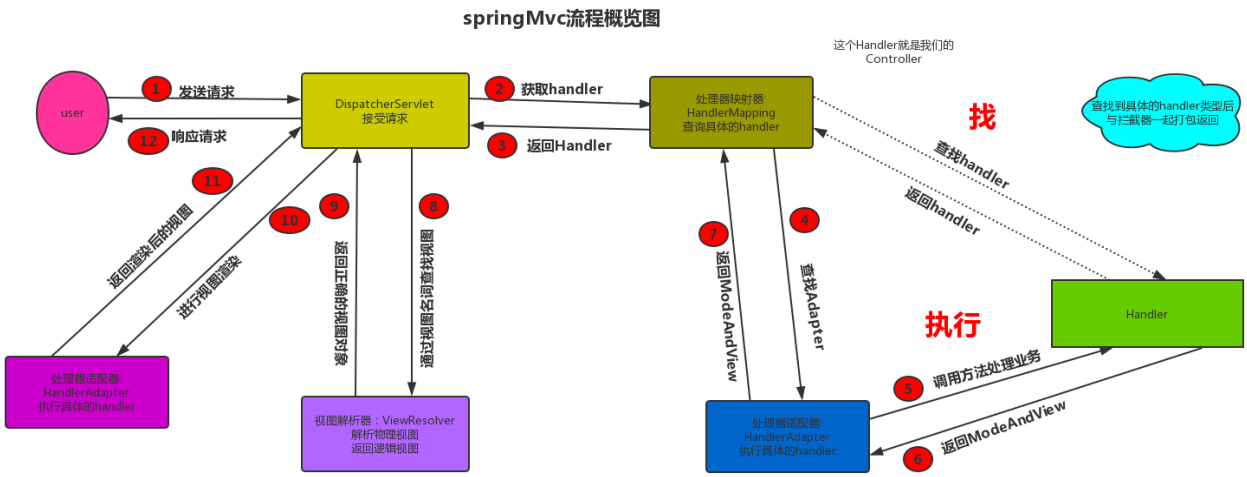
从源码的角度来看SpringMVC

./mvnw clean install -DskipTests -Pfast

SpringMVC 核心流程

先上图:SpringMVC核心流程图



简单总结:

首先请求进入DispatcherServlet 由DispatcherServlet 从HandlerMappings中提取对应的Handler

此时只是获取到了对应的Handle，然后得去寻找对应的适配器，即：HandlerAdapter

拿到对应HandlerAdapter时，这时候开始调用对应的Handler处理业务逻辑了（这时候实际上已经执行完了我们的Controller） 执行完成之后返回一个ModeAndView

这时候交给我们的ViewResolver通过视图名称查找出对应的视图然后返回

最后 渲染视图 返回渲染后的视图 -->响应请求

SpringMVC 源码解析

静态块初始化 DispacterServlet.java

//

```
try {
   ClassPathResource resource = new ClassPathResource(DEFAULT_STRATEGIES_PATH,
DispatcherServlet.class);
   defaultStrategies = PropertiesLoaderUtils.loadProperties(resource);
}
```

tomcat初始化servlet的init方法

调用 org.springframework.web.servlet.HttpServletBean#initServletBean（servlet 的init）

调用org.springframework.web.servlet.FrameworkServlet#initWebApplicationContext

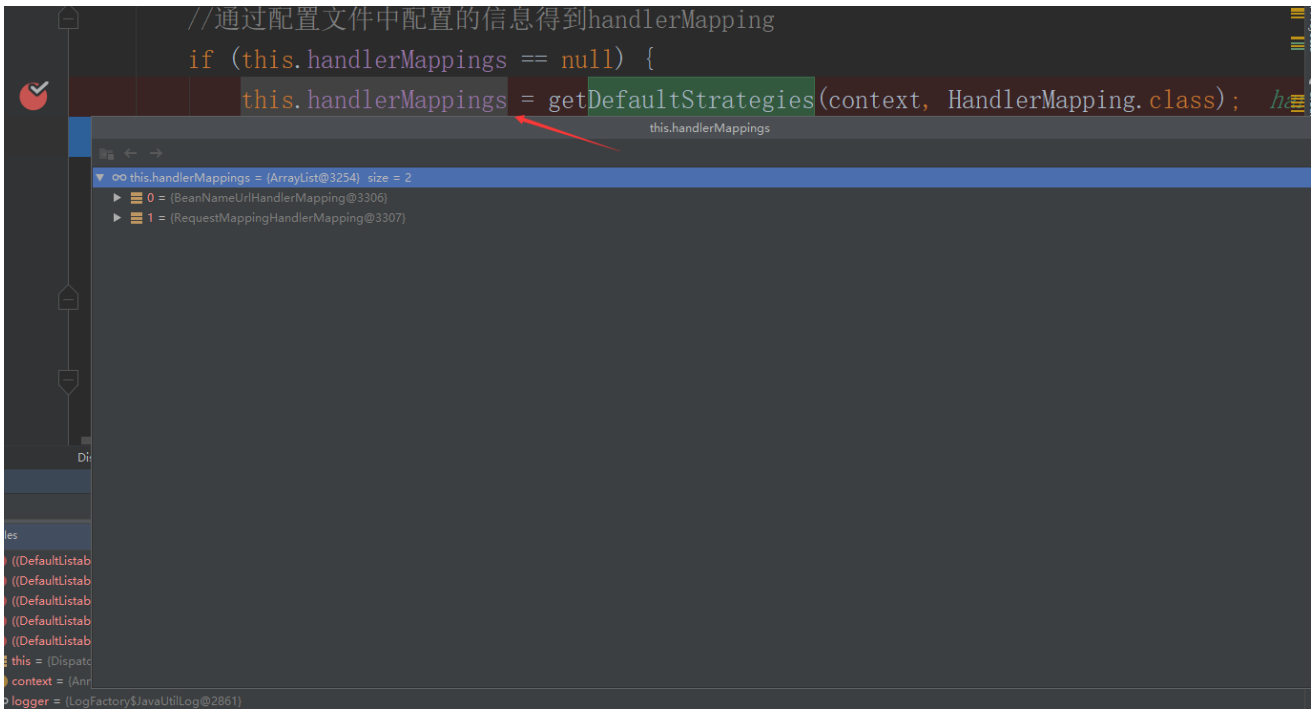
调用org.springframework.web.servlet.FrameworkServlet#onRefresh

