# Spring

http://jinnianshilongnian.iteye.com/category/231099

## IOC

1. 定义好Spring的配置文件。  
2. 通过Resource对象将Spring配置文件进行抽象，抽象成一个Resource对象。  
3. 定义好Bean工厂（各种BeanFactory）。  
4. 定义好XmlBeanDefinitionReader对象，并将工厂作为参数传递进去供后续回调使用。  
5. 通过XmlBeanDefinitionReader对象读取之前抽象出的Resource对象（包含了XML文件的解析过程）。  
6. IoC容器创建完毕，用户可以通过容器获取到所需的对象信息。

### 实现原理：

#### 三种启动方式

##### 监听器启动：

ContextLoaderListener与RequestContextListener区别：

ContextLoaderListener(或ContextLoaderServlet)将Web容器与spring容器整合。RequestContextListener将Spring容器与Web容器结合的更加密切。 前者为**必选配置**，后者为**可选配置**，并且后者与**scope=”request”**搭配使用。

在web.xml中配置spring的监听：

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/classes/applicationContext-\*.xml</param-value>

</context-param>

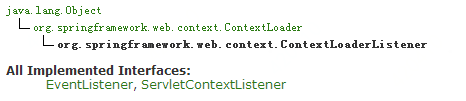
<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

ContextLoaderListener的继承关系：



当容器启动触发监听实现，调用初始化WebApplicationContext的方法

public void contextInitialized(ServletContextEvent event) {  
 this.initWebApplicationContext(event.getServletContext());  
}

接下来介绍spring容器初始化过程中非常关键的类或接口

**XmlWebApplicationContext：应用的容器，上下文，BeanFactory就是放在这里的**

**DefaultListableBeanFactory：spring IOC的容器**

**BeanDefinitionRegistry：将BeanDefinition注册到beanFactory里**

**XmlBeanDefinitionReader：spring xml文件的阅读器**

**：存放spring配置文件xml解析的资源**

**DefaultBeanDefinitionDecumentReader：xml解析成Decument之后的阅读器**

**BeanDefinitionParserDelegate：BeanDefinition解析授权**

**BeanDefinitionHolder：BeanDefinition的持有者**

**GenericBeanDefinition：里面存放了beanClass全路径名**

其中**DefaultListableBeanFactory是XmlWebApplicationContext的属性，GenericBeanDefinition是DefaultListableBeanFactory的属性，**

**ResourceLoader ApplicationContext**

**Implements extends**

**DefaultResourceLoader ConfiguableApplicationContext**

**extends implemetns**

**AbstractApplicationContext**

**extends**

**AbstractRefreshConfigApplicationContext**

**extends**

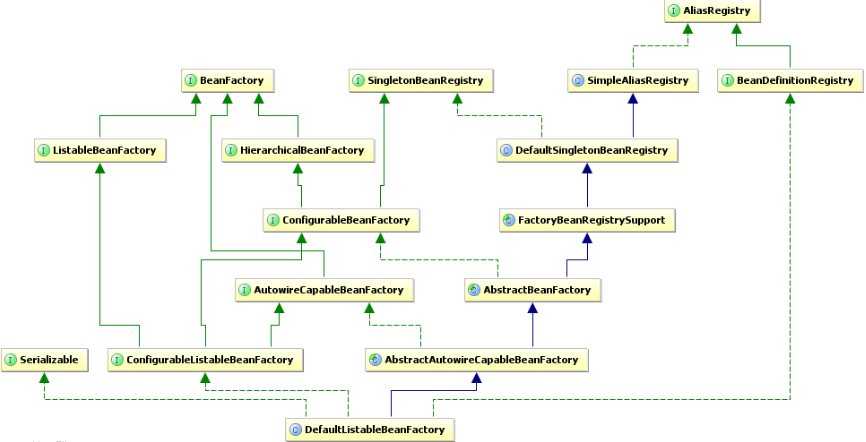
**AbstractRefreshConfigApplicationContext**

