# SpringMVC执行流程分析

## org.springframework.web.servlet.DispatcherServlet

### #doDispatch(HttpServletRequest request, HttpServletResponse response)

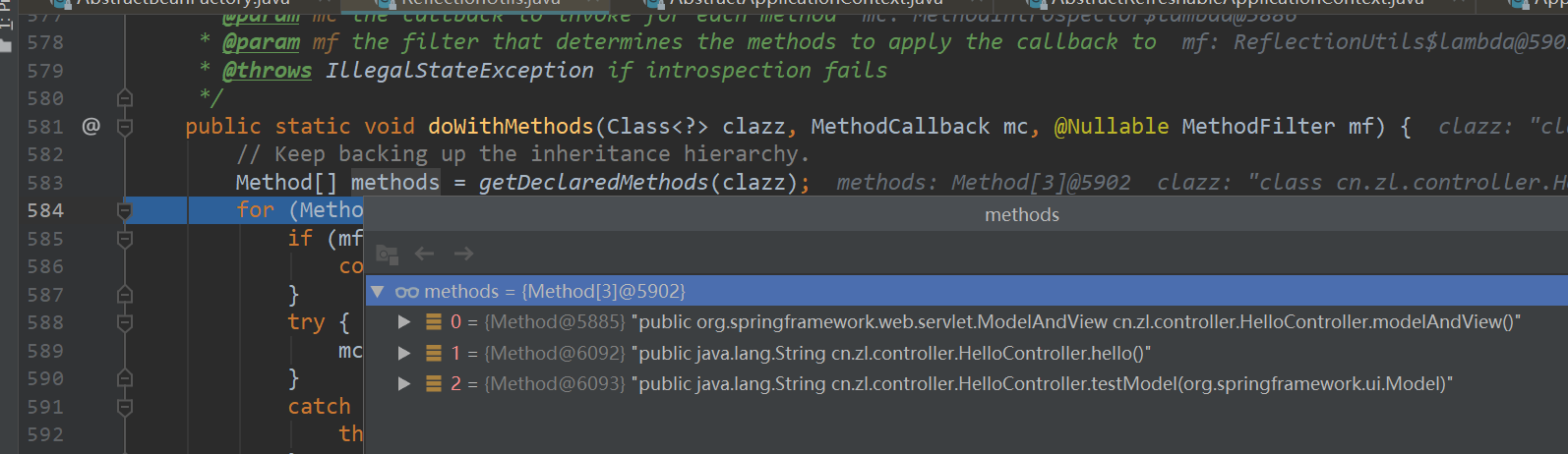
* + 1. org.springframework.web.servlet.DispatcherServlet#checkMultipart:
       1. Convert the request into a multipart request, and make multipart resolver available.
          1. 转换当前请求为 multipart请求，并使 multipart可用
       2. If no multipart resolver is set, simply use the existing request.
          1. 如果没有设置multipart解析器，则使用已存在的请求
    2. org.springframework.web.servlet.DispatcherServlet#getHandler
       1. Determine handler for the current request.
          1. 确定当前请求的处理程序。
    3. org.springframework.web.servlet.DispatcherServlet#getHandlerAdapter
       1. Determine handler adapter for the current request.
          1. 确定当前请求的适配器
    4. Process last-modified header, if supported by the handler.
       1. 处理最后修改的标头，如果处理程序支持。
    5. org.springframework.web.servlet.HandlerExecutionChain#applyPreHandle
       1. Apply preHandle methods of registered interceptors.
          1. 对已注册拦截器应用预(前置)处理方法
    6. org.springframework.web.servlet.HandlerAdapter#handle
       1. Actually invoke the handler.、
          1. 实际调用处理程序
       2. Use the given handler to handle this request.
          1. 使用给定的处理程序来处理此请求。
    7. org.springframework.web.servlet.DispatcherServlet#applyDefaultViewName
       1. org.springframework.web.servlet.DispatcherServlet#getDefaultViewName
          1. Translate the supplied request into a default view name.

