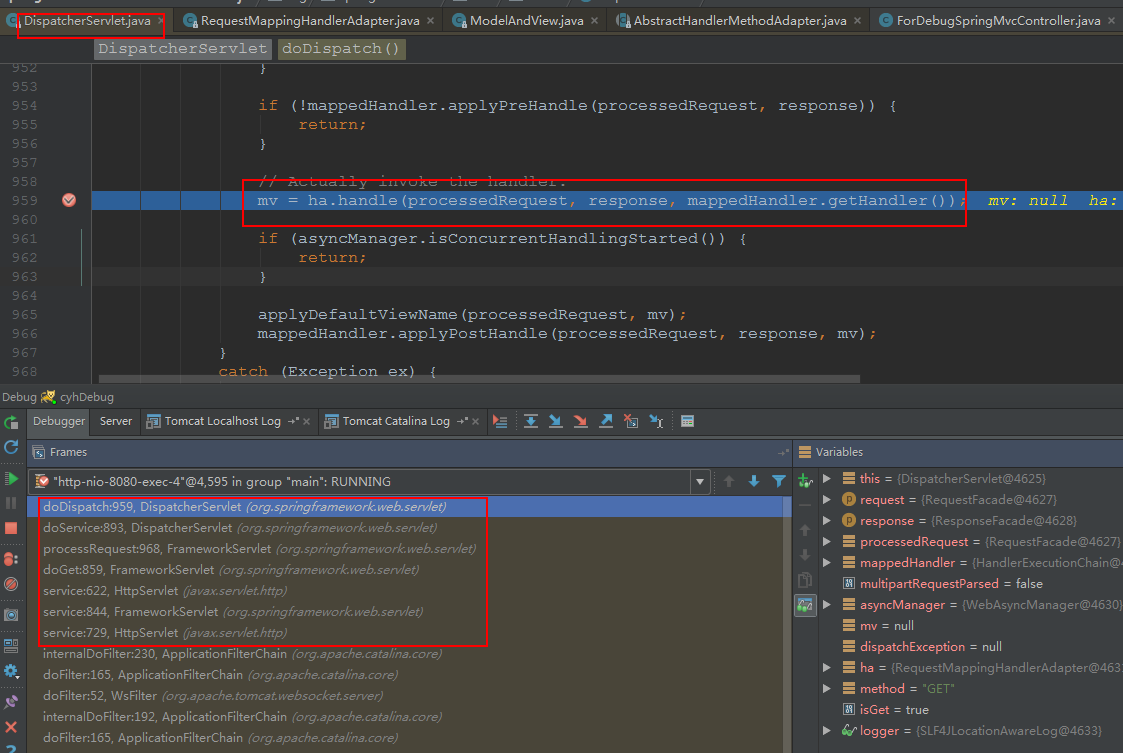
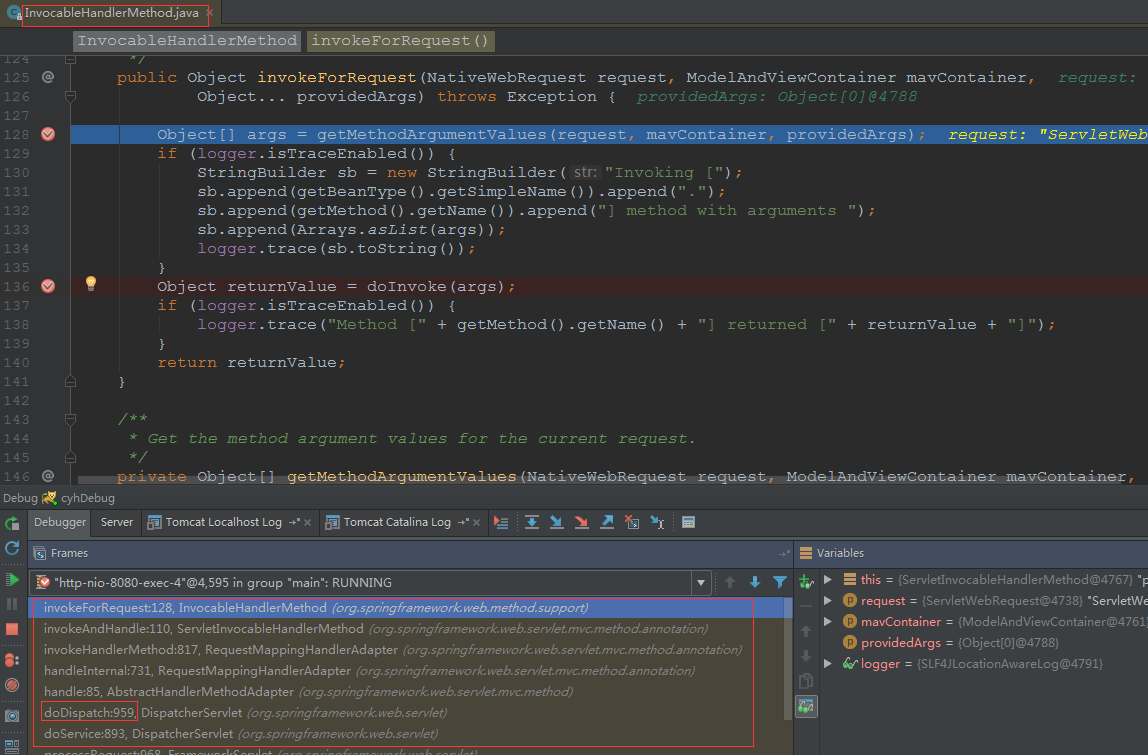
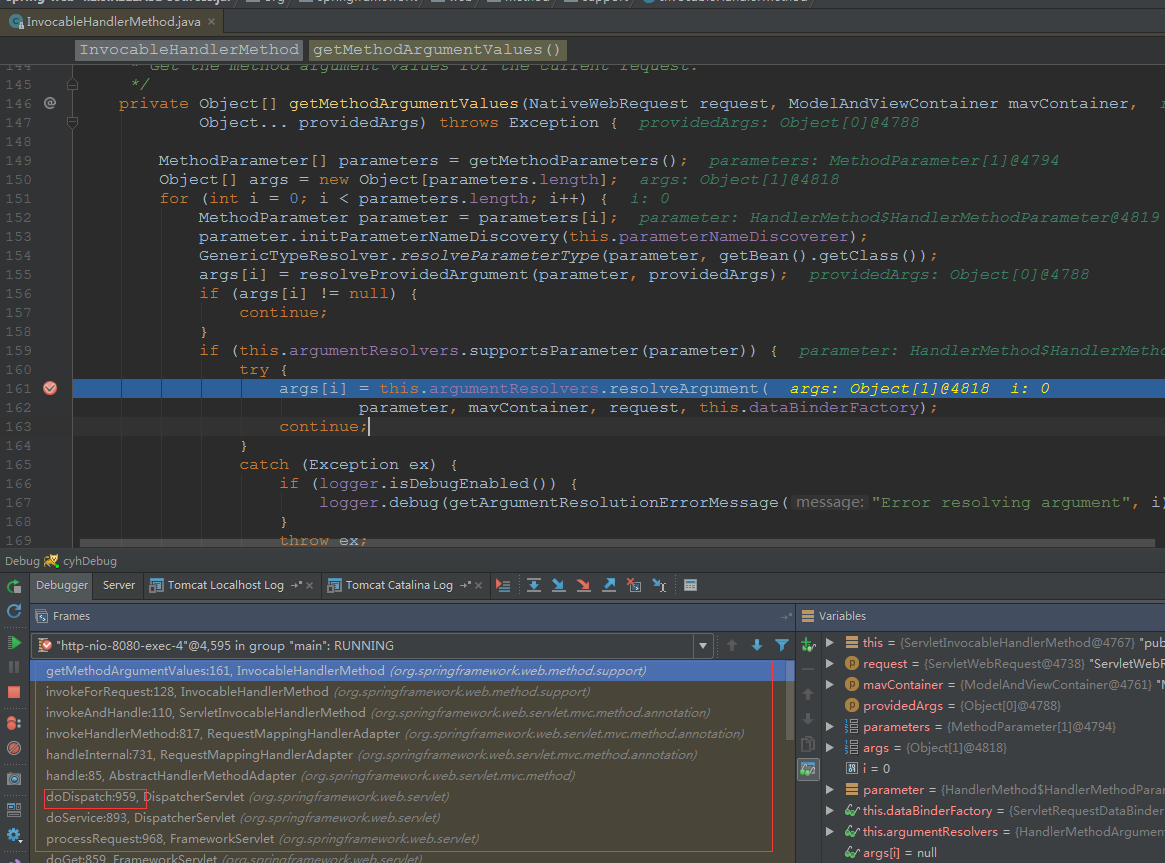
01.首先，浏览器发出的请求到DispatcherServlet；然后，找到合适的HandlerAdapter（此处是RequestMappingHandlerAdapter）；然后调用RequestMappingHandlerAdapter的handle()方法。此时，方法堆栈从Line959开始。



