[Spring MVC 原理 - DispatcherServlet调用完整过程（上）](http://blog.csdn.net/isea533/article/details/20949013)

标签： [spring](http://www.csdn.net/tag/spring)[spring mvc](http://www.csdn.net/tag/spring%20mvc)

2014-03-10 21:03 4733人阅读 [评论](http://blog.csdn.net/isea533/article/details/20949013#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/isea533/article/details/20949013#report)

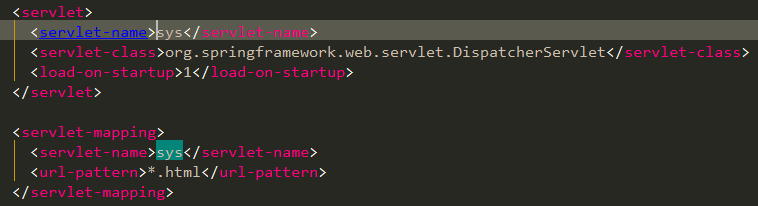
http://static.blog.csdn.net/images/category_icon.jpg 分类：

JAVA编程（72） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

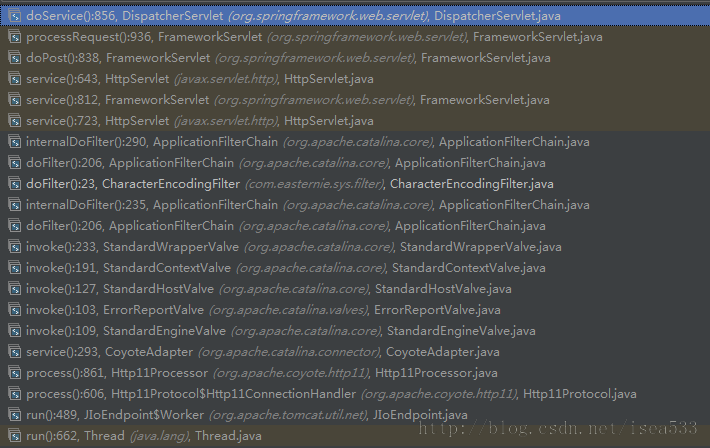
版权声明：版权归博主所有，转载请带上本文链接！联系方式：abel533@gmail.com