将提供的请求转换为默认视图名称。

* + 1. org.springframework.web.servlet.HandlerExecutionChain#applyPostHandle
       1. Apply postHandle methods of registered interceptors.
          1. 对已注册的拦截器应用后置处理方法
    2. org.springframework.web.servlet.DispatcherServlet#processDispatchResult
       1. Handle the result of handler selection and handler invocation, which is either a ModelAndView or an Exception to be resolved to a ModelAndView.
          1. 把程序选择和程序徐调用的结果解析为ModelAndView或者异常

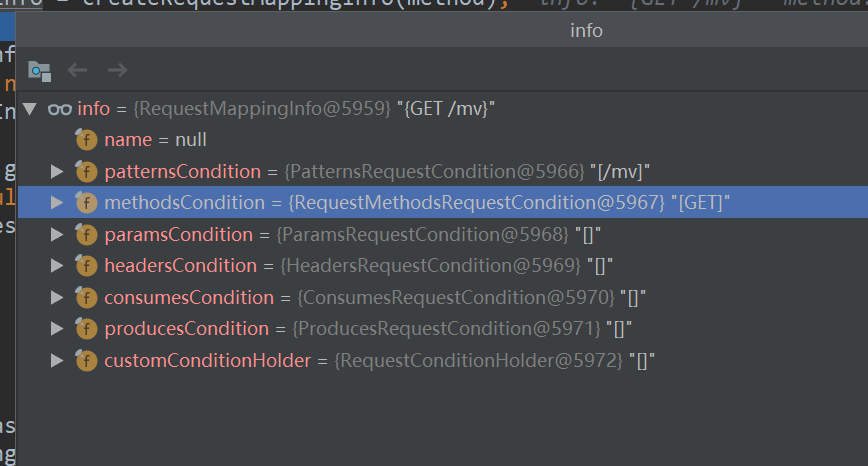
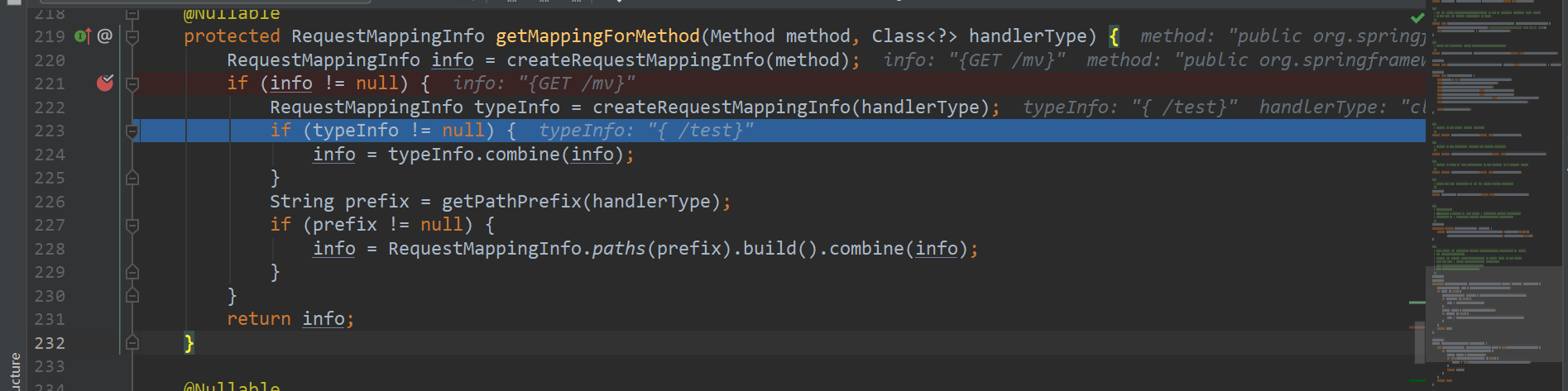
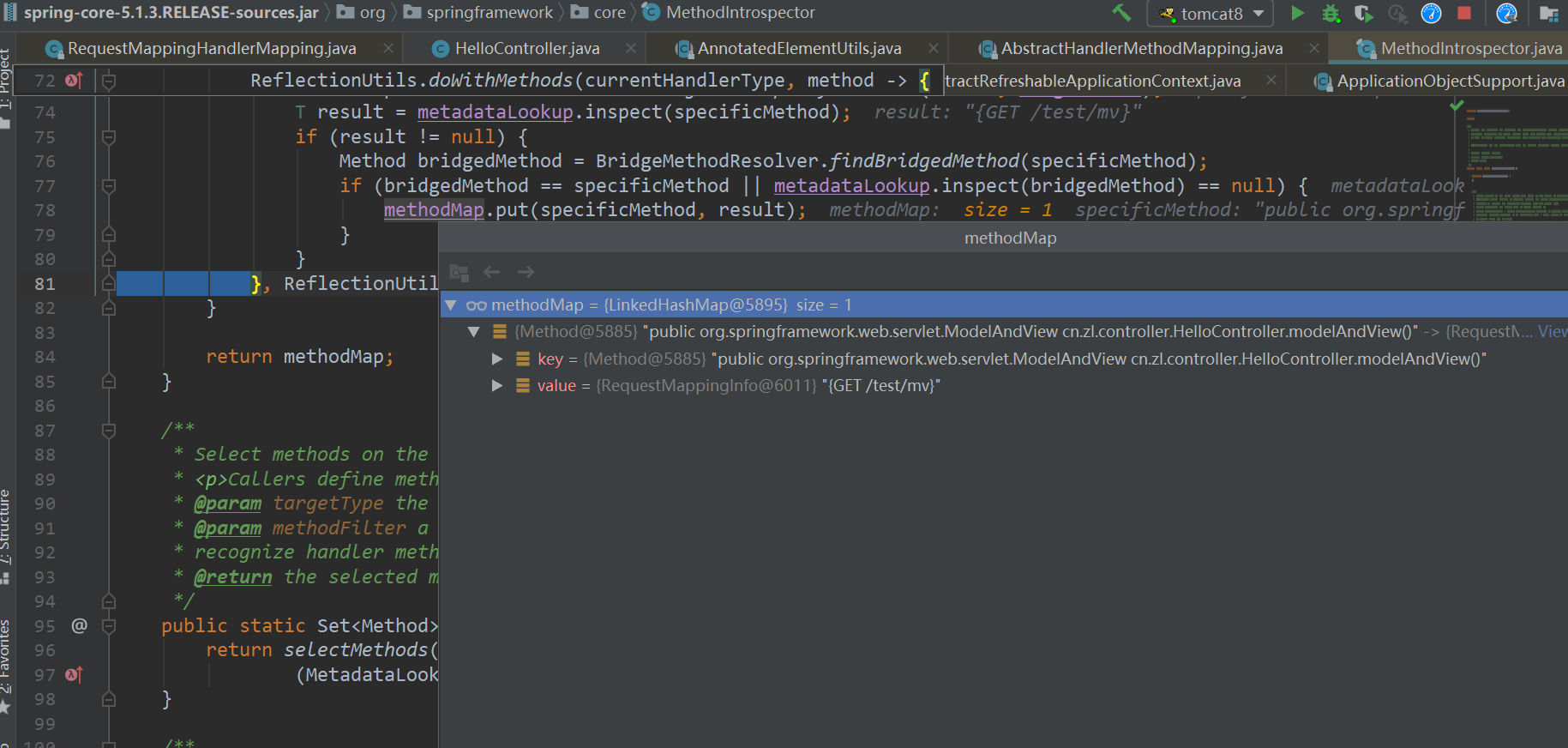
### SpringMvc扫描注解并注入的过程