调用org.springframework.web.servlet.DispatcherServlet#initStrategies

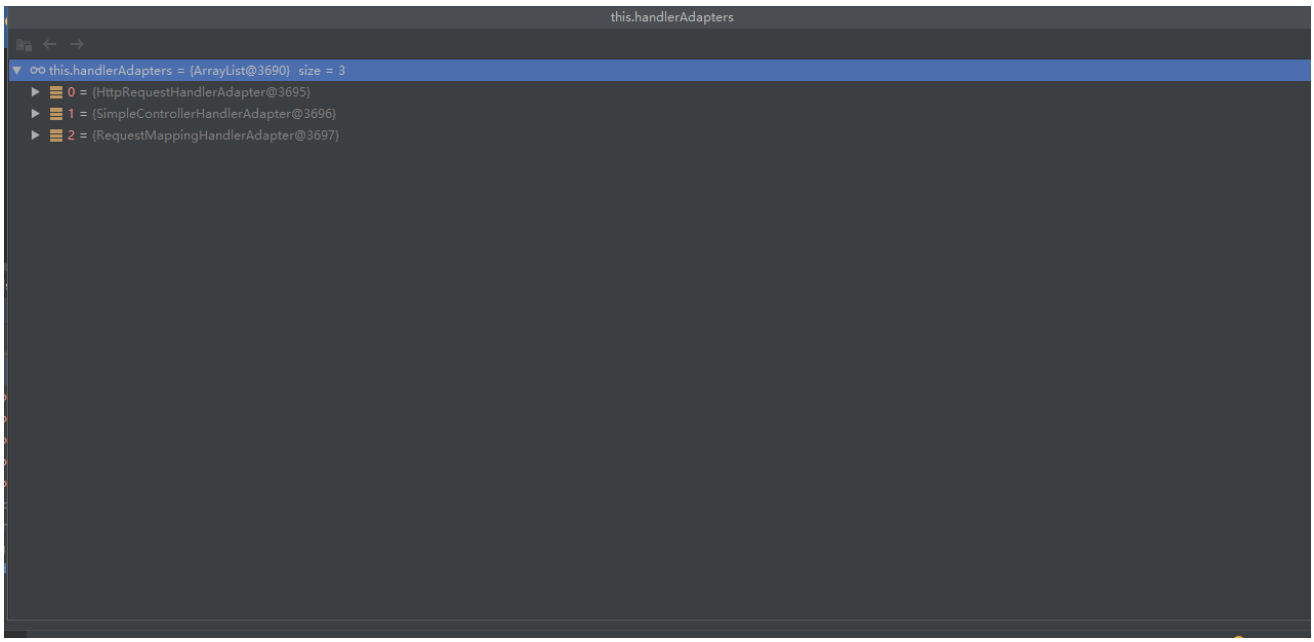
```
protected void initStrategies(ApplicationContext context) {
   initMultipartResolver(context);
   initLocaleResolver(context);
   initThemeResolver(context);
   initHandlerMappings(context);
   initHandlerAdapters(context);
   initHandlerExceptionResolvers(context);
   initRequestToViewNameTranslator(context);
   initViewResolvers(context);
   initFlashMapManager(context);
}
```

initHandlerMappings

首先从bean工厂当中得到handlerMapping继而从配置文件中去获取，配置文件一开始便被初始化了，从结果可以知道我们可以获取到两个handlerMapping，然后赋值给了`private List<HandlerMapping> handlerMappings;`