本篇主要讲解了Spring mvc的调用过程，通过这个过程来了解Spring的原理。

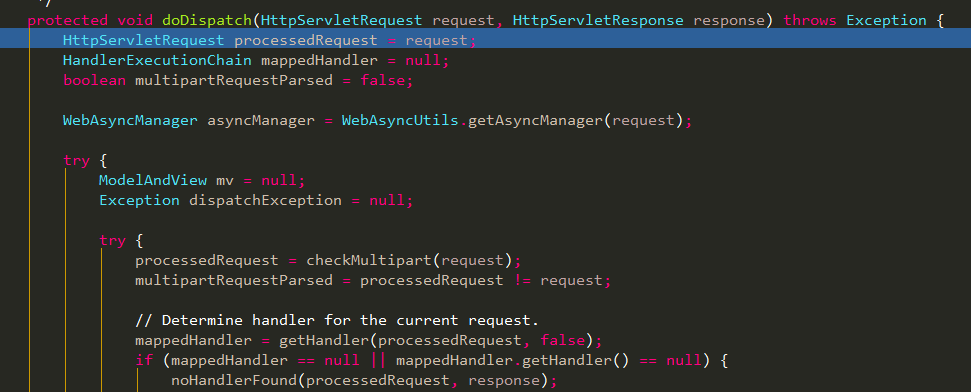
进入web.xml中配置的Spring拦截（调度）器：



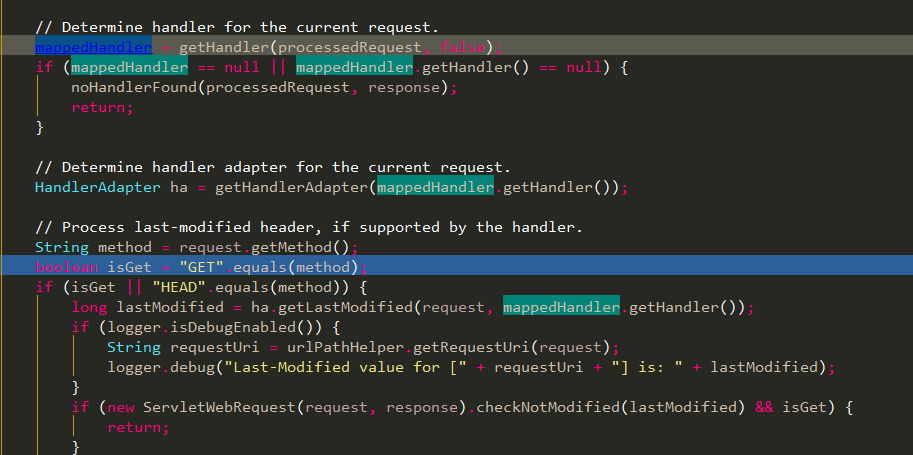
调用堆栈信息：



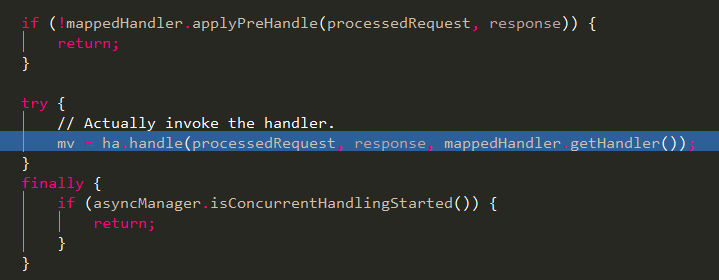
开始拦截，这一步主要是根据request获取handler（该handler可以配置）：



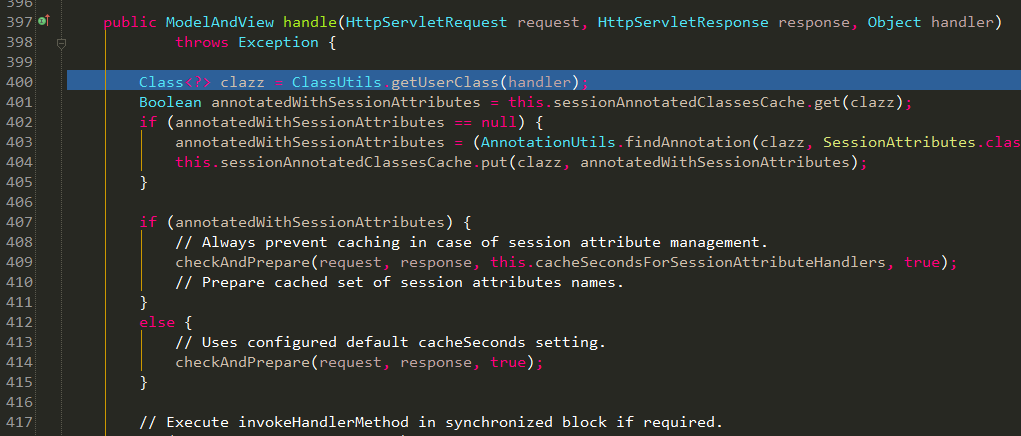
（接上面方法）这段代码中的if(isGet…主要解决浏览器的缓存问题，如果没有过修改，则返回（浏览器可能会用缓存响应）。