**extends**

**AbstractRefreshableWebApplicationContext**

**extends**

**XmlWebApplicationContext**

****

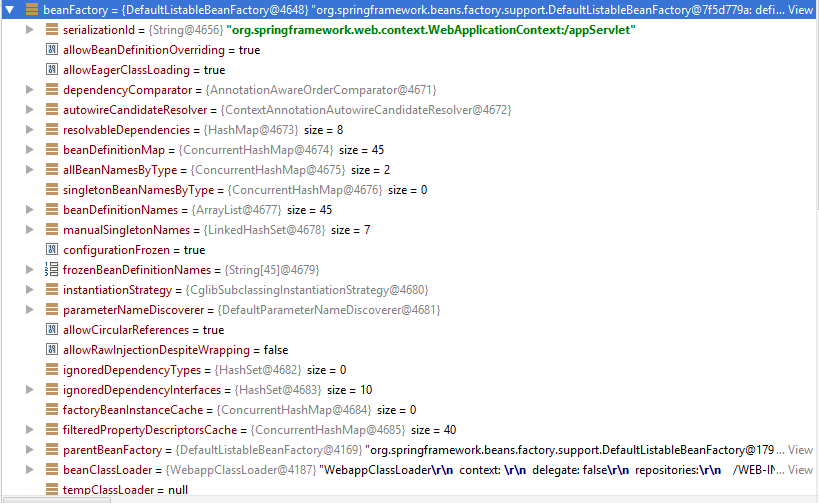
**XmlWebApplicationContext详解**

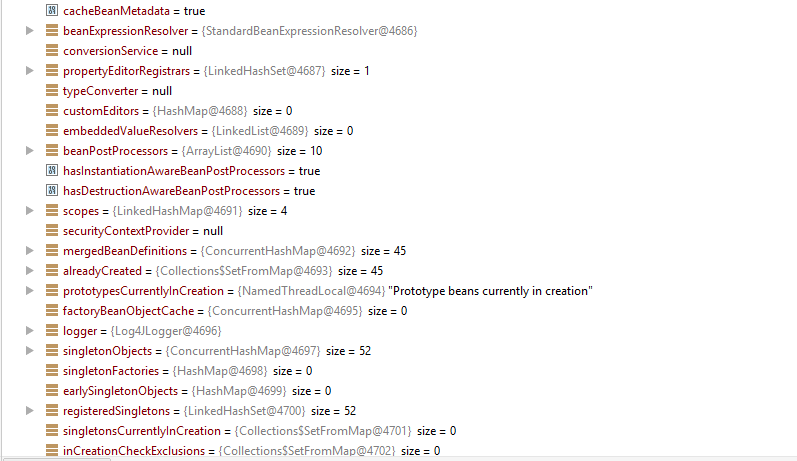
应用的容器，beanFactory作为他的属性，即为spring容器的宿主，并且它继承了ResourceLoader接口，所以也可以作为Resource对象的加载器

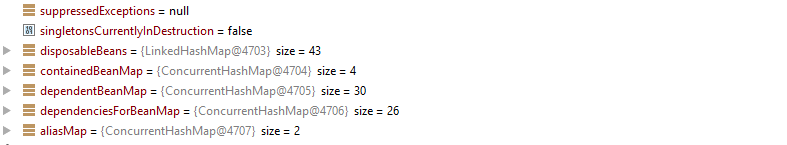
bean 实例的初始化是在**AbstractApplicationContext**的

finishBeanFactoryInitialization( beanFactory )方法里进行的，这个方法主要做了以下几件事：

**DefaultListableBeanFactory详解**





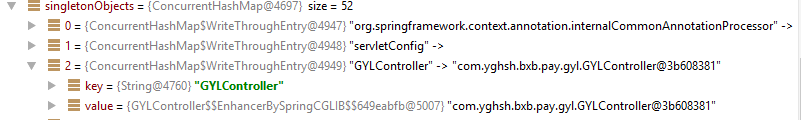


属性：

Map<String, BeanDefinition> beanDefinitionMap

singletonObjects=new ConcurrentHashMap<String, Object>(256);