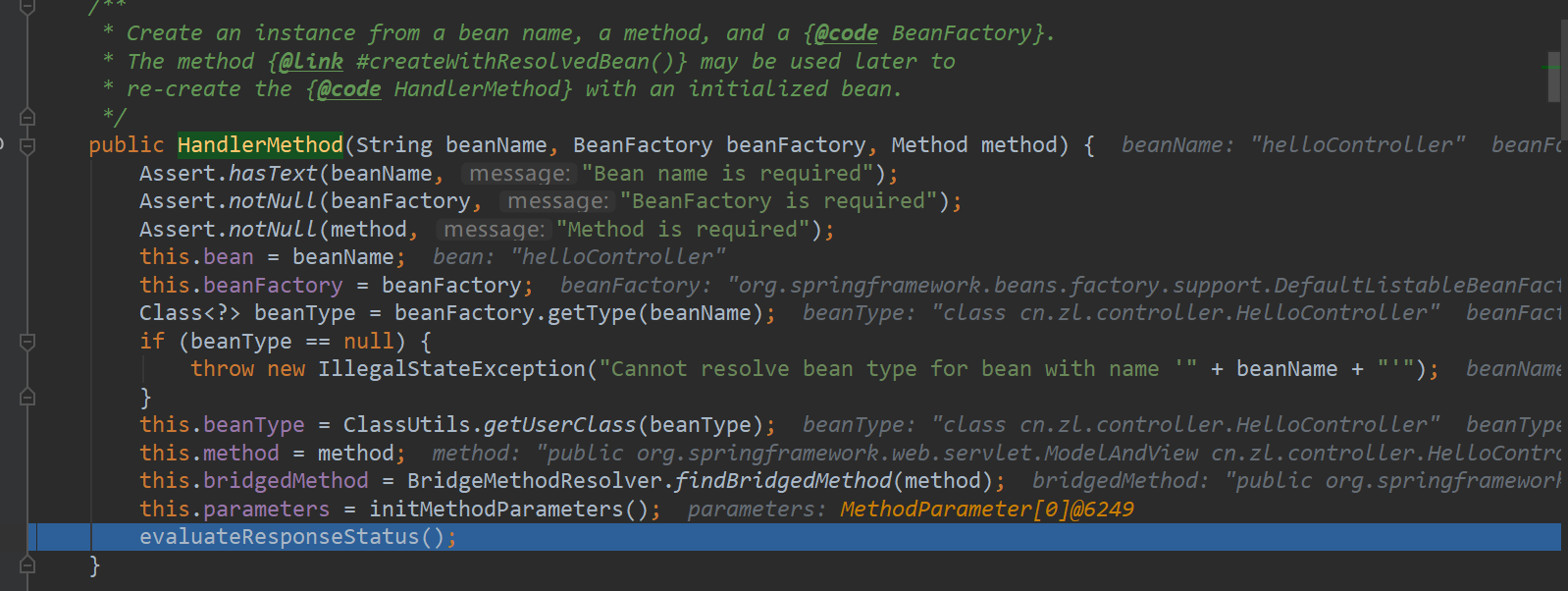
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping

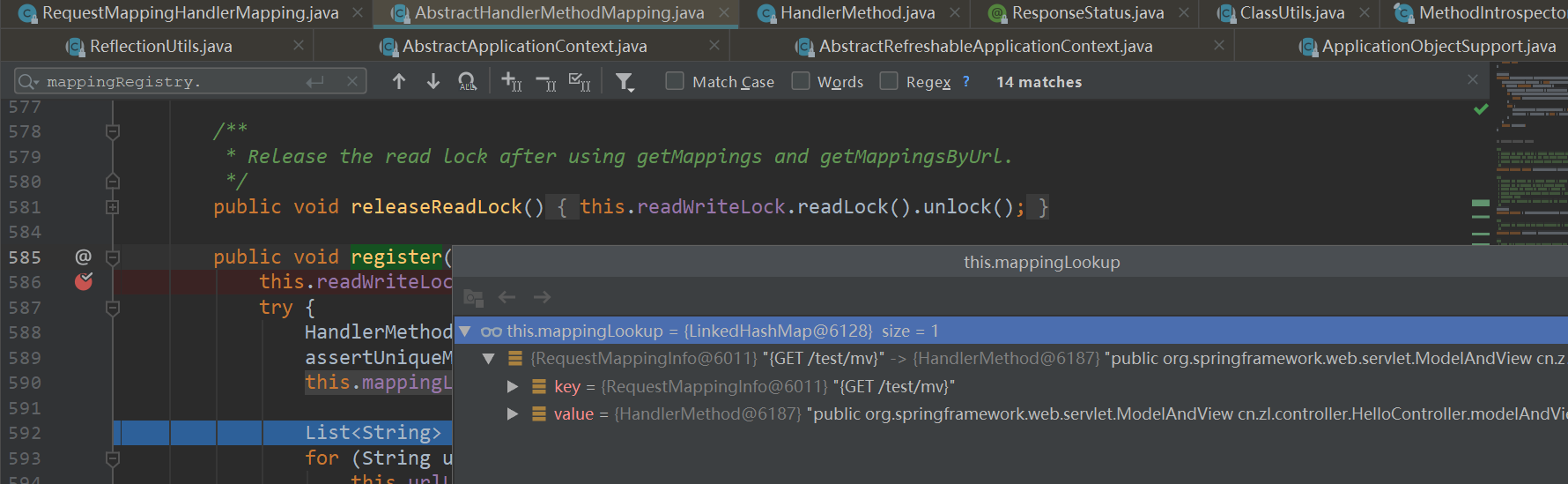
* + 1. org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#initHandlerMethods
       1. Scan beans in the ApplicationContext, detect and register handler methods.
          1. 在Application Context中扫描Bean，检测并注册handler方法
       2. org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#processCandidateBean 处理候选bean
          1. Determine the type of the specified candidate bean and call {@link #detectHandlerMethods} if identified as a handler type. <p>This implementation avoids bean creation through checking {@link org.springframework.beans.factory.BeanFactory#getType} and calling {@link #detectHandlerMethods} with the bean name.
          2. 检测指定候选bean的类型且确定是一个处理类型则调用{@link #detectHandlerMethods}.
          3. 此实现通过检查{@link org.springframework.beans.factory.BeanFactory＃getType}并使用bean名称调用{@link #detectHandlerMethods}来避免创建bean。
    2. org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping#createRequestMappingInfo(java.lang.reflect.AnnotatedElement)
       1. Delegates to {@link #createRequestMappingInfo(RequestMapping, RequestCondition)}, supplying the appropriate custom {@link RequestCondition} depending on whether the supplied {@code annotatedElement} is a class or method.
          1. 