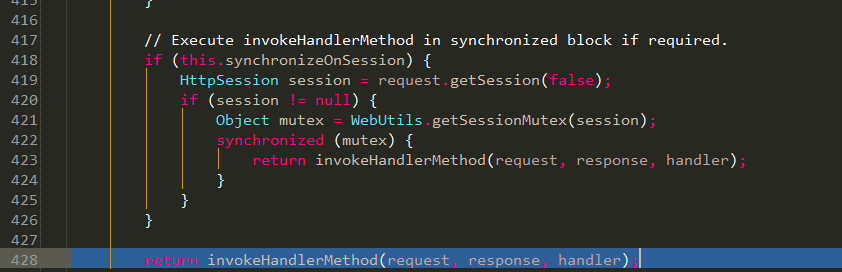
调用方法handler：



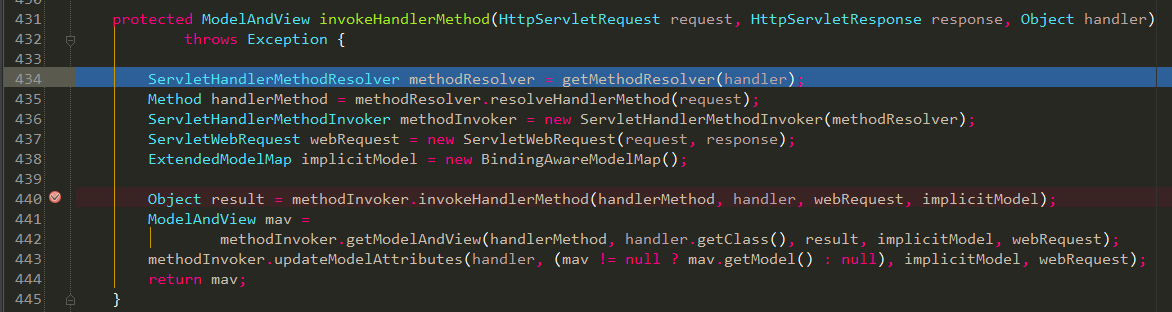
进入handler方法：

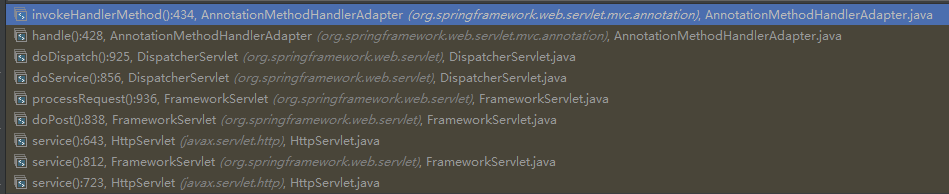


进入方法invokeHandlerMethod：

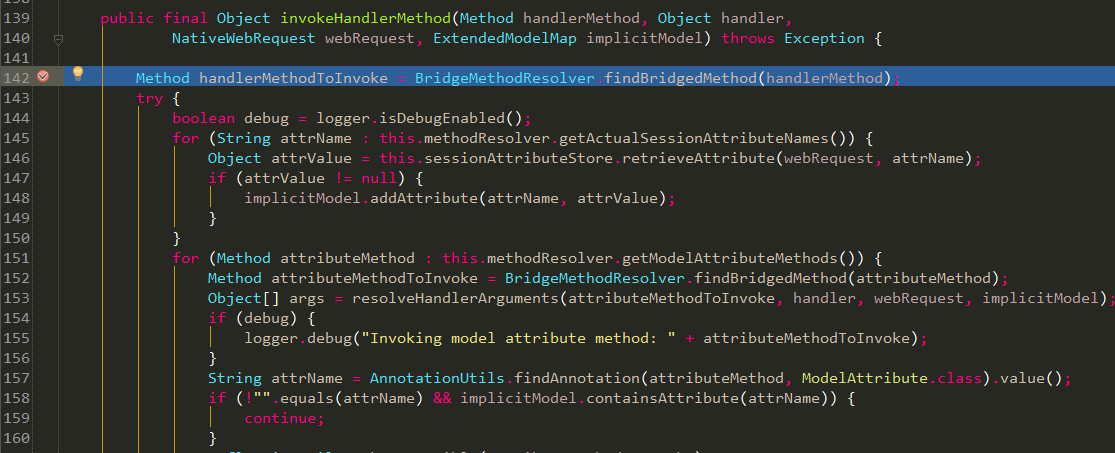


440行进入下一个方法invokeHandlerMethod：

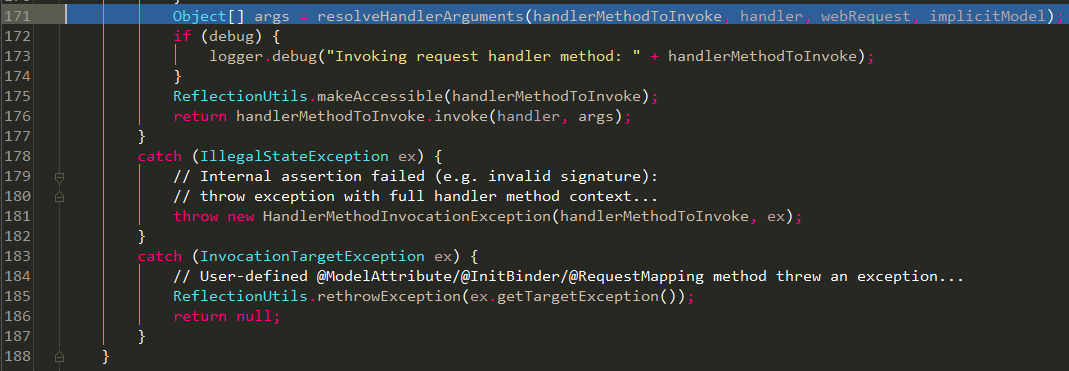


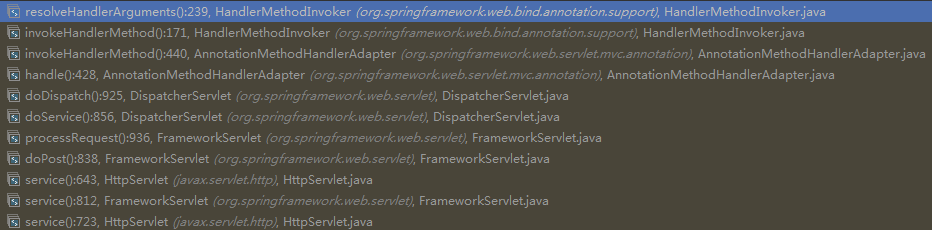


进入该方法：