这个属性是从**DefaultSingletonBeanRegistry**类继承来的



**GenericBeanDefinition详解**

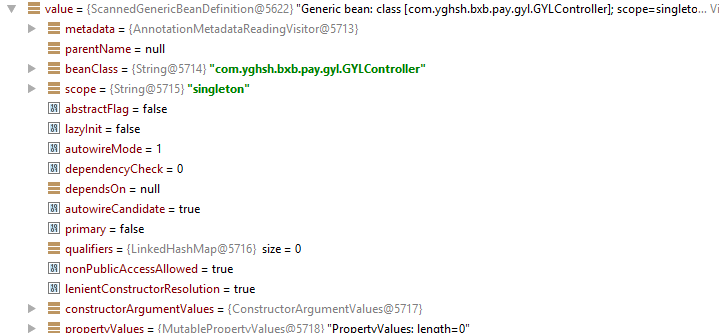
属性：

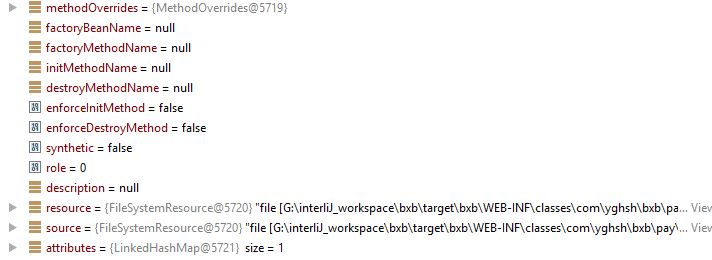
Object beanClass：bean类的全路径名

String scope：范围

boolean lazyInit ：是不是懒加载

int autowireMode





##### servlet方式

spring3.0已经不再支持

<servlet>

<servlet-name>context</servlet-name>

<servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

Spring3.0以后不再支持

##### plugin配置方式

<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">

<set-property property="contextConfigLocation" value="/WEB-INF/applicationContext.xml,/WEB-INF/action-servlet.xml" />

</plug-in>

用于和struts整合

## DI

## AOP

[Spring AOP中pointcut expression表达式解析](https://www.cnblogs.com/panie2015/p/5889729.html)

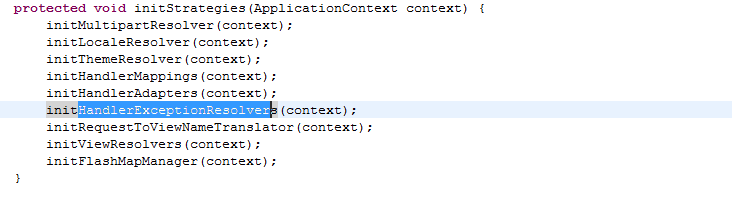
**见spring下的aop原理与使用**

## SpringMVC

使用springMVC在web.xml中配置如下：

<servlet>  
 <servlet-name>mis</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:conf/base-mis-servlet.xml</param-value>  
 </init-param>  
 <load-on-startup>1</load-on-startup>  
</servlet>  
<servlet-mapping>  
 <servlet-name>mis</servlet-name>  
 <url-pattern>/\*</url-pattern>  
</servlet-mapping>