接着再初始化adapter，道理同上结果如下图



org.apache.catalina.core.StandardWrapper#instanceInitialized=true

org.springframework.web.servlet.DispatcherServlet#doDispatch

通过循环HandlerMapping来依次获取HandlerExecutionChain，因为spring当中存在的controller有多重形式，我们需要处理controller的需要通过HandlerExecutionChain来反射执行controller当中的方法，所以不同的controller需要new不同的HandlerExecutionChain，那么问题来了HandlerExecutionChain不知道你的controller是什么类型，但是HandlerMapping知道，所以HandlerExecutionChain的实例化必须依赖

`HandlerMapping`

org.springframework.web.servlet.DispatcherServlet#getHandler

获取适配器？前面说过不同的controller会获取到不同的handler，那么不同的handler他是怎么实现处理不同的controller类型呢？spring的做法比较复杂，没有从代码去解决，而是使用了适配器，故而这里根据不同的handler或得到不同的适配器从而来处理其实就是反射调用controller当中的方法

org.springframework.web.servlet.DispatcherServlet#getHandlerAdapter

`拦截器处理`

org.springframework.web.servlet.HandlerExecutionChain#applyPreHandle

org.springframework.web.servlet.HandlerAdapter#handle

org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter#handleInternal

org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter#invokeHandlerMethod

org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod#invokeAndHandle

org.springframework.web.method.support.InvocableHandlerMethod#invokeForRequest

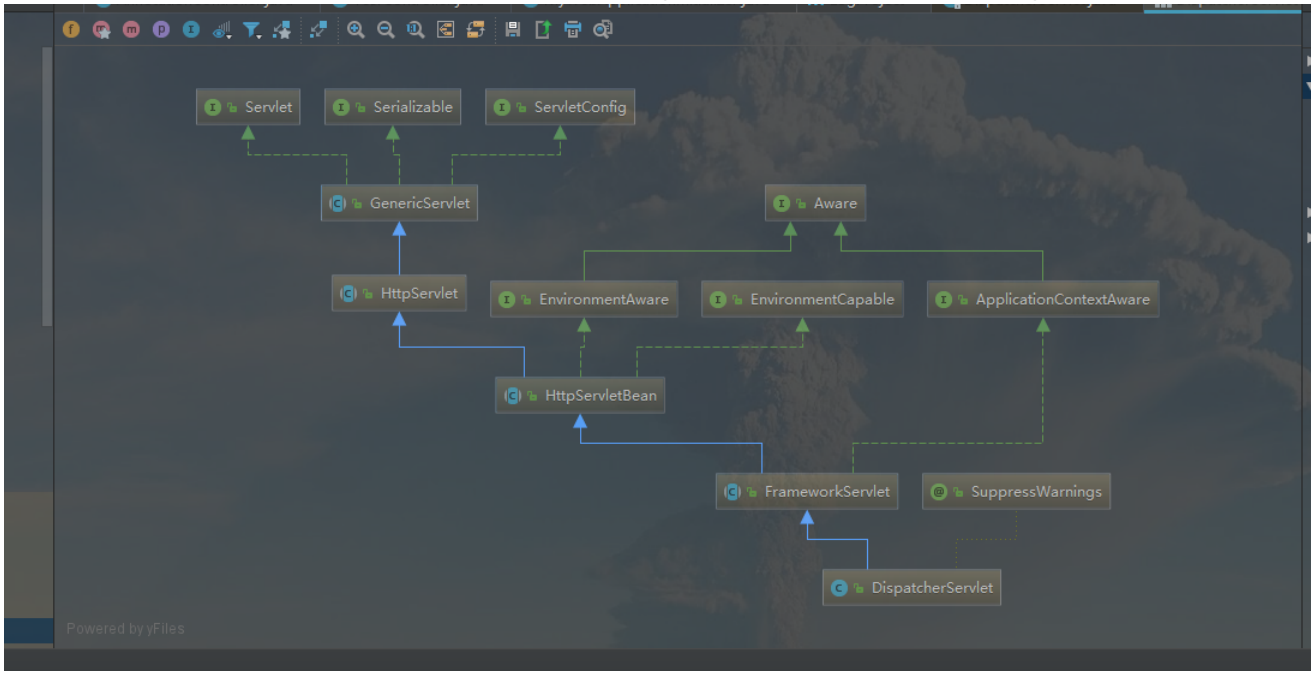
反射调用controller的方法

org.springframework.web.method.support.InvocableHandlerMethod#doInvoke

handleReturnValue

writeWithMessageConverters

首先我们查看继承关系(关键查看蓝色箭头路线) 会发现DispatcherServlet无非就是一个HttpServlet



由此，我们可以去查看Servlet的关键方法：service，doGet，doPost

service：

```
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        HttpMethod httpMethod = HttpMethod.resolve(request.getMethod());
        if (httpMethod == HttpMethod.PATCH || httpMethod == null) {
            processRequest(request, response);
        }
        else {
            super.service(request, response);
        }
    }
```

doGet：

```
@Override
    protected final void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        processRequest(request, response);
    }
```

doPost:

```
@Override
protected final void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

   processRequest(request, response);
}
```