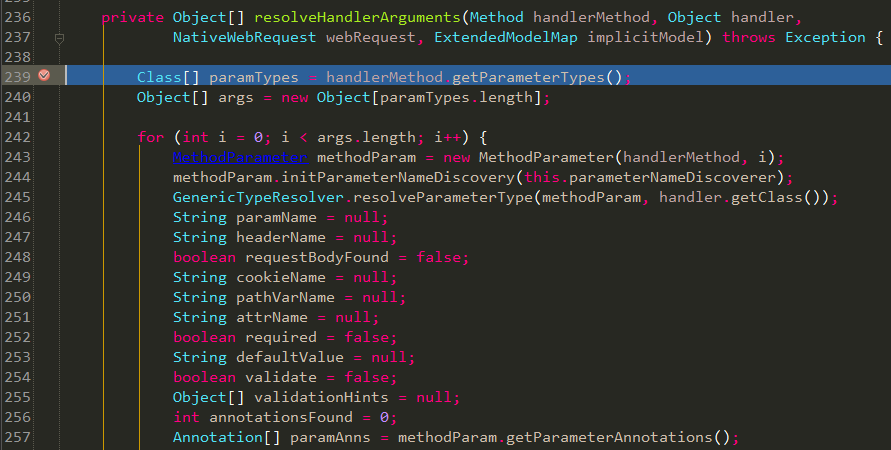


171行进入处理参数的方法：

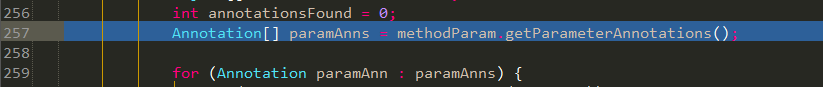




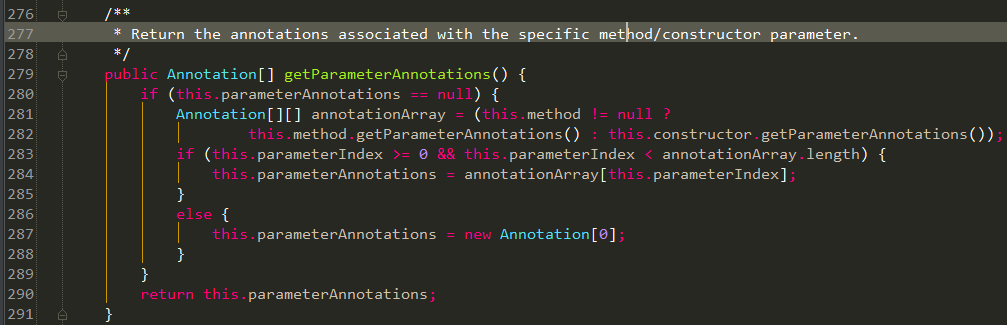
处理参数的方法，使用参数类型进行for循环：



获取一个参数的注解：



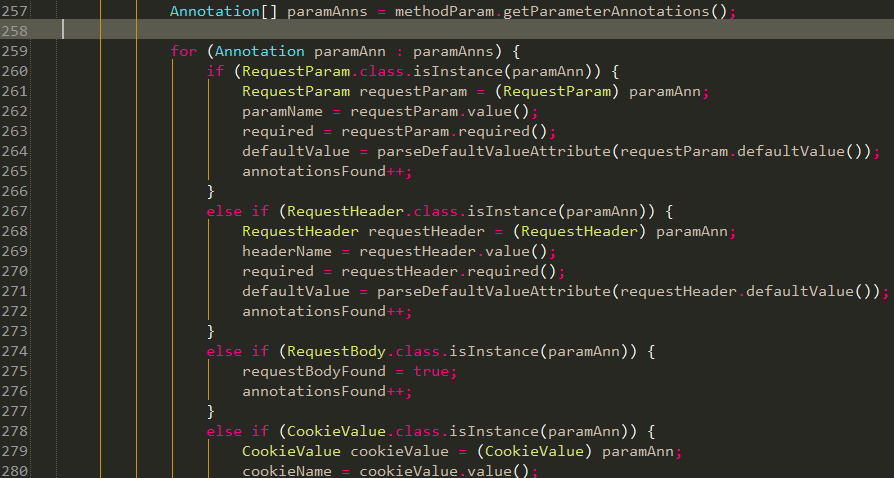
通过下面方法返回注解数组：

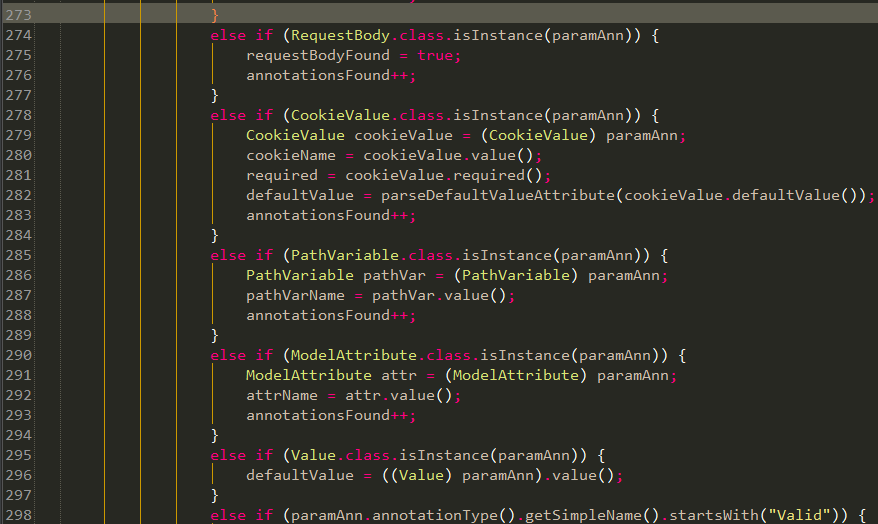


其中parameterIndex在创建该对象时赋值，和参数类型的位置是对应的（下图中的i）：

http://img.blog.csdn.net/20140310204343437