**重要的组件说明：**

****

**MultipartResolver：上传文件处理组件**

**LocaleResolver：国际化处理组件（zh-cn之类）**

**ThemeResolver：主题处理组件**

**HandlerMapping：根据request找到相应的HandlerMethod（处理器）**

**HandlerAdapter：使用处理器处理的组件**

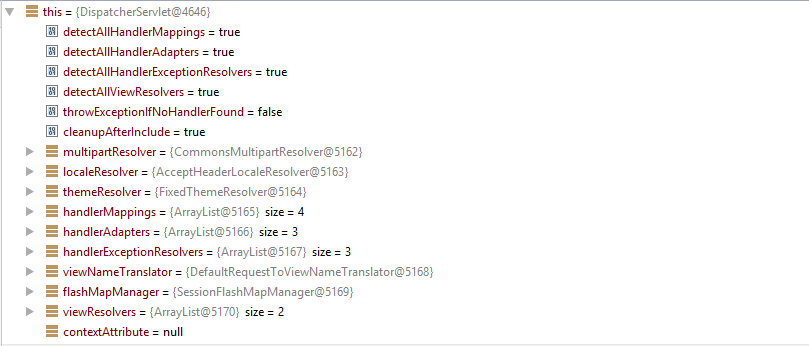
**HandlerExceptionResolver：异常处理组件**

**RequestToViewNameTranslator：**

**ViewResolvers：视图解析器**

**FlashMapManager：重定向传递参数组件**

**DispatcherServlet**

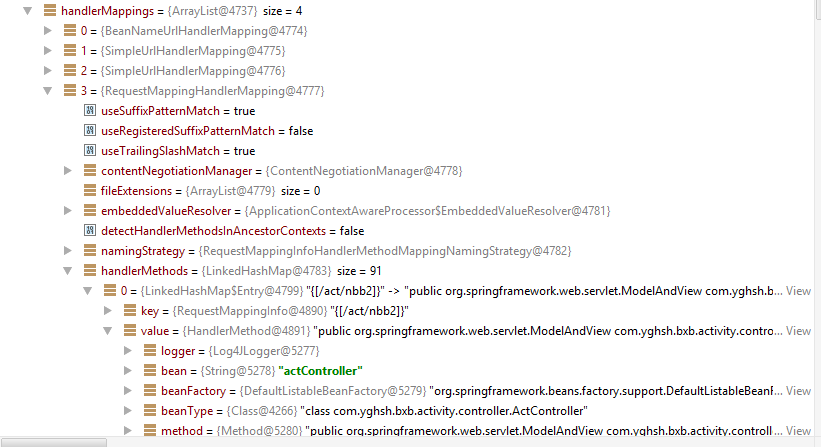




### Init()初始化

#### handlerMapping详解

RequestMappingHandlerMapping **extends** RequestMappingInfoHandlerMapping **extends** AbstractHandlerMethodMapping<RequestMappingInfo> **extends** AbstractHandlerMapping **implements** InitializingBean



当spring初始化handlerMapping这个bean的时候，AbstractHandlerMethodMapping<T>属性加载完会调用afterPropertiesSet()方法，这个方法里调用initHandlerMethods

整个初始化工作由AbstractHandlerMethodMapping的initHandlerMethods主导.

　　1. 取出容器中所有的beanNames

　　2. 迭代类,分别判断isHandler判断目标类是否Handler

　　2.1 RequestMappingHandlerMapping.isHandler根据@Controller或@RequestMapping注解判断(有任意一个)

　　3. 对handler解析出所有需要分发的方法detectHandlerMethods

　　　　3.1 获取原始的Class<?>

　　　　3.2 使用HandlerMethodSelector.selectMethods过滤具体handler method,预留getMappingForMethod模板方法给子类

　　　　　　RequestMappingHandlerMapping.getMappingForMethod根据类,方法上的RequestMapping注解生成匹配条件RequestMappingInfo

　　　　3.3 对过滤到的每个method进行注册registerHandlerMethod

　　　　　　a, 使用createHandlerMethod封装处理器为HandlerMethod

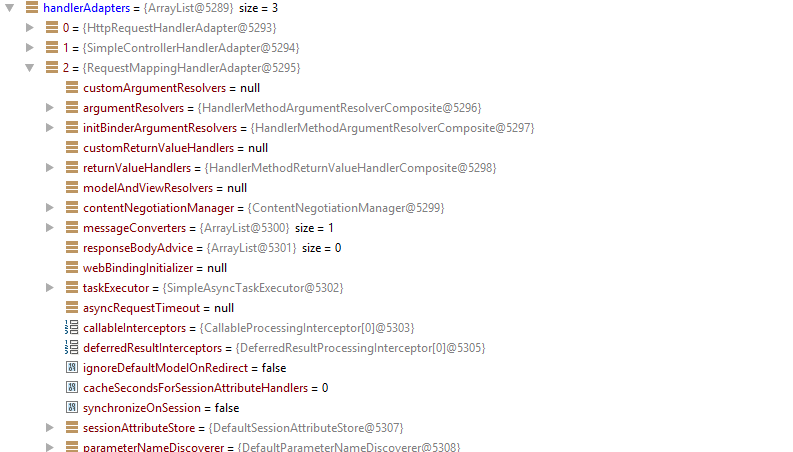
　　　　　　b, 判断之前是否已经匹配条件对应的处理器是否冲突(相同的匹配条件只能有一个对应的处理器)