当前Controller所有的方法

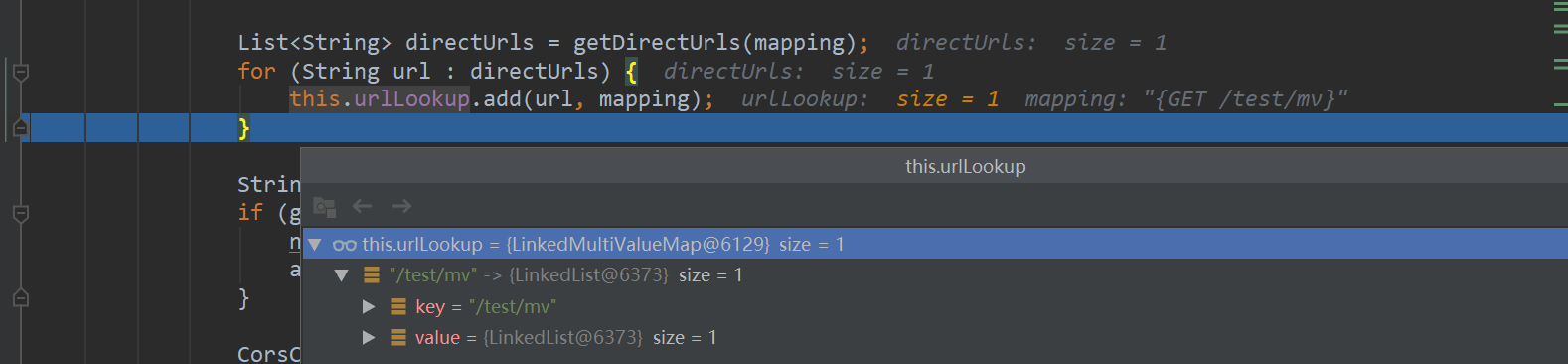
委托{@link #createRequestMappingInfo（RequestMapping，RequestCondition）}，提供适当的自定义{@link RequestCondition}，具体取决于提供的{@code annotatedElement}是类还是方法。

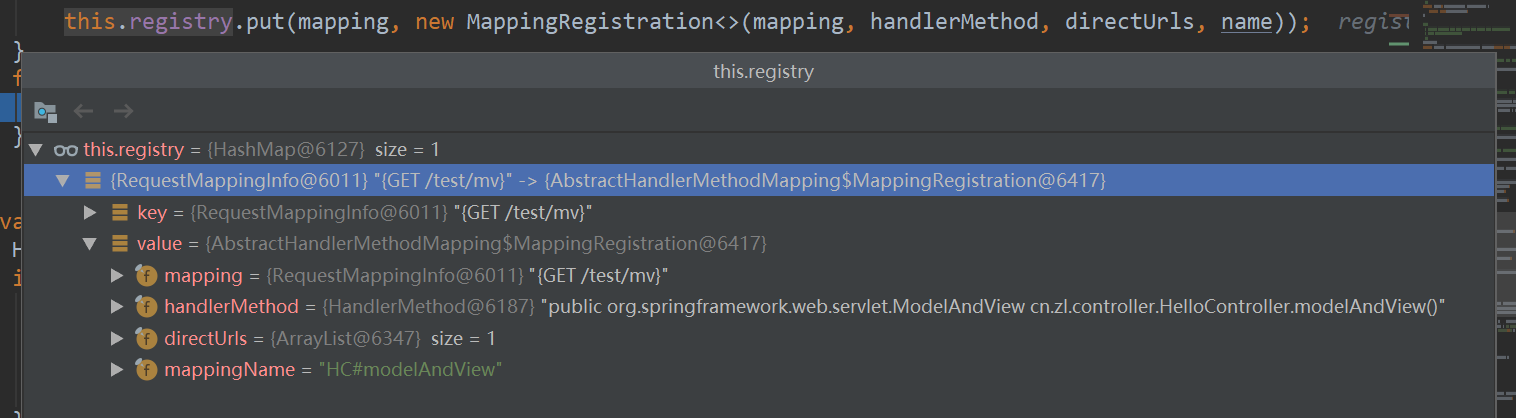
* + - * 1. 
        2. 
      1. 通过LinkedHashMap来存储Controller方法名称和请求类型以及访问路径
    1. org.springframework.web.method.HandlerMethod#HandlerMethod(java.lang.String, org.springframework.beans.factory.BeanFactory, java.lang.reflect.Method)
    2. Create an instance from a bean name, a method, and a {@code BeanFactory}. The method {@link #createWithResolvedBean()} may be used later to re-create the {@code HandlerMethod} with an initialized bean.
       1. 从bean名称，方法和{@code BeanFactory}创建实例。 稍后可以使用方法{@link #createWithResolvedBean()}来使用初始化的bean重新创建{@code HandlerMethod}。