获取注解之后，对注解进行遍历，通过判断注解类型来获取paramName或headerName或其他：

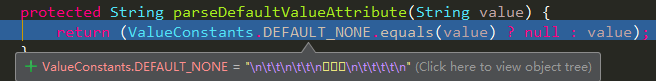




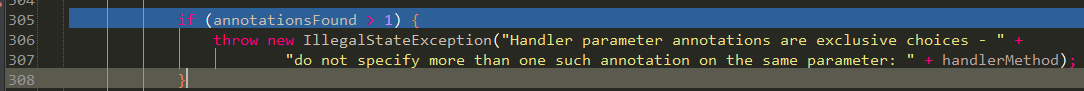
在每个if中的defaultValue中，方法parseDefaultValueAttribute用来获取注解的默认值设置：

http://img.blog.csdn.net/20140310204624187

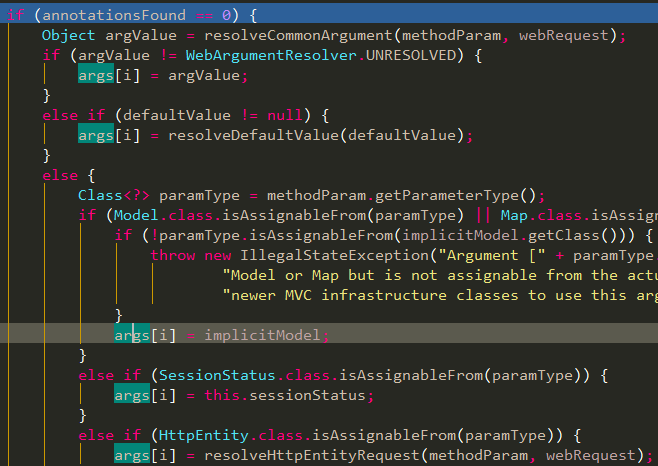
可以看到系统用了一个不常用的字符串作为默认值，用这个值和传入的value进行比较，使用这个值的目的是尽可能和用户设置的值区分开。