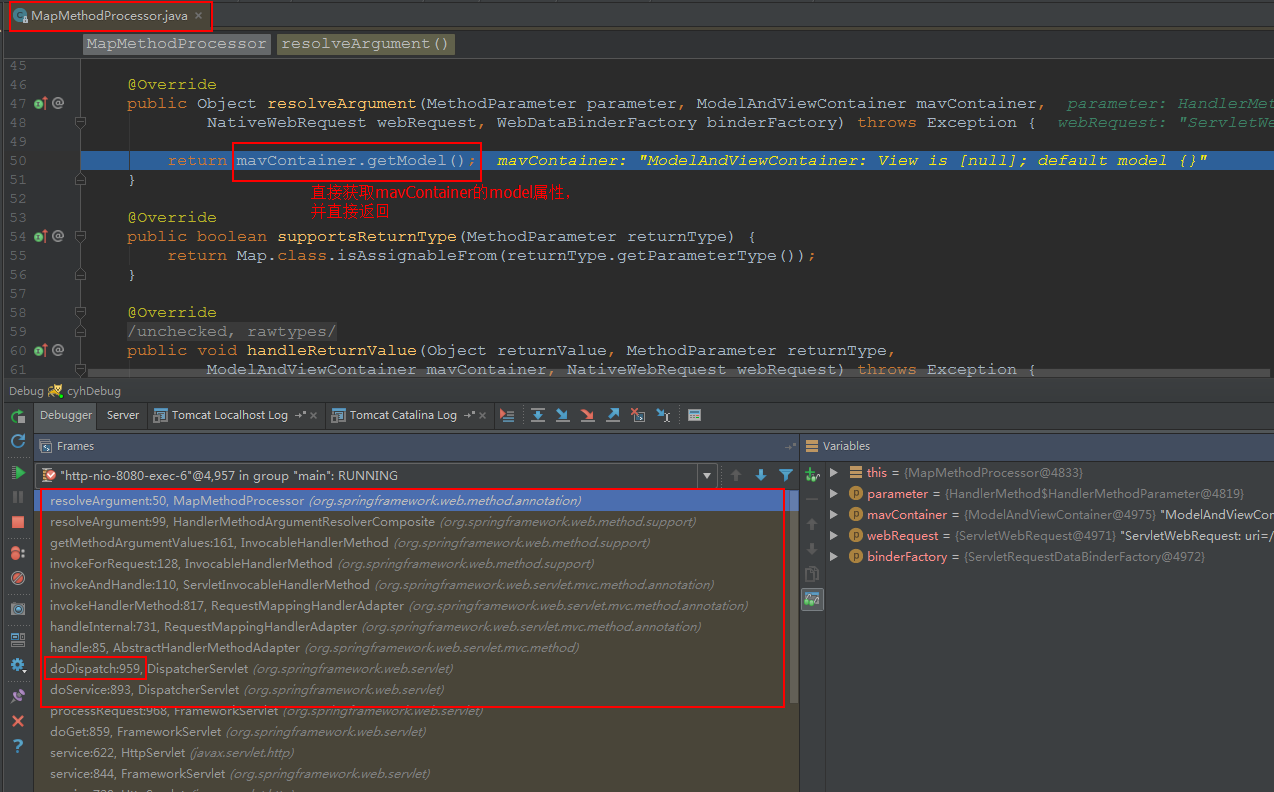
02.还是在DispatcherServlet的Line959的doDispatch()方法内，又调用了几个方法，到达invokeForRequest()方法；顾名思义，此方法会真正的调用Request方法（即Controller中的方法）；不过，先会在Line128解析参数。



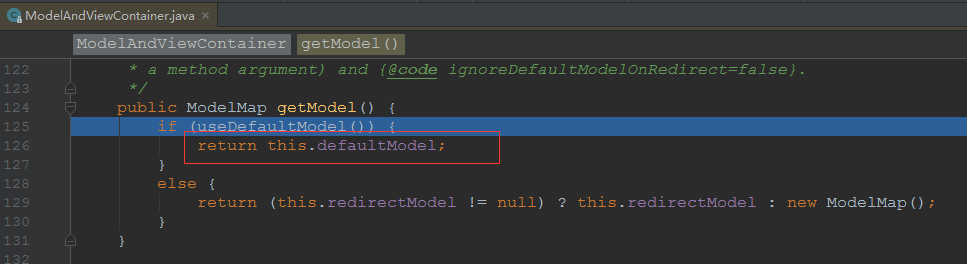
03.解析参数的方法又会走到Line161.



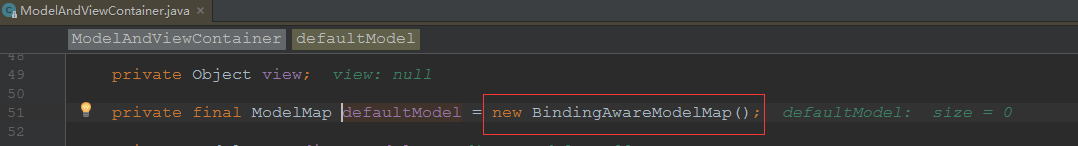
04.再经过两个方法的调用，可以看到上述的参数解析方法是直接返回了 mavContainer.getModle()



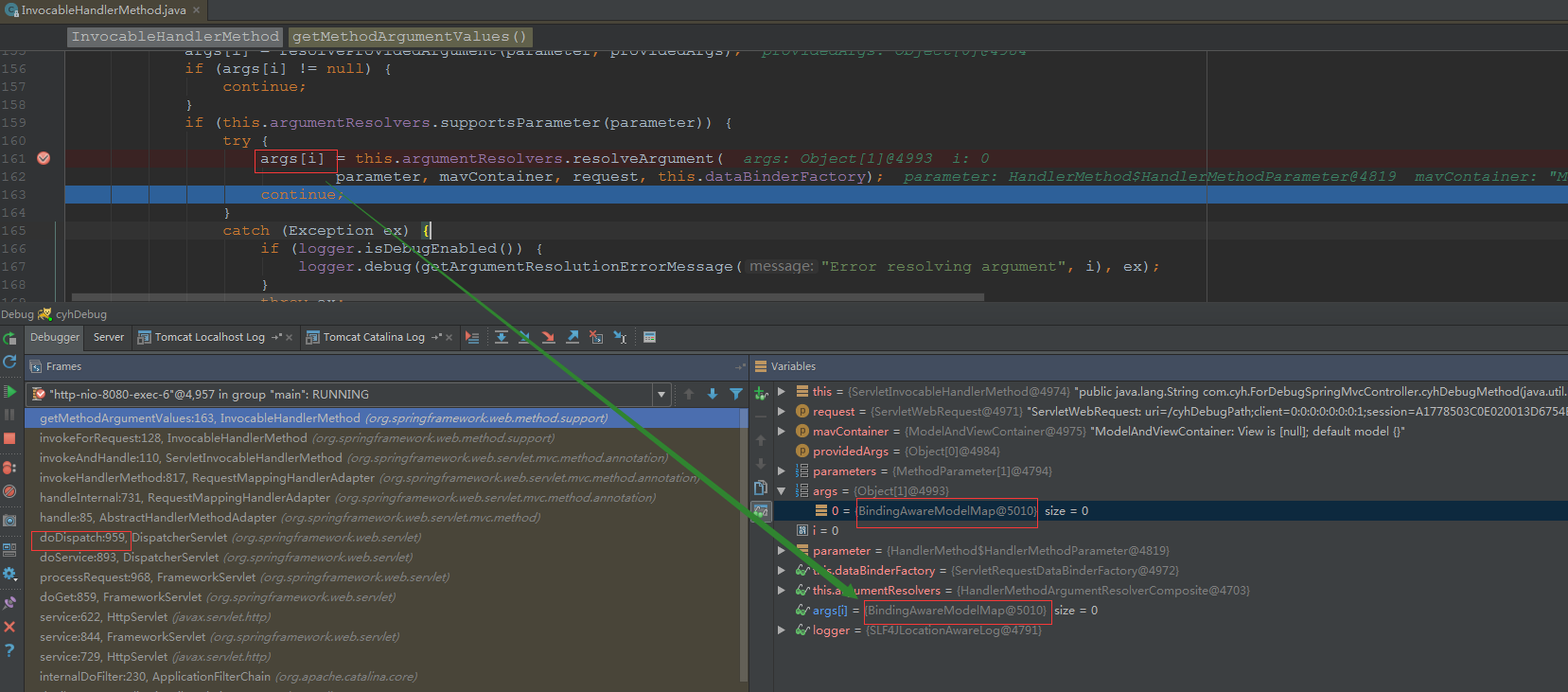
05.而getModel() 方法返回 defaulutModel 的成员变量。