构建一个类的过程

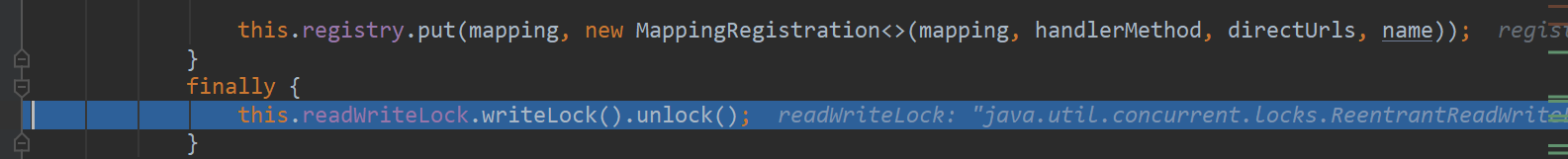
* + 1. org.springframework.web.method.HandlerMethod#evaluateResponseStatus
    2. org.springframework.web.servlet.handler.AbstractHandlerMethodMapping.MappingRegistry#register
    3. 

Controller方法的注解和方法的名称也是k-v，LinkedHashMap存储

* + 1. 最终路径映射如下
    2. 最终注册的HashMap的存储结果



值得一提的是，spring在此处使用重入锁避免多线程竞争



## CORS[跨域访问]

### org.springframework.web.cors.CorsUtils#isCorsRequest



* + 1. <https://www.jianshu.com/p/f9c21da2c661> CORS 跨源请求

Access-Control-Allow-Origin

这个头部信息由服务器返回，用来明确指定那些客户端的域名允许访问这个资源。它的值可以是：

* \* —— 允许任意域名
* 一个完整的域名名字（比如：<https://example.com）>

Access-Control-Allow-Credentials

这个头部信息只会在服务器支持通过cookies传递验证信息的返回数据里。它的值只有一个就是 true。跨站点带验证信息时，服务器必须要争取设置这个值，服务器才能获取到用户的cookie。

Access-Control-Allow-Headers

提供一个逗号分隔的列表表示服务器支持的请求数据类型。假如你使用自定义头部(比如：x-authentication-token 服务器需要在返回OPTIONS请求时，要把这个值放到这个头部里，否则请求会被阻止)。

Access-Control-Expose-Headers

相似的，这个返回信息里包含了一组头部信息，这些信息表示那些客户端可以使用。其他没有在里面的头部信息将会被限制（译者注：这个头信息实战中使用较少）。

Access-Control-Allow-Methods

一个逗号分隔的列表，表明服务器支持的请求类型（比如：GET, POST）

Origin

这个头部信息，属于请求数据的一部分。这个值表明这个请求是从浏览器打开的哪个域名下发出的。出于安全原因，浏览器不允许你修改这个值。

* + - Java中org.springframework.web.servlet.handler.AbstractHandlerMapping#getCorsHandlerExecutionChain有对跨源访问进行判断，在CorsUtils.isPreFlightRequest(request)中，判断请求头是否为Origin

1. 同源策略：同源指的是域名（或IP），协议，端口都相同，不同源的客户端脚本(javascript、ActionScript)在没明确授权的情况下，不能读写对方的资源。<https://www.cnblogs.com/morethink/p/6525216.html>
   1. 跨域情况示例

|  |  |  |
| --- | --- | --- |
| URL | 解释 | 是否跨域 |
| http://www.morethink.cn | 原URL |  |
| http://www.image.morethink.cn | 子域名 | 跨域(cookie也无法访问) |
| http://morethink.cn | 不加www | 跨域 |
| https://www.morethink.cn | 更改协议 | 跨域 |
| http://www.morethink.cn:8080 | 更改端口号 | 跨域 |