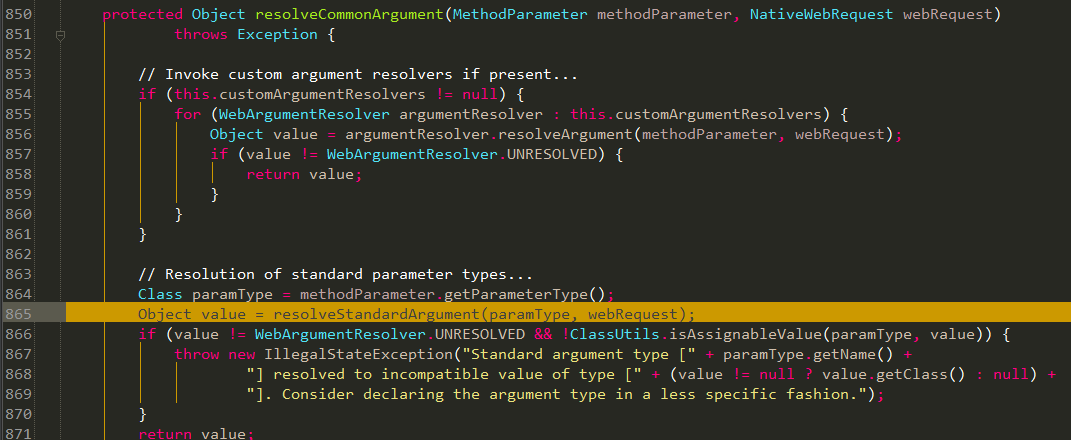
指定类型的注解（if判断里面的），只能存在一个，超过1个会报错：



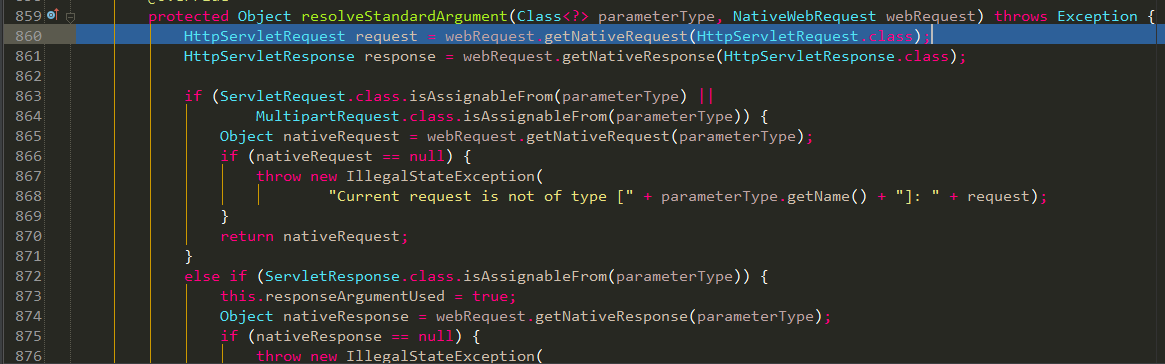
如果没有注解，则判断类型是否为常用的(request,response,session等)：