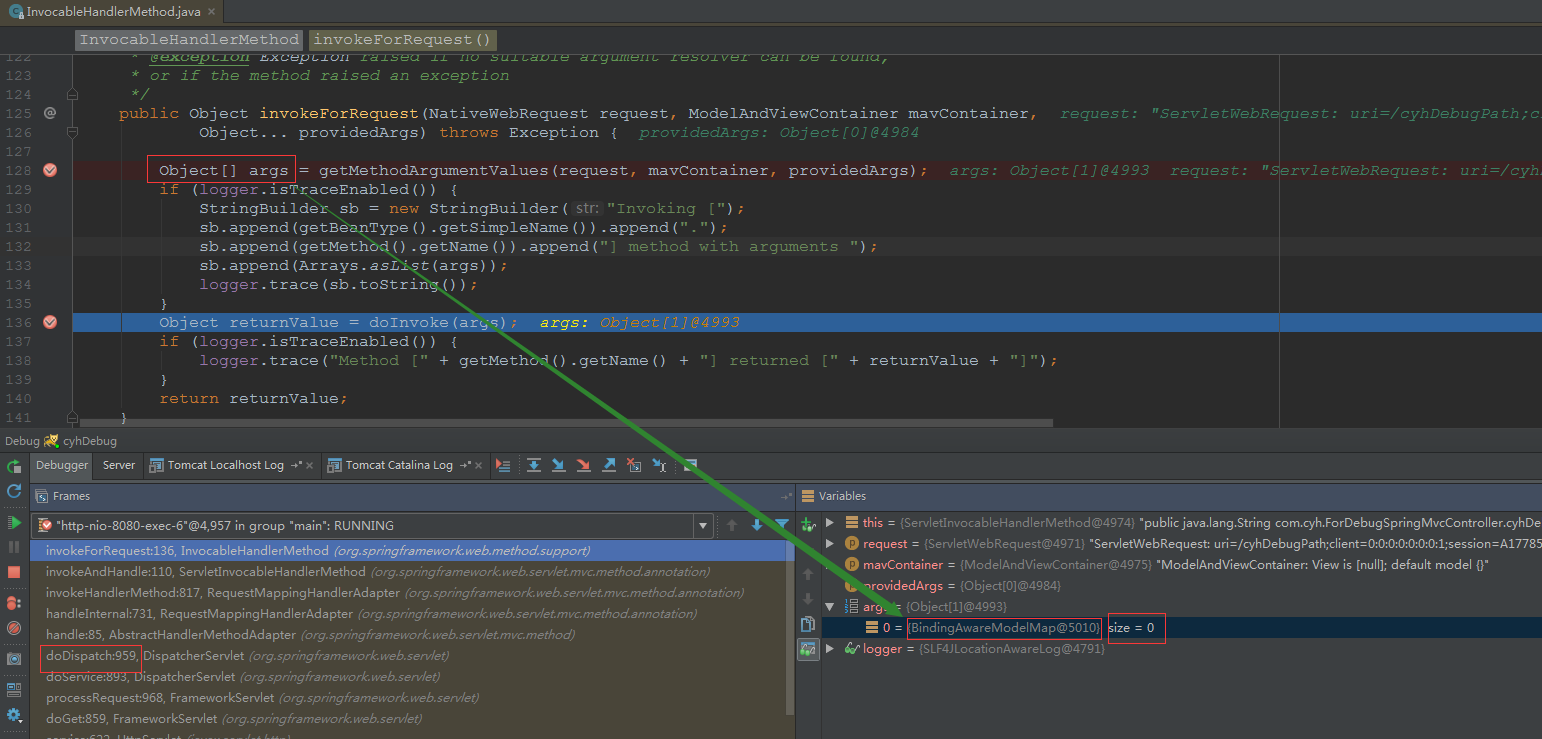
06. defaulutModel 其实就是一个 BindingAwareModelMap



07.回到刚才的Line161，可以看到 args[i] 指向了刚才得到的BindingAwareModelMap，且内存地址是5010。



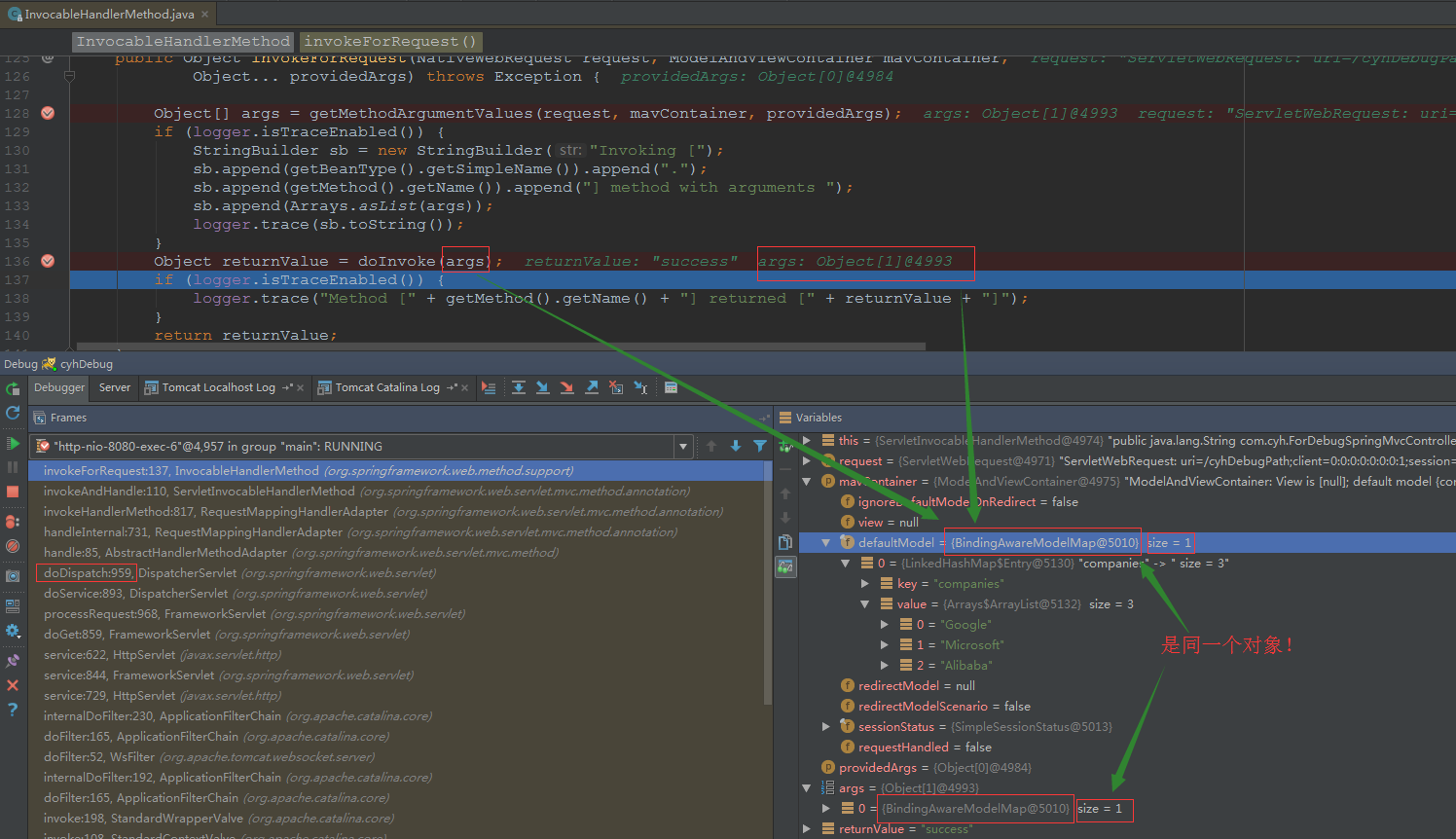
08.再返回一层，此处的 Object[] args 还是BindingAwareModelMap@5010，且其中的元素为空；此处Line136的 doInvoke(args) 方法就是通过反射调用真实的Controller中的方法。



