掉原来的数据

当返回值是”redirect：get”的时候，当前View会被转化为RedirectView

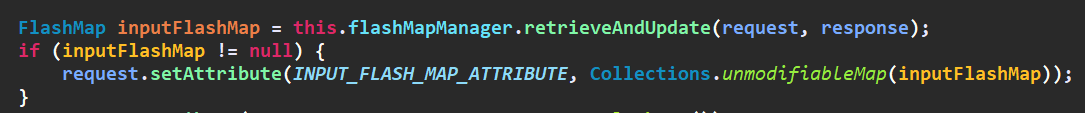


在RedirectView的renderMergedOutputModel方法，会将原来的数据保存到flashMap中，然后执行转发操作

flashMapManager.saveOutputFlashMap(flashMap, request, response)：是将flashMap保存到session中

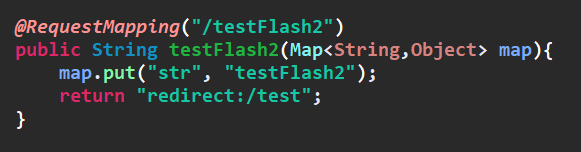
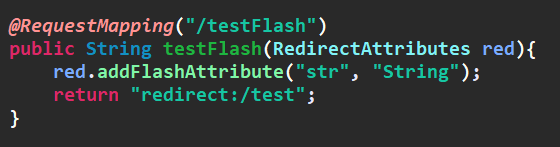


retrieveAndUpdate：是从Session中获取数据



在DispatcherService的doService方法中有一个获取FlashMap的操作，从Session中获取FlashMap后，将结果保存到request中，

从Session中获取到FlashMap后，将Session中对应的FlashMap给移出掉



如果将重定向的数据保存在Model中，则在重定向时，地址栏会显示数据Xxx?key=value&key=value

如果将重定向的数据保存到RedirectAttributes，则会以post的方式传递数据

mvc:annotation-driven：

可以在spring-webmvc包下的org.springframework.web.servlet.config的AnnotationDrivenBeanDefinitionParser.class的说明文档中发现mvc:annotation-driven会自动注册

RequestMappingHandlerMapping、BeanNameUrlHandlerMapping、

RequestMappingHandlerAdapter、HttpRequestHandlerAdapter、SimpleControllerHandlerAdapter

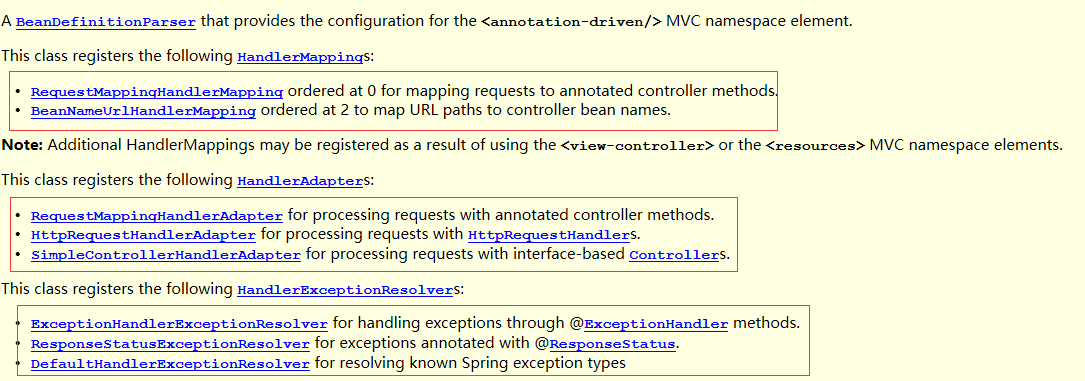
ExceptionHandlerExceptionResolver、ResponseStatusExceptionResolver、DefaultHandlerExceptionResolver

并且支持使用ConversionService实例对表单参数进行类型转换

使用@NumberFormat、@DateTimeFormat对数据进行格式化

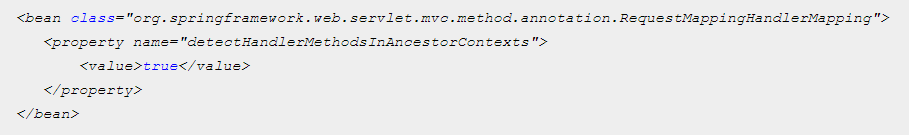
使用@Valid注解对JavaBean进行校验

使用@RequestBody、@ResponseBody



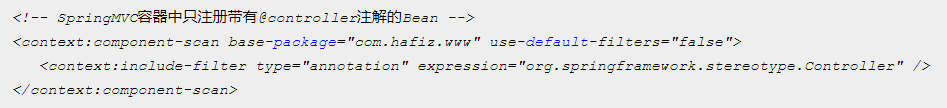
SpringMVC(子)和Spring(父)容器：

在Spring和SpringMVC的配置文件中，都有一个<context:component-scan base-package="com….” />用户加载Bean，首先是执行Spring的配置文件，Spring会将@Controller、@Service、@Respository、@Component注解类加载IOC容器中，此时Spring是不知道有SpringMVC的，所以父容器不能访问子容器，但子容器可以访问父容器，而SpringMVC也会去扫指定的路径，并将@Controller注解的类加载到SpringMVC容器中，而SpringMVC中可通过mvc:annotation-driven加载SpringMVC相关的类，如果不在SpringMVC中去加载@Controller注解类，那么会因为找不到Controller而抛异常，虽然子容器可以访问父容器，但是SpringMVC只会在子容器寻找，而不会到父容器中寻找，可以做如下配置使得SpringMVC会去访问父容器



但是在实际开发中，很少这样，而是让Spring不加载@Controller注解的类，让SpringMVC只加载@Controller注解的类





缓存机制：Last-Modified/ETag

Last-Modified：当浏览器第一次响应时，返回内容和状态码200，并且通过添加一个Last-Modified响应头，表示此文件最后一次在服务器最后更新的时间，下次请求时，浏览器会向服务器发送if-Modified-since请求头，询问资源文件是否被修改，如果未修改返回状态码为304(只有响应头，没有内容)，告诉浏览器直接在自己的缓存中读取

ETag则是在返回的时候返回一个hash，下次请求的时候，判断此hash是否一样

页面在浏览器缓存中存活的时间可以通过Expires进行设置

SpringMVC和Struts2的区别：

Struts2是类级别的拦截，一个类对应一个request，而SpringMVC是方法级别的，一个方法对应一个request

Struts2将每个request进行封装，将request，session等Servlet生命周期封装成一个Map，提供给Action使用，但这样消耗内存

Struts2是基于filter，SpringMVC基于Servlet，在Servlet上进行扩展

SpringMVC集成了Ajax，通过@Responsebody即可，非常方便，支持JSR303校验，灵活

SpringMVC和Spring无缝连接，安全