　　　　　　c, 设置匹配条件到handler method的映射关系

　　　　　　d, 从匹配条件中解析出url,并注册到urlMap(url到匹配条件的映射),这边由RequestMappingInfoHandlerMapping.getMappingPathPatterns

#### handlerAdapter详解

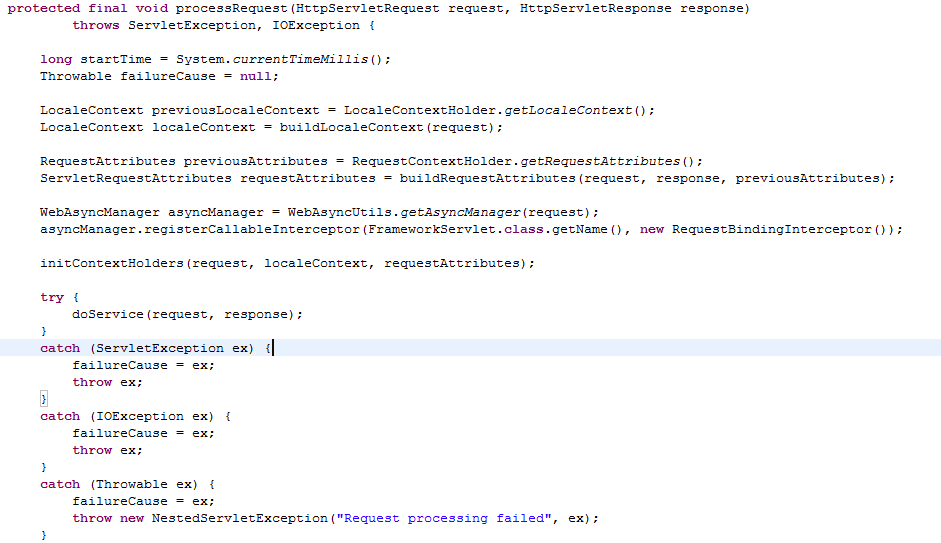
RequestMappingHandlerAdapter **extends** AbstractHandlerMethodAdapter  
 **implements** BeanFactoryAware, InitializingBean