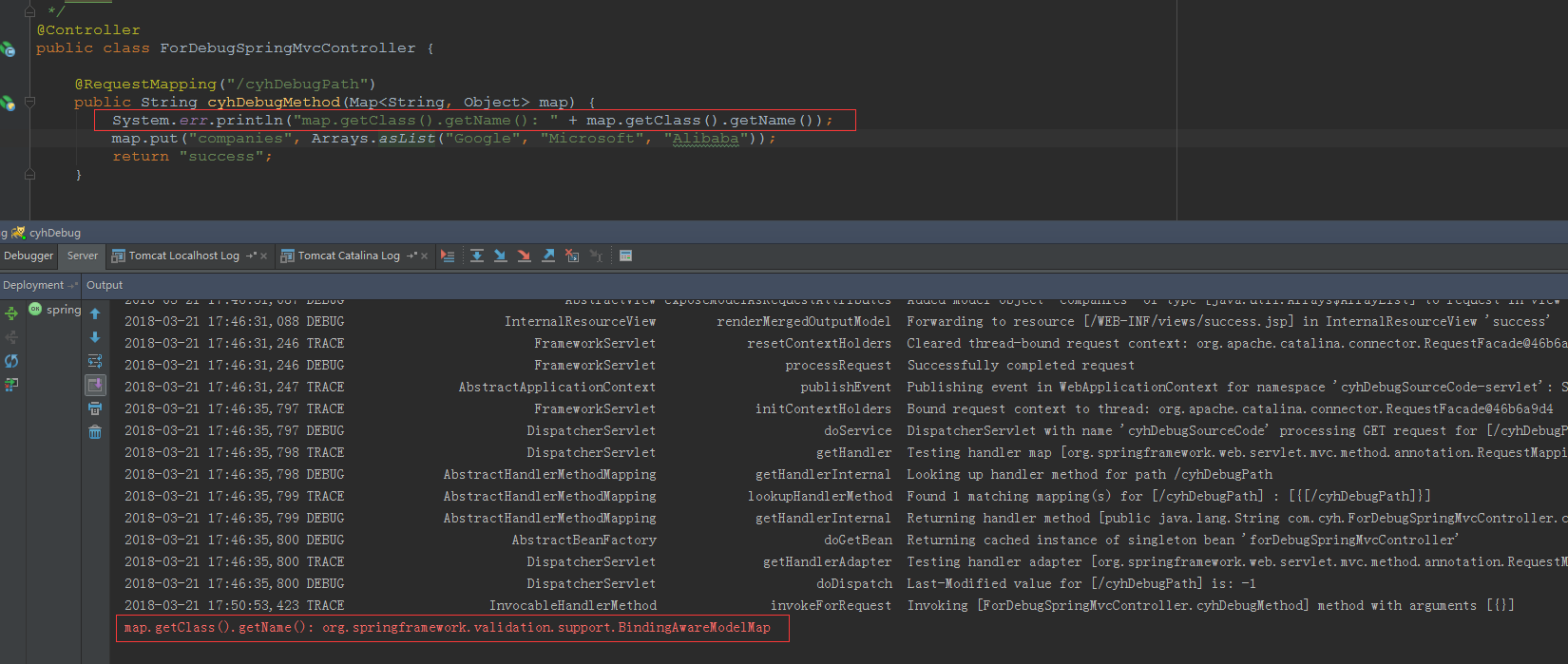
09.真实Controller方法调用返回后，可以看到 mavContainer 对象中的 defaultModel 属性已经被赋值，且这个值就是BindingAwareModelMap@5010，与args是同一个对象（非常重要）！！！！

（备注：SpringMVC是如何为mavContainer 对象中的 defaultModel 属性赋值的，最开始调试了很久也没有发现，心想着，它既然是个Map，那应该是调用setXXX(), put(), putAll()这样的方法赋值进去的，但调试了很久始终没发现这样的方法被调用；同时，也可以确定它是在Line136行调用后就被赋值的；最后猜测，是同一个对象引用；现在，证明确实如此）

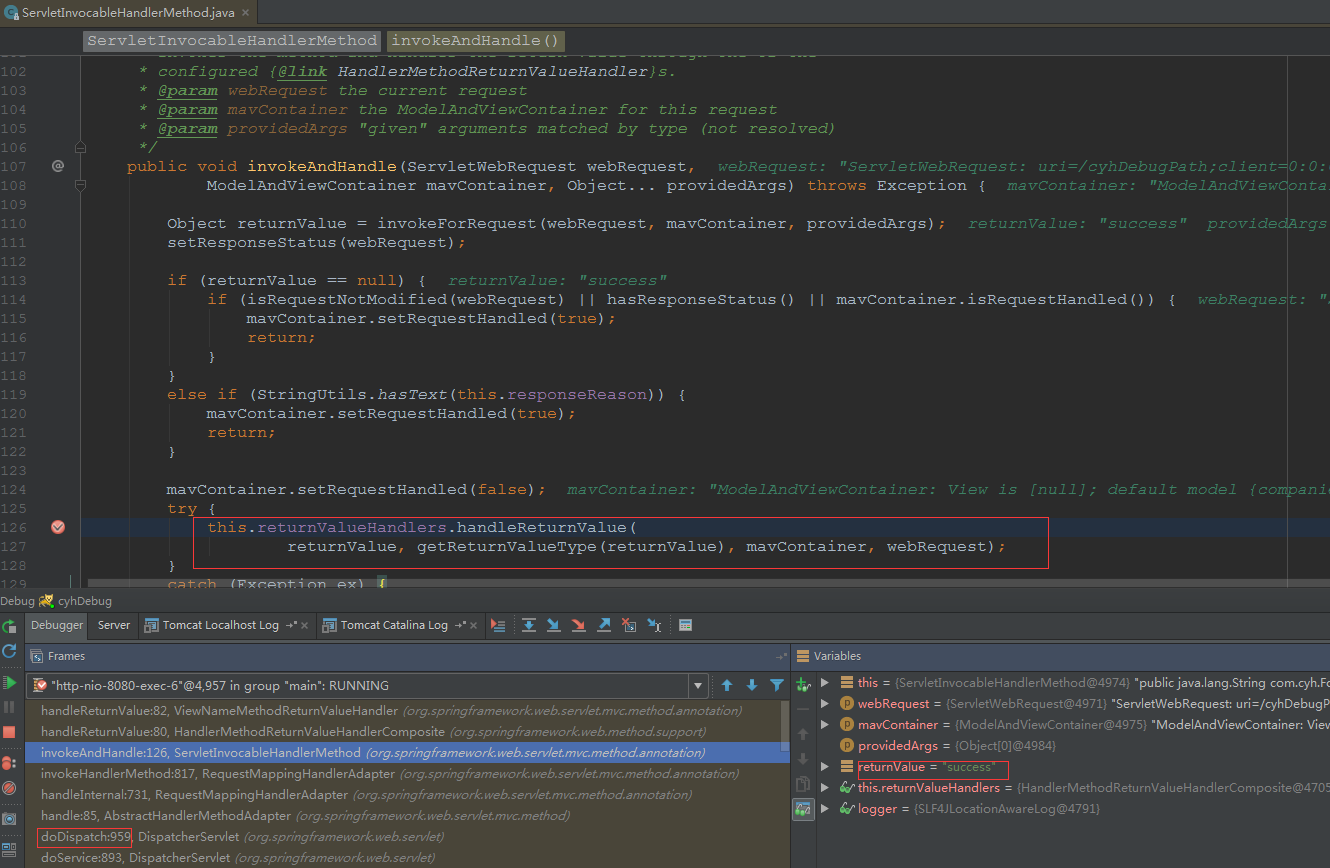
再啰嗦一句，其实就是：Line128的 Object[] args 变量指向了 mavContainer 对象的 defaultModel 属性！所以在将 args 通过Line136反射调用真实的Controller方法并填充数据后，defaultModel中也就有了相应数据！