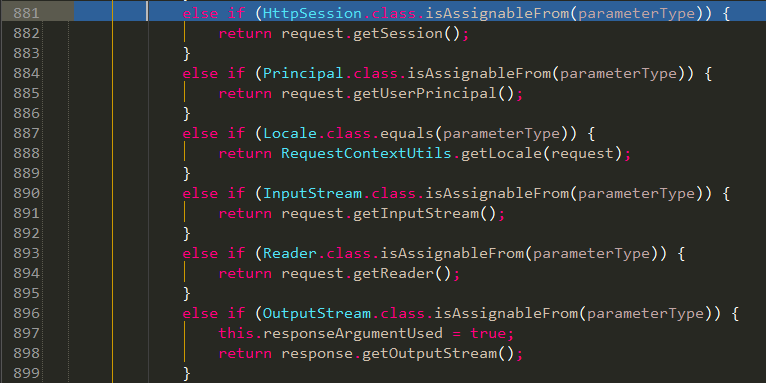


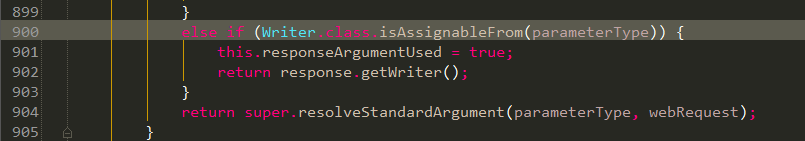
对于没有注解的参数，可能会执行下面的方法：



进入处理标准类型的方法：

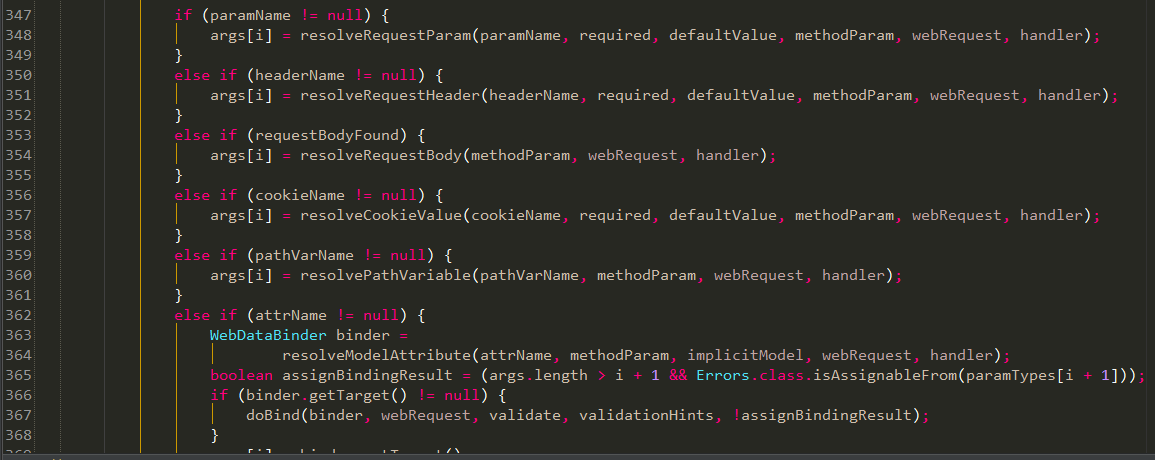






通过上面的方法判断出基本的类型。

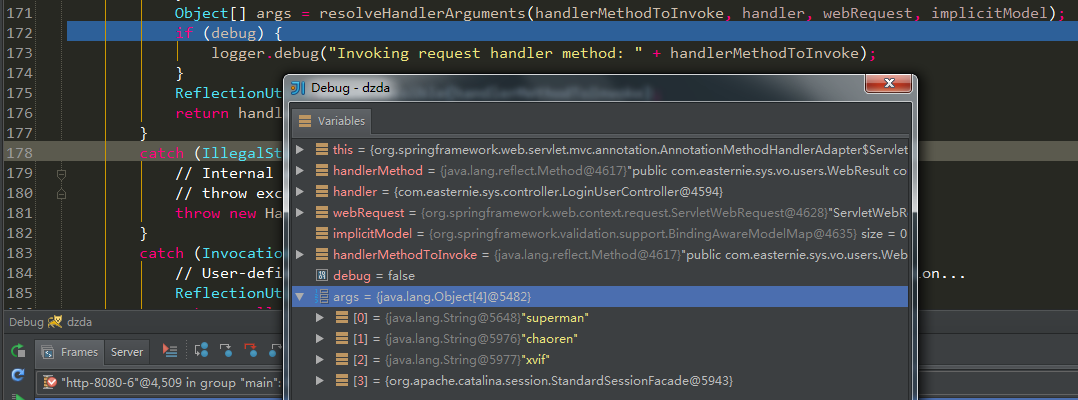
回到有注解的情况，针对不同的注解，具体处理过程如下（if顺序没有优先级，因为只有一个注解，一种情况）：