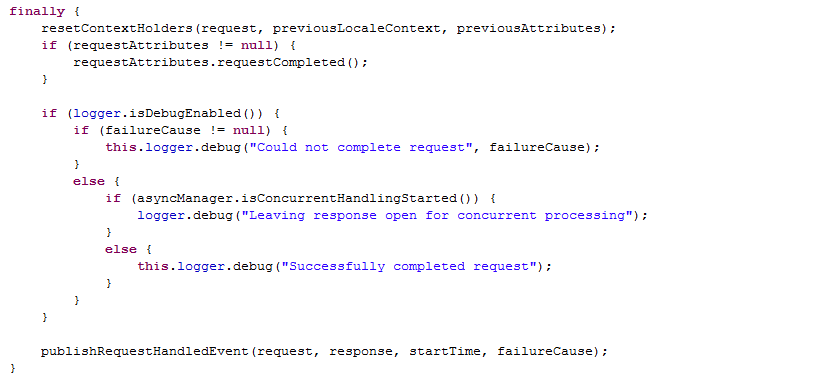


### service()方法

#### FrameworkServlet类

重写了doGet、doPost等方法，然后全部调用processRequest（）方法





最核心的是doService(request, response);方法，此为模板方法，在DispatcherServlet中实现

此方法主要做了两件事：

1.对LocaleContext和RequestAttributes的设置和恢复

LocaleContext：本地化信息，如zh-cn

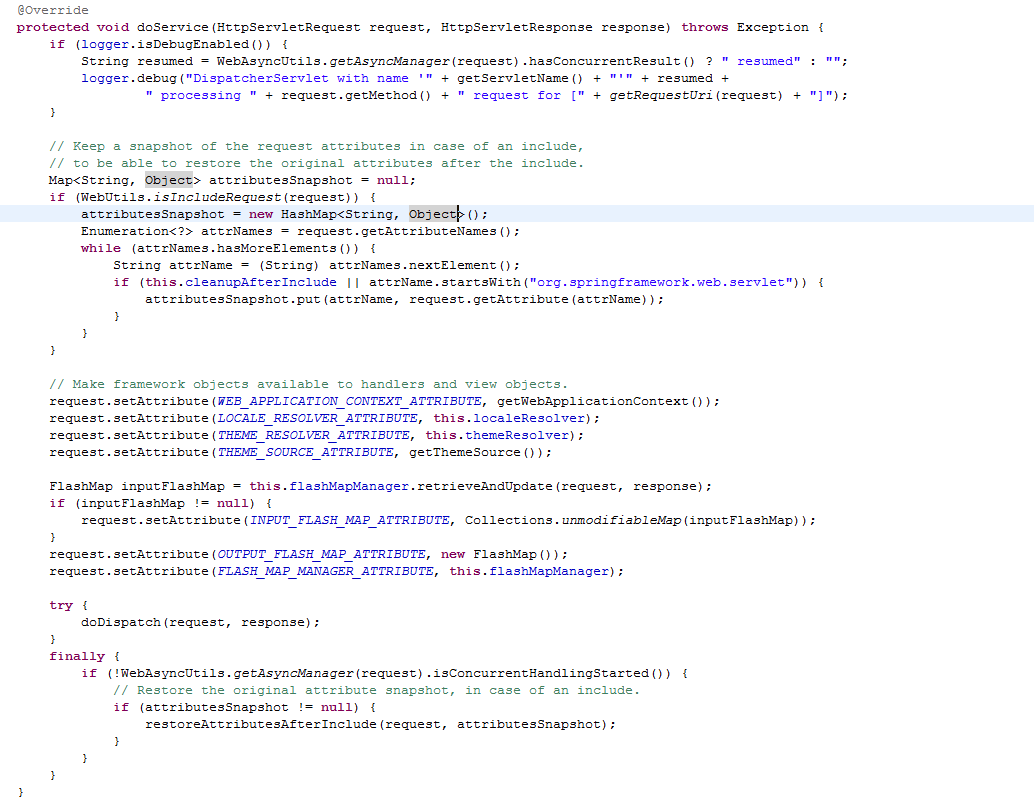
RequestAttributes：可以通过它操作request或者session中参数