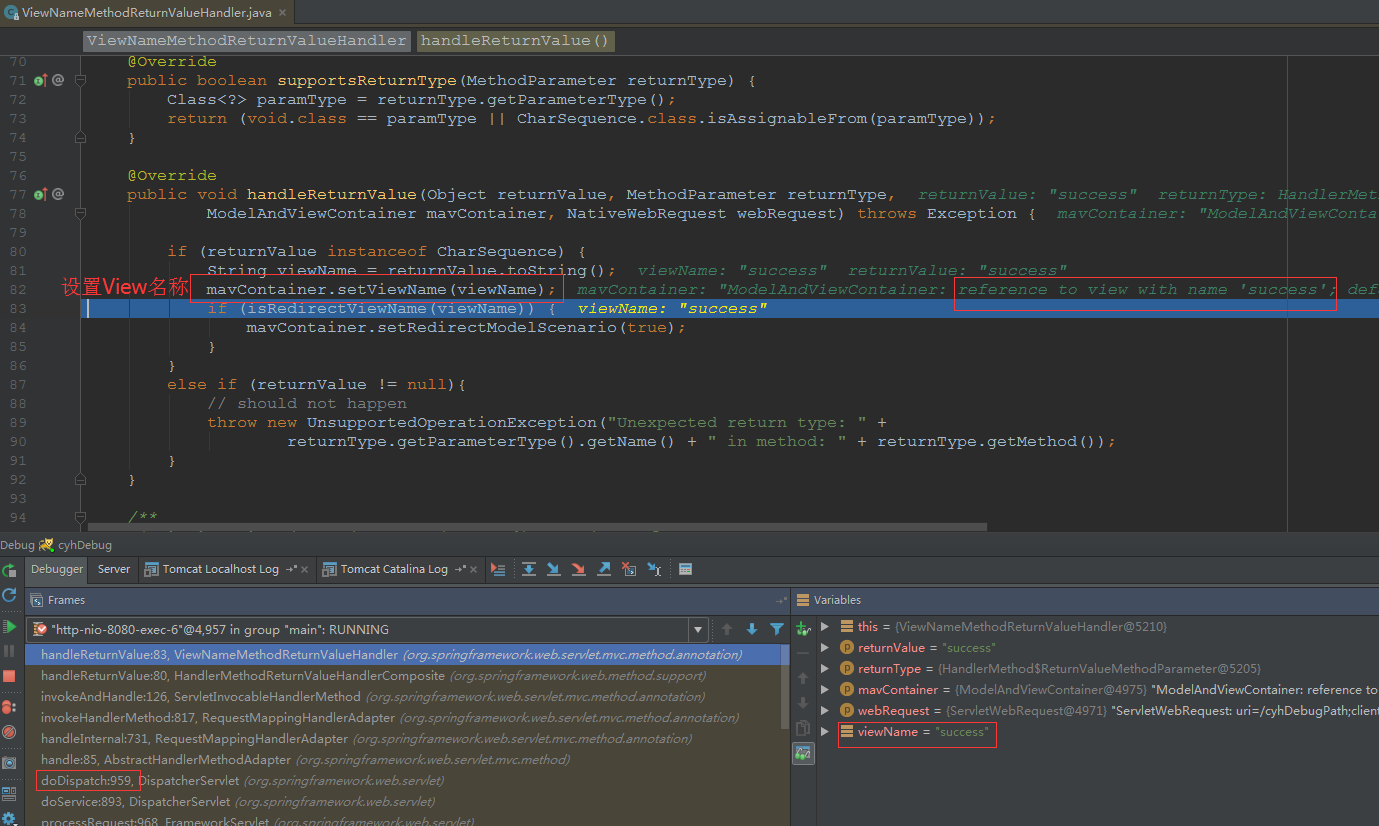
10.控制台打印的入参Map其实也是BindingAwareModelMap（遗憾的是没有拿到内存地址；但还是可以辅助证明09中的结论）