这里会发现 无论是哪个方法 最后都调用了 processRequest(request, response);

我们把焦点放在这个方法上，会发现一个核心的方法：

doService(request, response);

然后会发现 这个方法貌似,呃..有点不一样:

```
    protected abstract void doService(HttpServletRequest request, HttpServletResponse
response)
            throws Exception;
```

它居然是一个抽象方法...这就得回到刚刚的继承关系中，找到他的子类了：DispatcherServlet

反正我看到这个方法的实现的时候，脑海里就浮现出4个字：**花 里 胡 哨** 。代码太多，就不放在笔记里面了，太占地方了.. 为什么这样说呢。。 因为你看完之后会发现 关键在于：

```
doDispatch(request, response);
```

是的，没看错，这一行才是关键!

我们把视角切入这个方法 (至于代码..还是不放进来了..) 总结一下：

把要用的变量都定义好:比如我们的ModelAndView以及异常..等等;

下面即将看到的是一个熟悉的陌生人(噢不，关键词)

```
processedRequest = checkMultipart(request);
```

"Multipart" 这个关键词好像在哪见过？？..让我想想..（渐渐步入了知识盲区） 哦对！在文件上传的时候！（勉强想起来了。。） 是的 其实这行代码就是判断当前请求是否是一个二进制请求（有没有带文件） 当然 这里只是提一下，并不是本文的核心内容。。。(有时间的小伙伴可以自己去了解一下)

好的，现在回到我们的主题,来看看这个方法：

```
mappedHandler = getHandler(processedRequest);
```

看过我们上面流程图的同学应该会知道他现在在干嘛。 现在来获取我们的Handler了..从哪获取呢？ 从他的HandlerMapping里面获取。

问题1：至于这个HandlerMappings 哪里来的呢

这个等下讨论 我们先来看看他获取的代码:

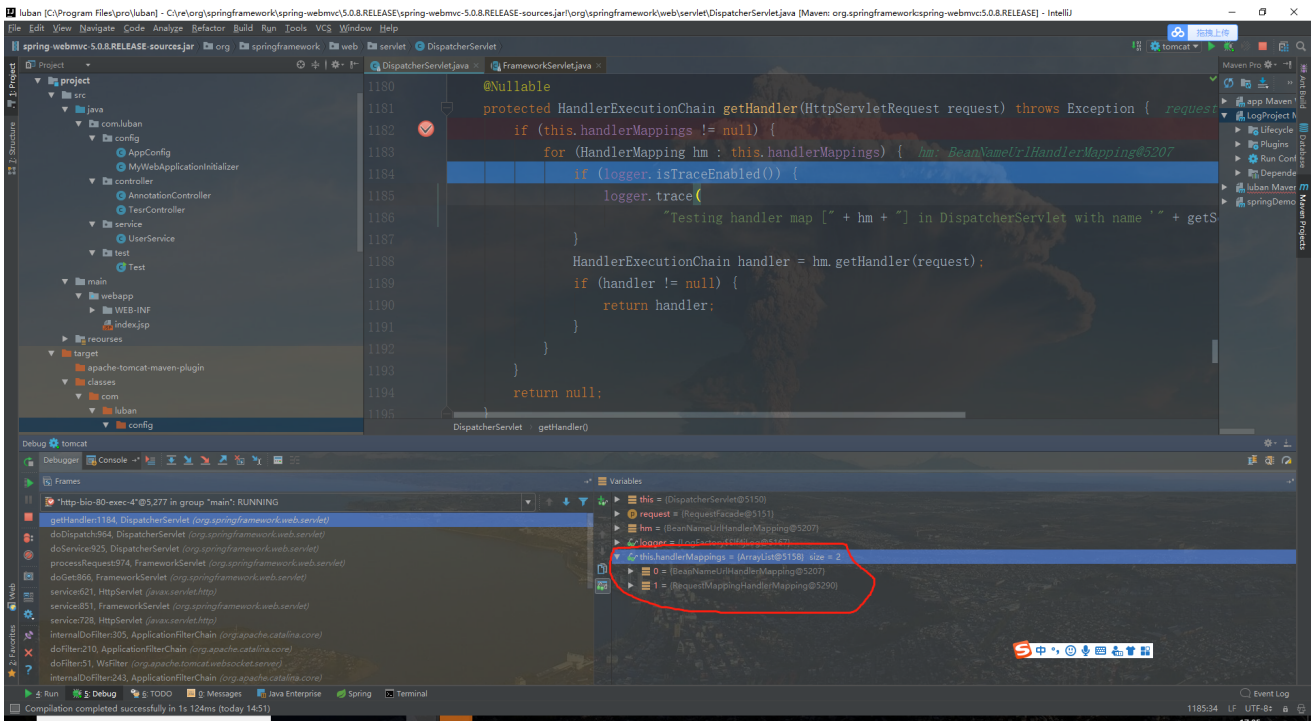
```

```

```
protected HandlerExecutionChain getHandler(HttpServletRequest request) throws Exception
{
   if (this.handlerMappings != null) {
      for (HandlerMapping hm : this.handlerMappings) {
         if (logger.isTraceEnabled()) {
            logger.trace(
                  "Testing handler map [" + hm + "] in DispatcherServlet with name '" +
getServletName() + "'");
         }
         HandlerExecutionChain handler = hm.getHandler(request);
         if (handler != null) {
            return handler;
         }
      }
   }
   return null;
}
```

我们能看见他是在遍历一个handlerMappings

问题2：至于这个handlerMapping是什么呢，

请看：

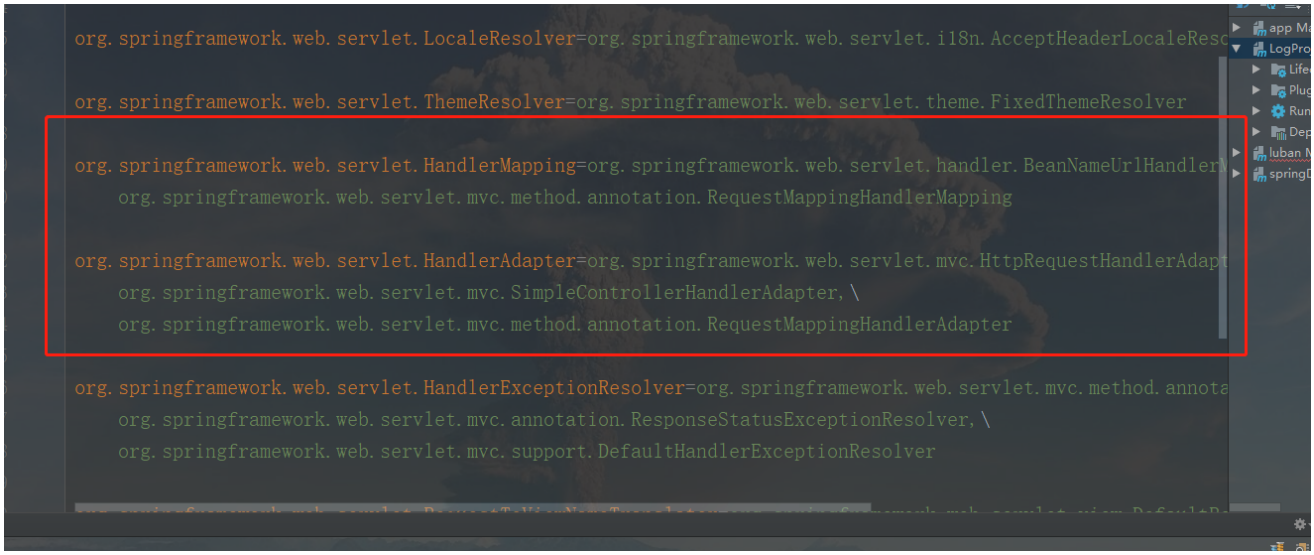


是2个我们不认识的东西，至于是什么，现在说了也不知道，我们继续往下走

可以看见图片上1188行代码， 他从这个mapping 里面获取了一个handler 如果获取到了 这个方法就走完了， 不然就下一次循环.

问题1解释:

我们先回到刚刚那个问题，这个HandlerMapping 哪里来的呢。 不多说，上图：



我们在源码包的DispatcherServlet.properties文件下会看见， 他定义了图片里的这些属性。 重点放在方框内，第一个属性，就是我们刚看见的HandlerMappings， 也就是说 HandlerMappings也就是他自己事先定义好的呢。至于第二个属性，咱们待会儿见~

也就是说SpringMVC自己自带了2个HandlerMapping 来供我们选择 至于 为什么要有2个呢？ 这时候得启动项目从断点的角度来看看了;

我们用2种方式来注册Controller 分别是:

作为Bean的形式:

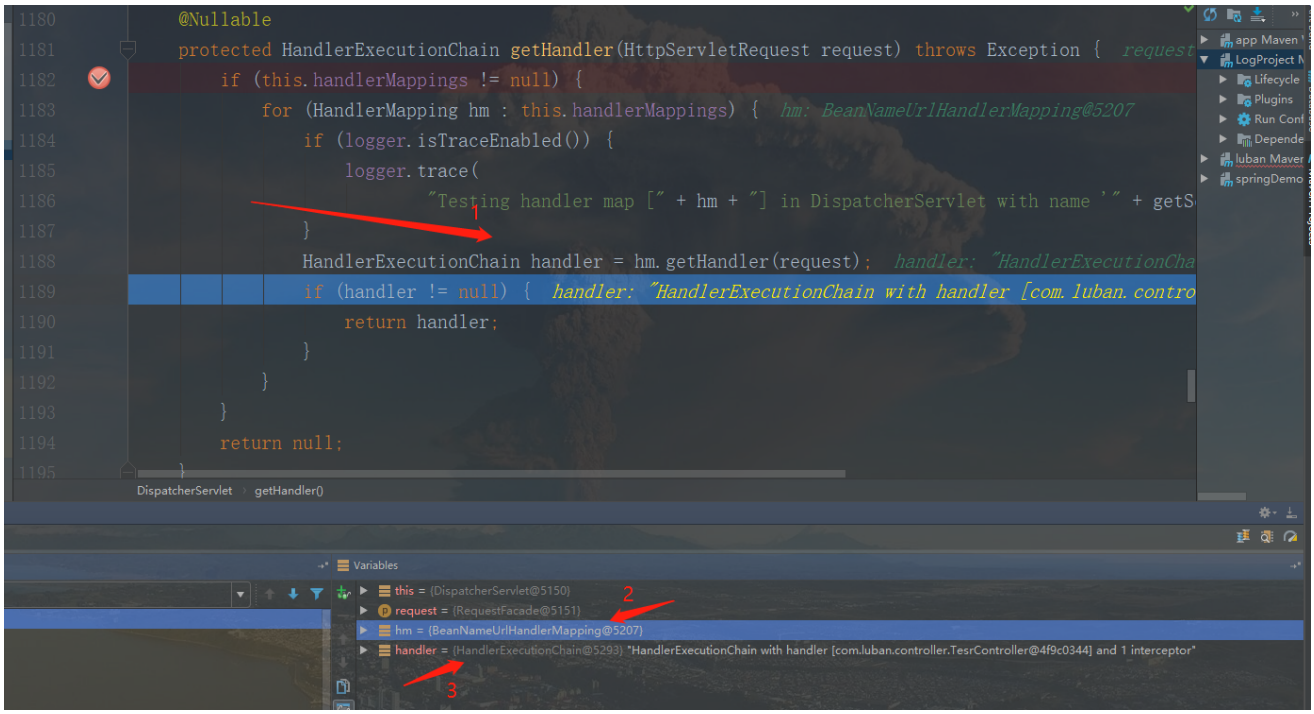
```
@Component("/test")
public class TesrController  implements org.springframework.web.servlet.mvc.Controller{


    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse
response) throws Exception {
        System.out.println("1");
        return null;
    }
}
```

以Annotation形式:

```
@Controller
public class AnnotationController {

    @RequestMapping("/test2")
    public Object test(){

        System.out.println("test");

        return null;
    }
}
```

我们先用Bean的方式来跑:

视角走到我们的mappedHandler = getHandler(processedRequest);里面



问题2解释:

来，跟着箭头走，我们发现 我们以Bean的形式注册的Controller 可以从这个BeanNameUrlHandlerMapping里面获取到对应的Handler ; 这里 我们是不是对于这个HandlerMapping有了懵懂的了解了?

猜想1:

我们来猜一下 如果是以Annotation的形式注册的Controller的话 就会被第二个HandlerMapping获取到。 至于对不对 这个问题我们先留着。

我们先让代码继续走，会发现 Handler返回出来紧接着会执行下面这个方法，这个方法我们从流程图中可以了解到，就是在找一个适配器。

```
HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());
```

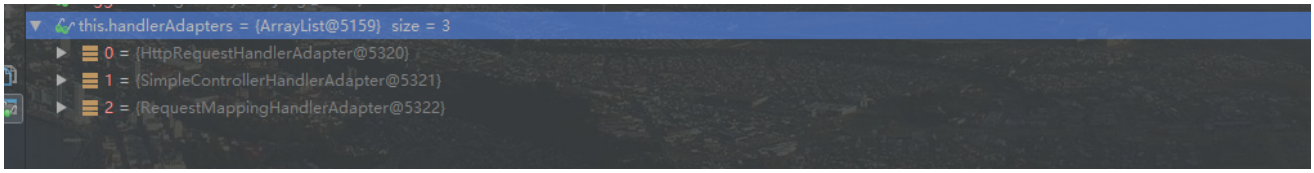
问题3：何为适配器?