2.处理完发布了一个ServletRequestHandledEvent事件消息

#### DispatcherServlet

springMVC最核心的类

##### doService（）方法



1.检查是不是include方法，是就对request的Attribute做一个快照备份

2.对request设置一些属性

FlashMap：主要用于redirect转发时的参数传递

redirect传递参数的三种方法：

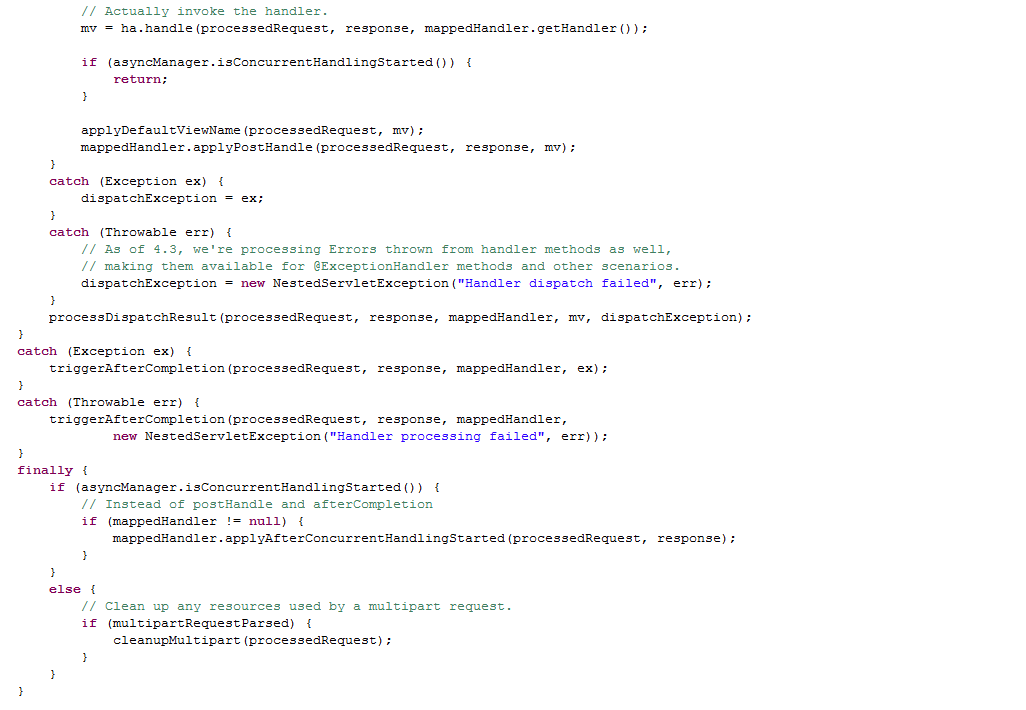
Controller的handle方法，引入参数RedirectAttributes attr

然后attr.addFlashAttribute(“orderId”,”123”);