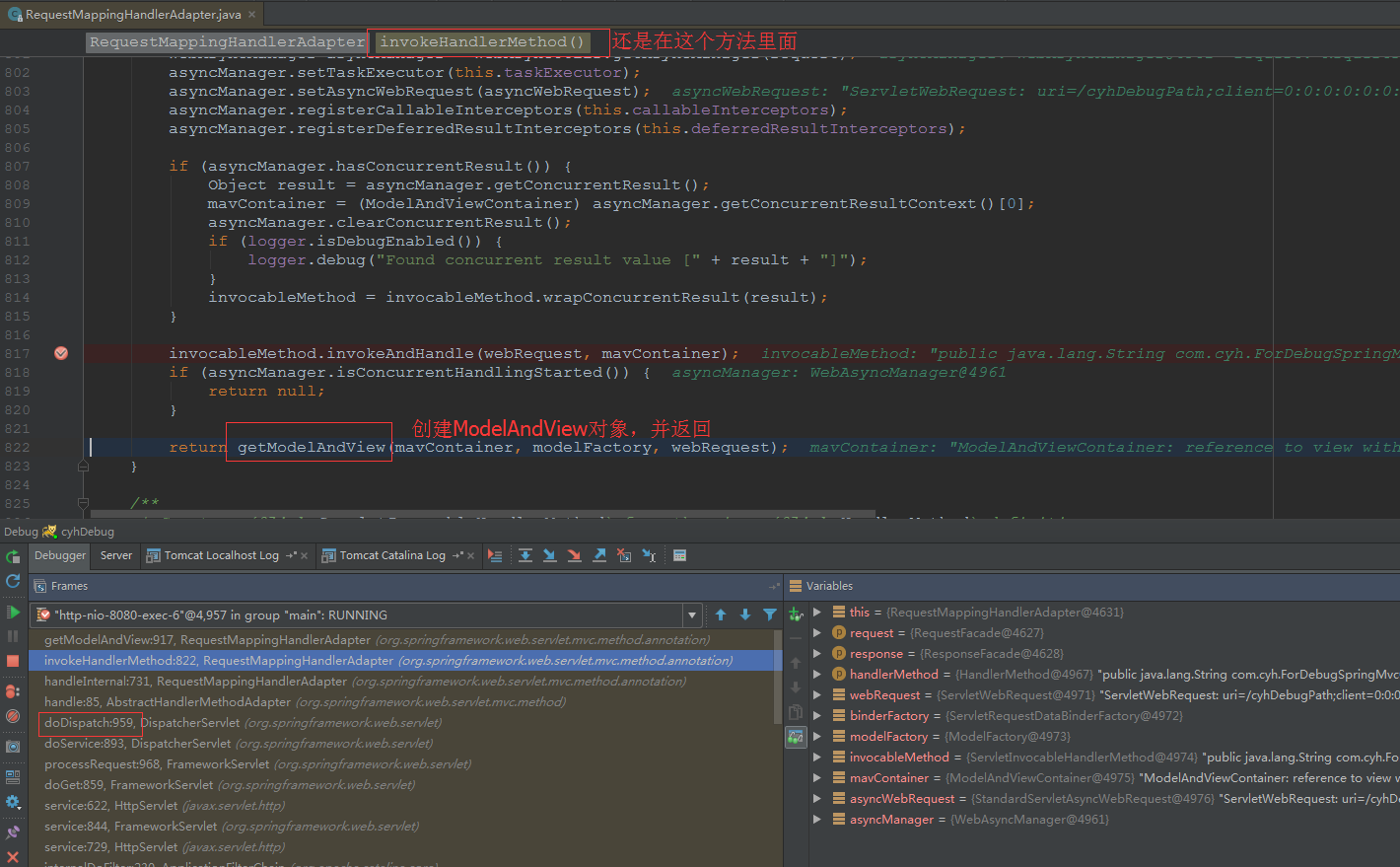
11.真实Controller方法调用完成后，开始处理返回值



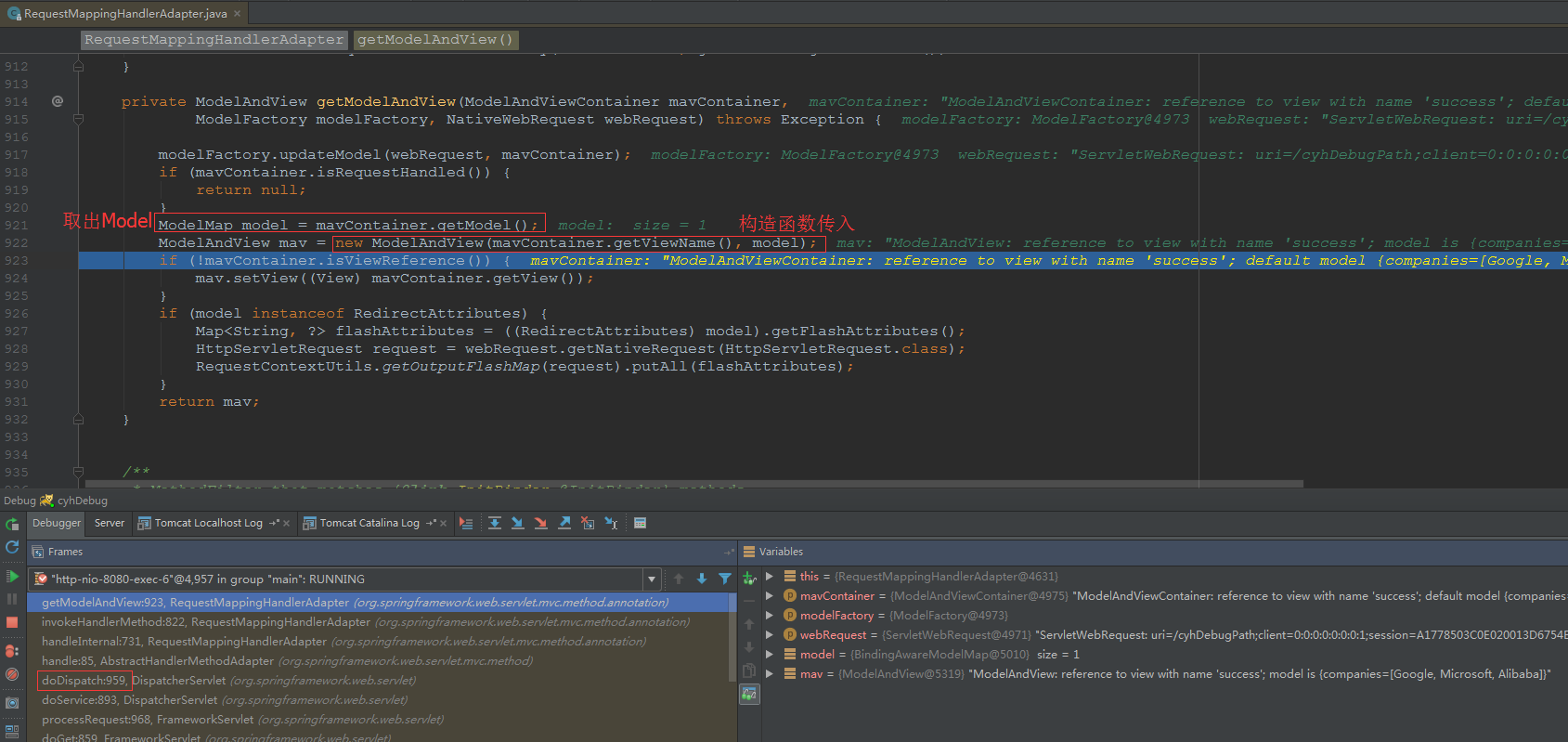
12.设置视图名称



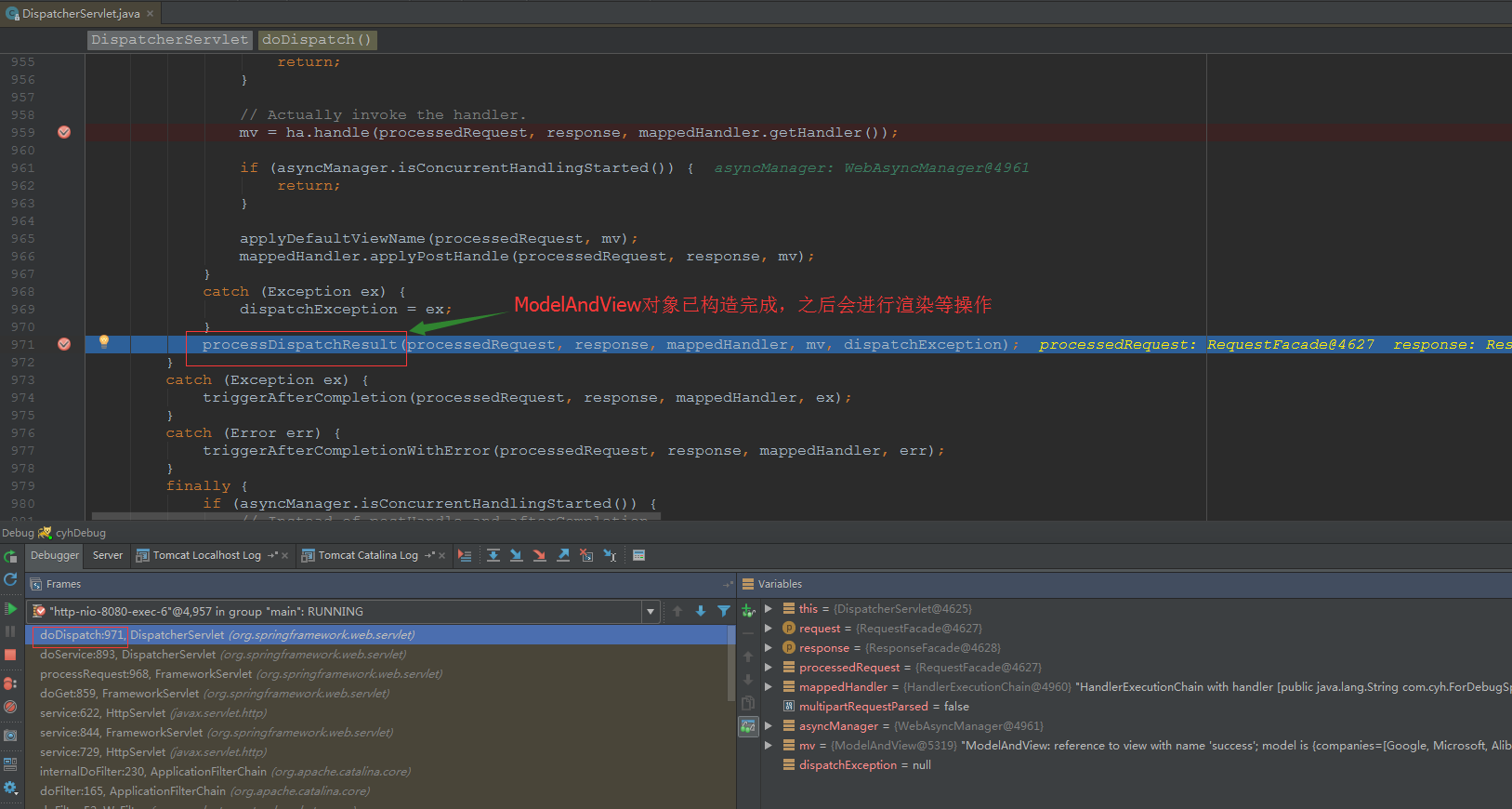
13.准备创建ModelAndView对象