1. 同源政策的目的，是为了保证用户信息的安全，防止恶意的网站窃取数据。  
    设想这样一种情况：A网站是一家银行，用户登录以后，又去浏览其他网站。如果其他网站可以读取A网站的Cookie， 会发生什么？  
    很显然，如果Cookie包含隐私（比如存款总额），这些信息就会泄漏。更可怕的是，Cookie往往用来保存用户的登录状 态，如果用户没有退出登录，其他网站就可以冒充用户，为所欲为。因为浏览器同时还规定，提交表单不受同源政策的 限制。  
    由此可见，"同源政策"是必需的，否则 Cookie 可以共享，互联网就毫无安全可言了。
2. 同源策略限制以下几种行为：

* Cookie、LocalStorage 和 IndexDB 无法读取
* DOM 和 Js对象无法获得
* AJAX 请求不能发送

1. Java中跨院请求的配置
   1. 使用注解@CrossOrigin

|  |
| --- |
| @CrossOrigin(origins = "http://www.zhihu.com")  @RequestMapping(value = "/allProductions", method = RequestMethod.GET)  public Result getAllOldProductions() { } |
| <mvc:cors>  <mvc:mapping path="/\*\*" />  </mvc:cors> |
| 更加细粒度的配置：  <mvc:cors>  <mvc:mapping  path="/api/\*\*"  allowed-origins="http://domain1.com, [http://domain2.com"](http://domain2.com\")  allowed-methods="GET, PUT"  allowed-headers="header1, header2, header3"  exposed-headers="header1, header2"  allow-credentials="false"  max-age="123"  />  <mvc:mapping  path="/resources/\*\*"  allowed-origins="http://domain1.com"  />  </mvc:cors> |

1. 另外WebSocket以及jsonp都可以实现跨域访问

## SpringMvc执行过程详解

### org.springframework.web.servlet.DispatcherServlet#getHandler

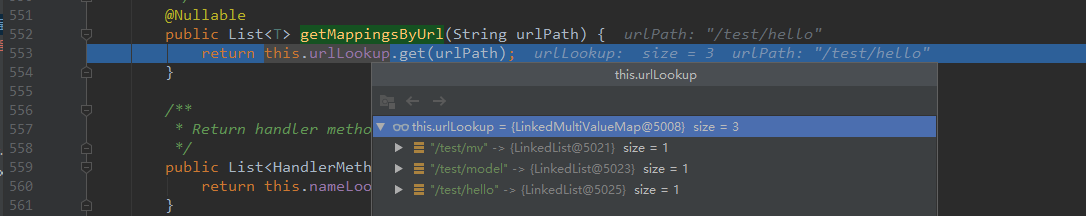
getHandler的作用是返回当前请求的HandlerExecutionChain

* + 1. org.springframework.web.servlet.handler.AbstractHandlerMapping#getHandler

查找当前请求的处理器，如果没有指定的则返回默认的处理器

* + - 1. org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#getHandlerInternal
         1. org.springframework.web.servlet.handler.AbstractHandlerMethodMapping#lookupHandlerMethod

org.springframework.web.servlet.handler.AbstractHandlerMethodMapping.MappingRegistry#getMappingsByUrl



注解扫描时填入的

org.springframework.beans.factory.support.AbstractBeanFactory#getBean(java.lang.String)

Implementation of BeanFactory interface

org.springframework.beans.factory.support.AbstractBeanFactory#doGetBean

Return an instance, which may be shared or independent, of the specified bean.