3.调用**doDispatch(request, response);**

##### doDispatch





1.根据request找到handler

2.根据handler找到对应的HandlerAdapter

预处理（拦截器实现）

3.用HandlerAdapter处理Handler

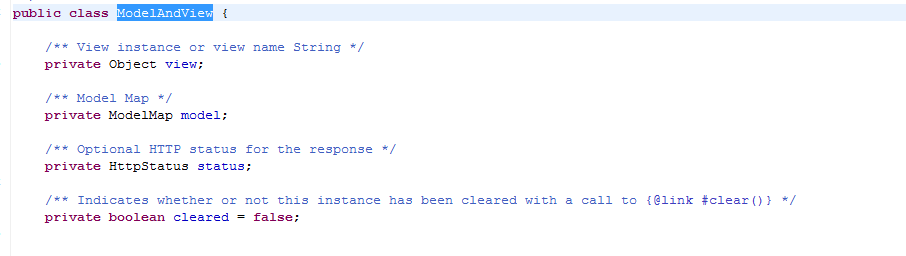
后处理（拦截器实现）

4.调用processDispatchResult处理上面结果（包含找到view并渲染输出给用户）

multipartRequestParsed 是不是上传请求标示

这里面牵涉到几个类：

ModelAndView



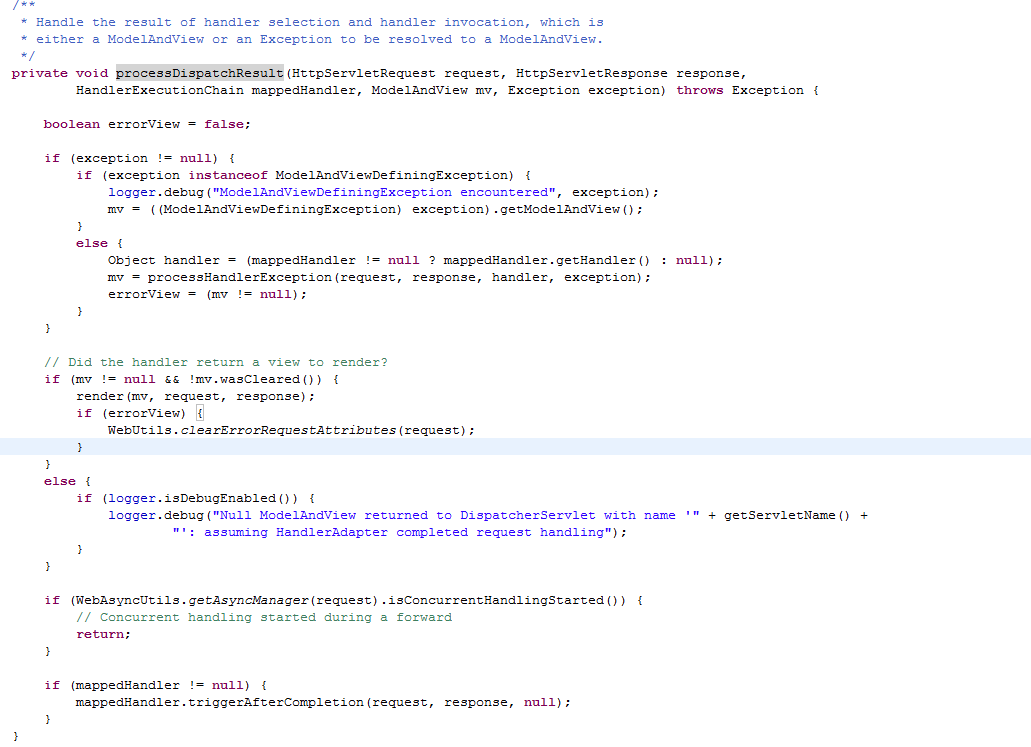
HandlerExecutionChain

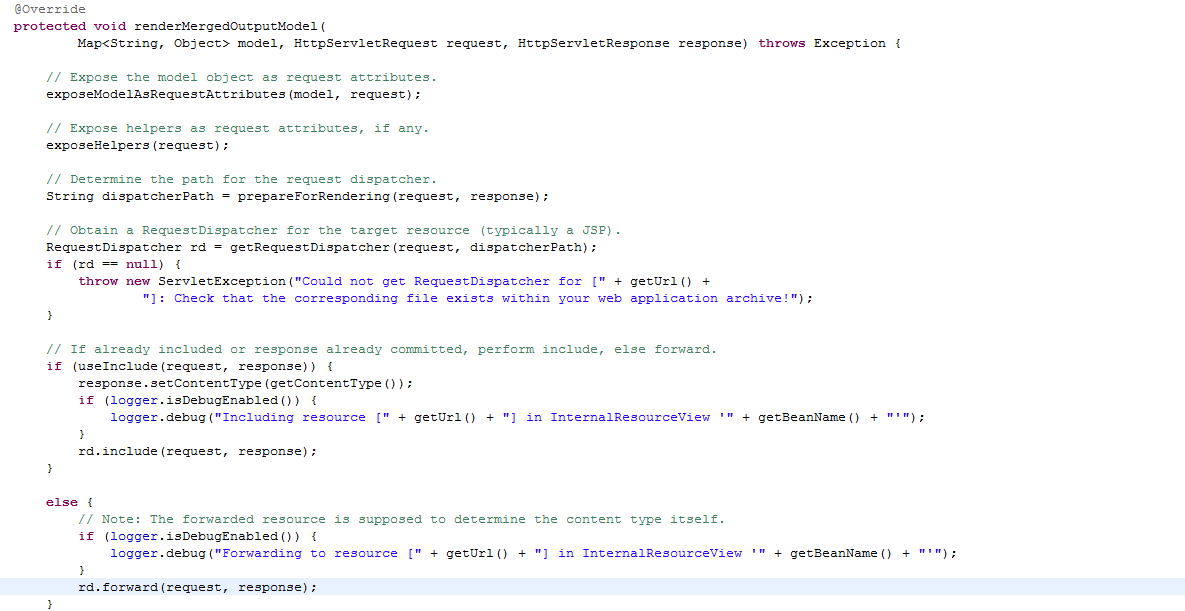
处理请求的处理器链（包含处理器和对应的interceptor拦截器）

执行的顺序是，请求的时候先执行拦截器的preHandle，再执行Handler，然后再执行postHandler方法

Controller的方法就是在Handler里面执行的

##### 渲染页面





## springMVC实战问题

405错误：[Spring MVC Request method 'POST' not supported错误](https://blog.csdn.net/ruiruiafafafa/article/details/51507150)

# 深入Spring:自定义注解加载和使用

https://blog.csdn.net/z69183787/article/details/53784845