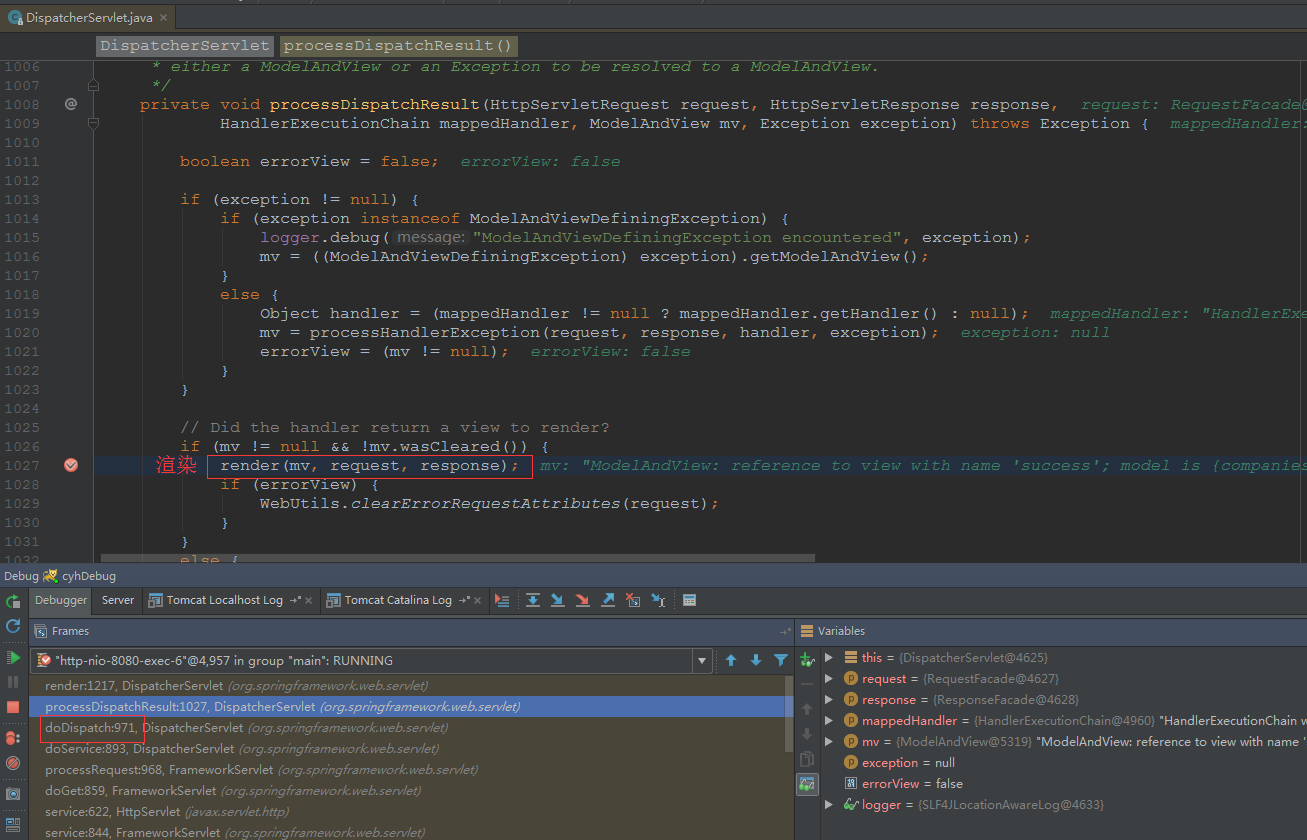
14.从 mavContainer 中取出Model数据，并通过构造函数传入ModelAndView



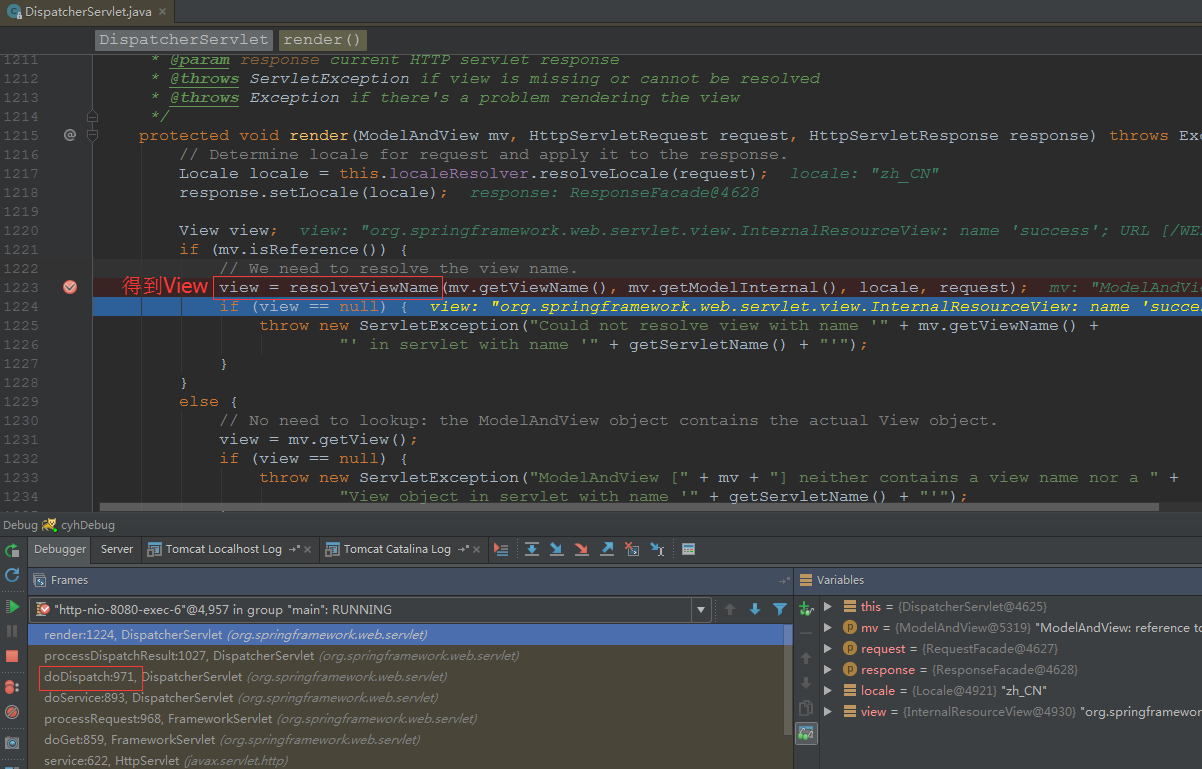
15.已经获得ModelAndView对象，进行后续操作（如渲染）；注意，此时DispatcherServlet中的方法堆栈从Line971开始。