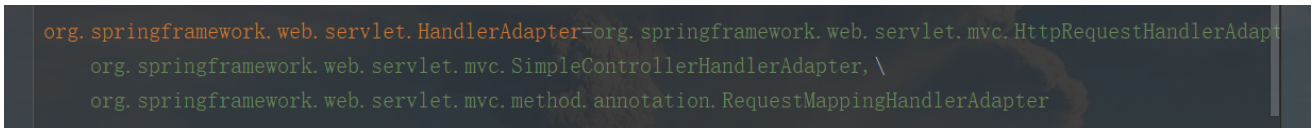
我们先来看看他这个方法里面干了啥:

```
protected HandlerAdapter getHandlerAdapter(Object handler) throws ServletException {
   if (this.handlerAdapters != null) {
      for (HandlerAdapter ha : this.handlerAdapters) {
         if (logger.isTraceEnabled()) {
            logger.trace("Testing handler adapter [" + ha + "]");
         }
         if (ha.supports(handler)) {
            return ha;
         }
      }
   }
   throw new ServletException("No adapter for handler [" + handler +
```

```
            "]: The DispatcherServlet configuration needs to include a HandlerAdapter that
supports this handler");
}
```

其实能看见他是从一个handlerAdapters属性里面遍历了我们的适配器 这个handlerAdapters哪来的呢？ 跟我们的HandlerMappings一样 在他的配置文件里面有写，就是我们刚刚所说的 待会儿见的那个东西~ 不多说 上图：





问题3解释：

至于什么是适配器，我们结合Handler来讲， 就如我们在最开始的总结时所说的， 一开始只是找到了Handler 现在要执行了，但是有个问题，Handler不止一个， 自然而然对应的执行方式就不同了， 这时候适配器的概念就出来了：对应不同的Handler的执行方案。

当找到合适的适配器的时候， 基本上就已经收尾了，因为后面在做了一些判断之后（判断请求类型之类的），就开始执行了你的Handler了，上代码：

```
mv = ha.handle(processedRequest, response, mappedHandler.getHandler());
```

这个mv就是我们的ModlAndView 其实执行完这一行 我们的Controller的逻辑已经执行完了， 剩下的就是寻找视图 渲染图的事情了....

我们这里只是使用了Bean的形式执行了一遍流程 假设使用Annotation呢?

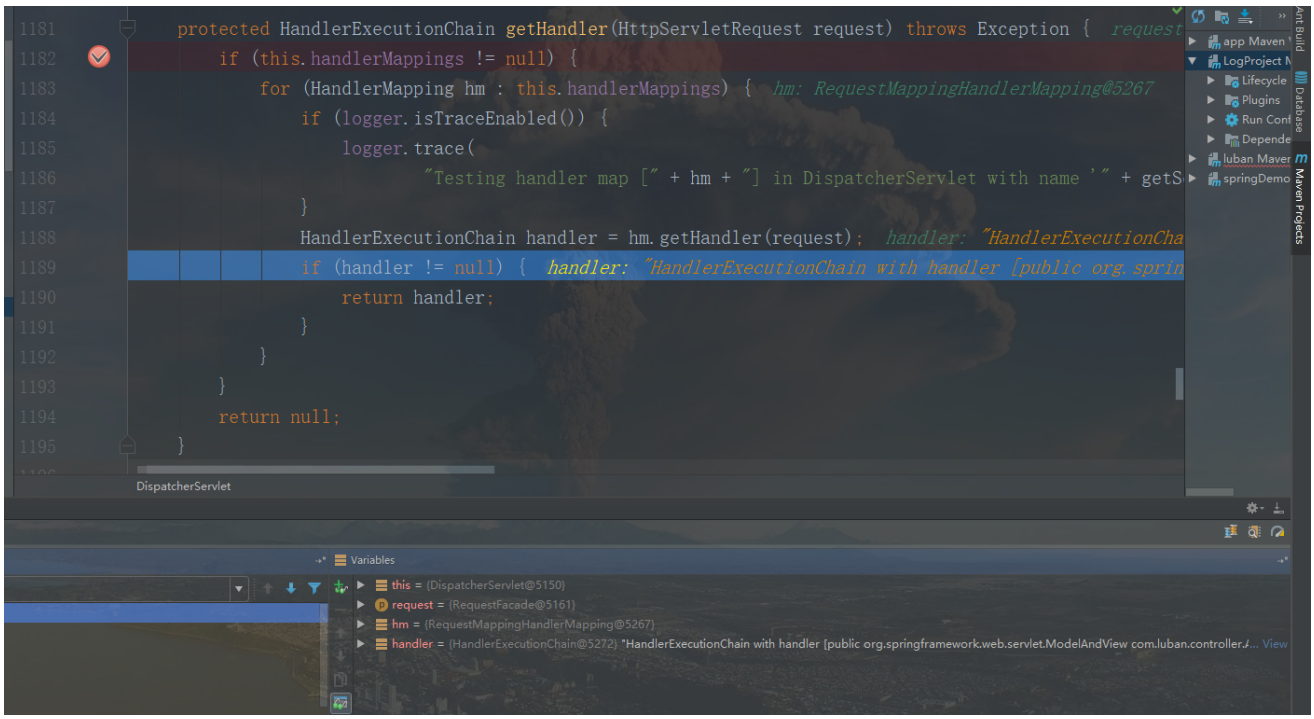
SpringMVC BeanName方式和Annotation方式注册Controller源码分析