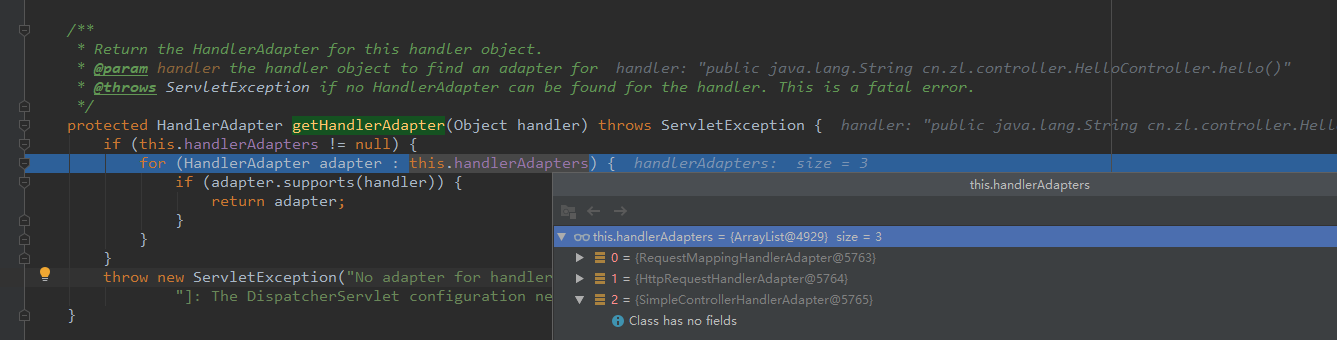
返回指定bean的实例，该实例可以是共享的或独立的。

org.springframework.web.servlet.handler.AbstractHandlerMapping#getHandlerExecutionChain

为给定的处理程序构建{@link HandlerExecutionChain}，包括适配拦截器。

org.springframework.web.servlet.HandlerExecutionChain#addInterceptor

### org.springframework.web.servlet.DispatcherServlet#getHandlerAdapter



### org.springframework.web.servlet.HandlerExecutionChain#applyPreHandle

前置处理

* + 1. org.springframework.web.servlet.HandlerExecutionChain#getInterceptors

Return the array of interceptors to apply (in the given order).给定的顺序

* 1. org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter#handle
     1. org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter#handleInternal
     2. org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter#invokeHandlerMethod

Invoke the {@link RequestMapping} handler method preparing a {@link ModelAndView} if view resolution is required.

如果需要查看解析，请调用{@link RequestMapping}处理程序方法准备{@link ModelAndView}。

* + - 1. org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter#getDataBinderFactory
      2. org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter#getModelFactory
      3. org.springframework.web.method.support.ModelAndViewContainer > new
    1. org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod#invokeAndHandle
    2. org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter#getModelAndView

### org.springframework.web.servlet.HandlerExecutionChain#applyPostHandle

后置处理

### org.springframework.web.servlet.DispatcherServlet#processDispatchResult

要么返回ModelAndView要么抛异常

### org.springframework.web.servlet.DispatcherServlet#render

Render the given ModelAndView.渲染给定的ModelAndView

* + 1. 通过给定请求解析当前区域设置。有默认的区域设置
    2. 设置区域编码
    3. 创建View并渲染
    4. org.springframework.web.servlet.view.AbstractView#render
       1. Prepares the view given the specified model, merging it with static attributes and a RequestContext attribute, if necessary. Delegates to renderMergedOutputModel for the actual rendering.

如果需要，准备给定指定模型的视图，将其与静态属性和RequestContext属性合并。 委托renderMergedOutputModel进行实际渲染。

## 拦截器是如何加载的

### org.sf.web.servlet.config.InterceptorsBeanDefinitionParser#parse

|  |
| --- |
|  |

成功转化出xml里配置的拦截器

### org.springframework.beans.factory.xml.DefaultBeanDefinitionDocumentReader#parseBeanDefinitions

感觉加载dispatcher-servlet.xml像是分步加载，需要加载哪块才加载

* + 1. org.springframework.web.context.support.ServletContextResource#getInputStream
    2. org.springframework.beans.factory.xml.DefaultDocumentLoader#loadDocument
    3. org.springframework.beans.factory.xml.DefaultDocumentLoader#createDocumentBuilderFactory
    4. Still in startup registration phase
    5. org.springframework.beans.factory.config.BeanDefinitionHolder#BeanDefinitionHolder(org.springframework.beans.factory.config.BeanDefinition, java.lang.String, java.lang.String[])

实例化BeanDefinitionHolder时将拦截器和拦截器的BeanName赋值对应

|  |
| --- |
|  |

* + 1. org.springframework.beans.factory.parsing.CompositeComponentDefinition#addNestedComponent组件注册

|  |
| --- |
|  |

* + 1. org.springframework.beans.factory.parsing.ReaderContext#fireComponentRegistered 触发组件注册事件

## 如何让不同的拦截器按指定的顺序加载拦截

### 目前的顺序

|  |
| --- |
|  |
|  |
|  |
| 结果： |

### 修改一下xml里配置的顺序

|  |
| --- |
|  |
|  |
|  |
| 结果：拦截器的顺序改变 |

### 总结

如何让不同的拦截器按指定的顺序加载拦截？

修改xml里配置的顺序就可以了，最上边的先加载