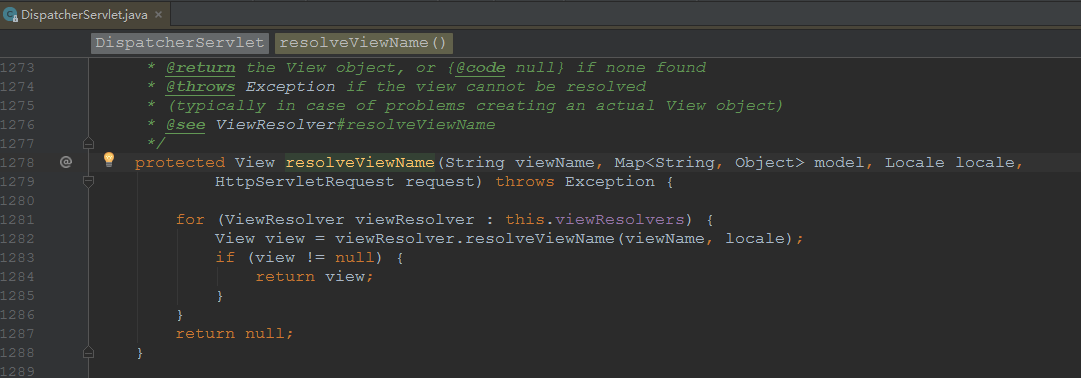
16.准备渲染



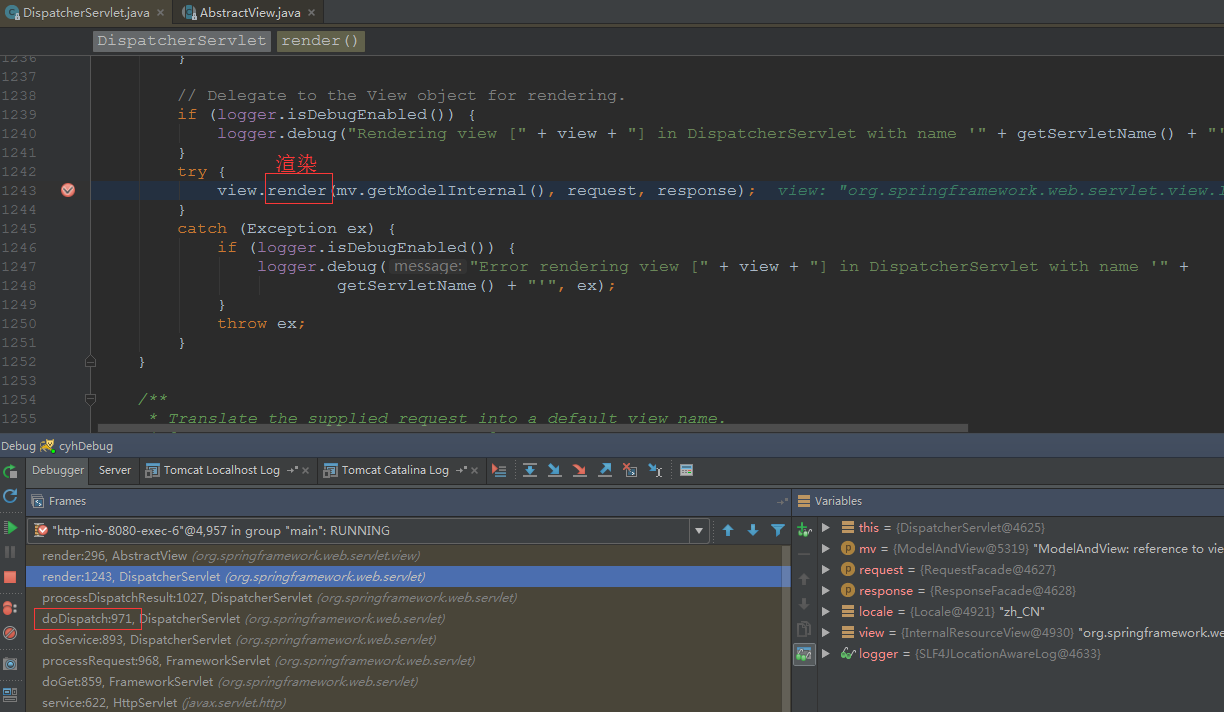
17.得到View对象



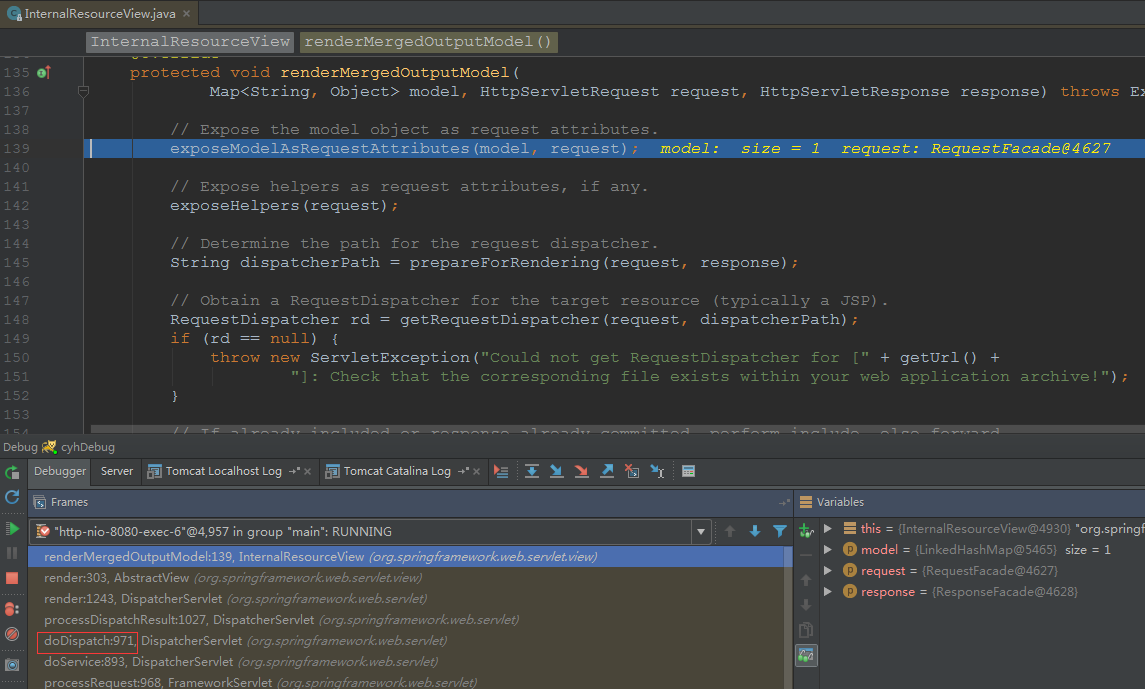
18.遍历 viewResolvers 找到一个合适的就返回给17步中的View对象



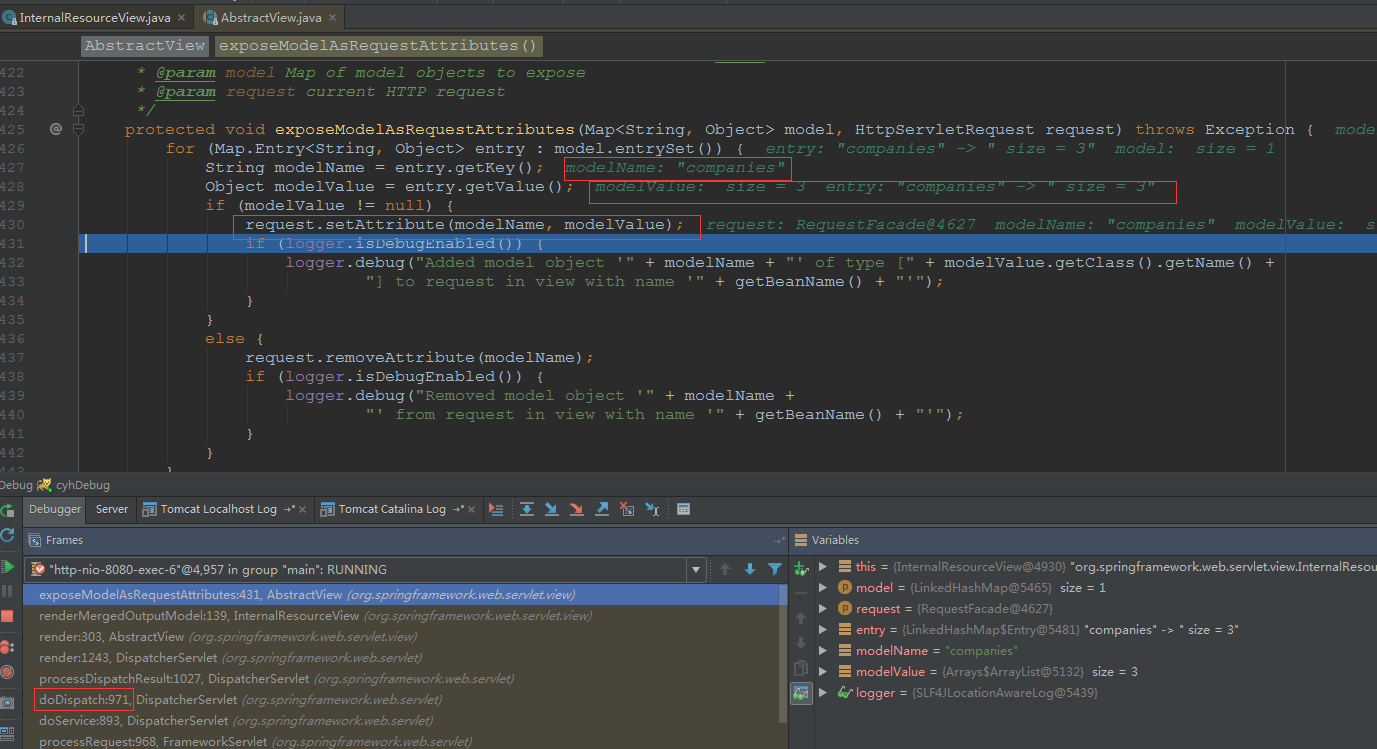
19.准备渲染



20.将Model数据暴露为RequestAttribute



21.暴露的本质其实是：request.setAttribute(modelName, modelValue)



22.最后是一个转发操作