现在我们来使用Annotation来注册Controller看看

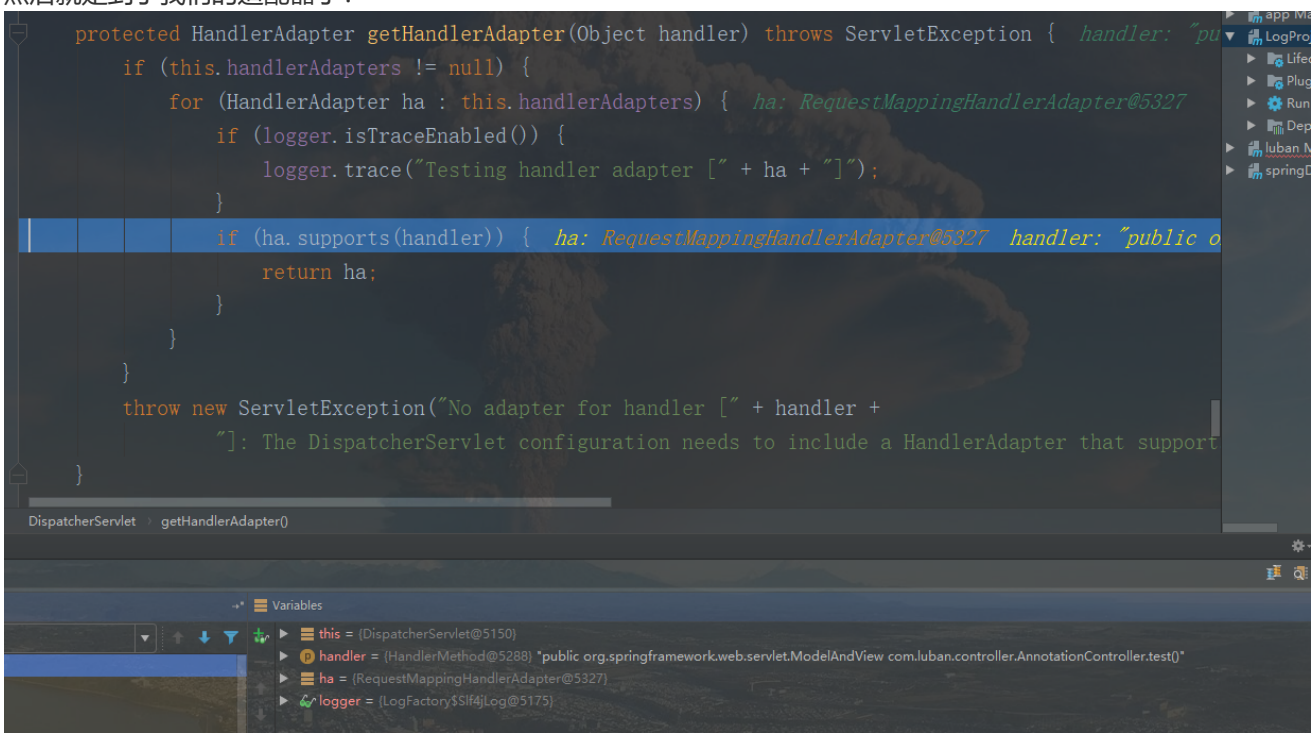
我们这里 只看不同的地方。

猜想1证明：

首先 在这个HandlerMappings这里 之前的那个就不行了 这里采用了另外一个HandlerMapping 其实也就证明了我们的猜想1



然后就是到了我们的适配器了:



这里我们会看到用的是这个适配器 而我们的Bean方式注册的Controller 的话 使用的是另外两个适配器来的，至于有什么区别呢？ 我们来看看他执行的时候:

```
@Override
@Nullable
public final ModelAndView handle(HttpServletRequest request, HttpServletResponse
response, Object handler)
      throws Exception {

   return handleInternal(request, response, (HandlerMethod) handler);
}
```

我们的Annotation的形式 是拿到这个handler作为一个HandlerMethod 也就是一个方法对象来执行 这时候我们看看Bean是什么样子的：

```
@Override
@Nullable
public ModelAndView handle(HttpServletRequest request, HttpServletResponse response,
Object handler)
      throws Exception {

   return ((Controller) handler).handleRequest(request, response);
}
```

由最开始可以看到 我们如果以Bean的形式注册Controller的话 我们的实现一个Controller的接口 在这里 他把我们的handler强制转换为一个Controller来执行了。

总结

其实我们的SpringMVC关键的概念就在于Handler（处理器） 和Adapter(适配器)

通过一个关键的HandlerMappings 找到合适处理你的Controller的Handler 然后再通过HandlerAdapters找到一个合适的HandlerAdapter 来执行Handler即Controller里面的逻辑。 最后再返回ModlAndView...