到这一步，会根据具体的注解类型执行不同的resolve\*\*\*()方法。

resolve\*\*\*的方法最终返回的是类型符合要求（resolve内部有各种类型转换的方法）的参数值。

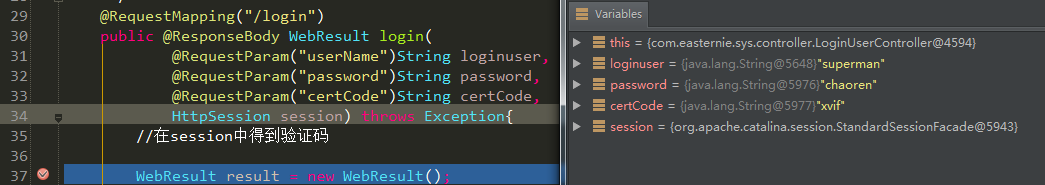
最后处理完成，返回参数列表：



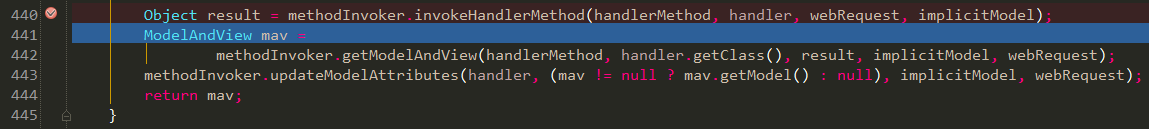
反射调用方法：

http://img.blog.csdn.net/20140310205758640

进入用户写的处理方法中：



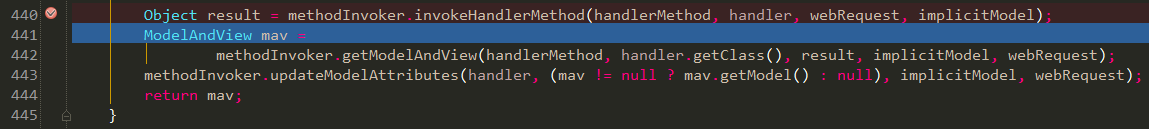
在用户方法执行完成后，返回：



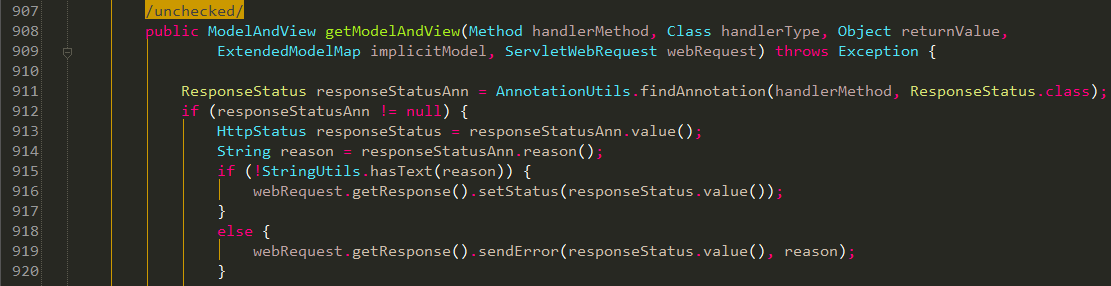
到这里，主要的方法就执行完了，后面还有对返回结果result的处理，后面的内容不是很多了，由于本篇已经包含了30多张图片，插入图片已经很不方便，所以考虑分成两个部分来说明，并且尽力让这第一部分比较完整。

接上一篇：[Spring MVC 原理 - DispatcherServlet调用完整过程（上）](http://blog.csdn.net/isea533/article/details/20949013)

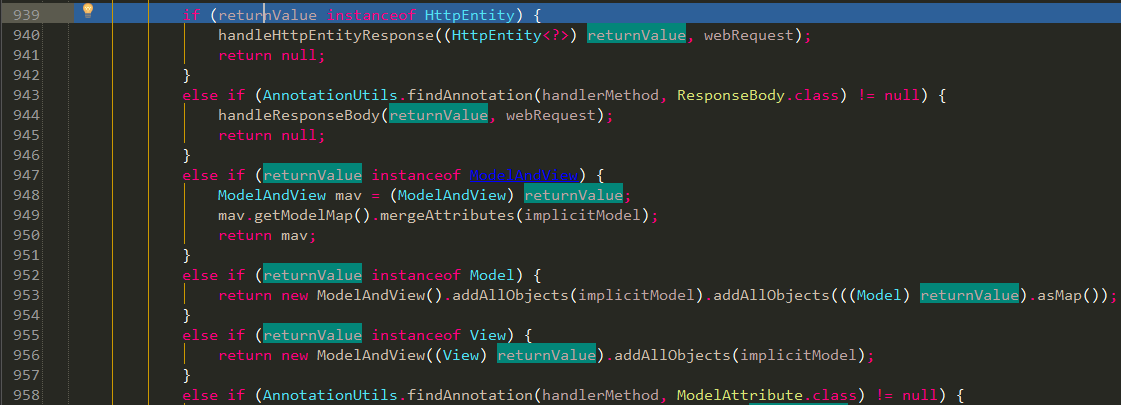
上一篇到440行返回结果result.



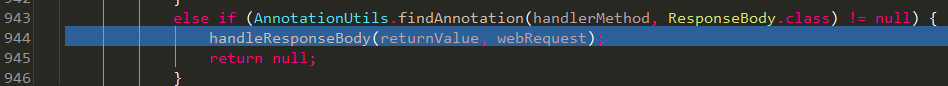
返回result之后，使用result去获取ModelAndView，执行方法getModelAndView：



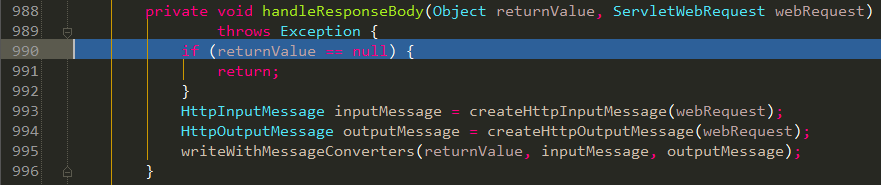
该方法主要通过result返回值来判断：



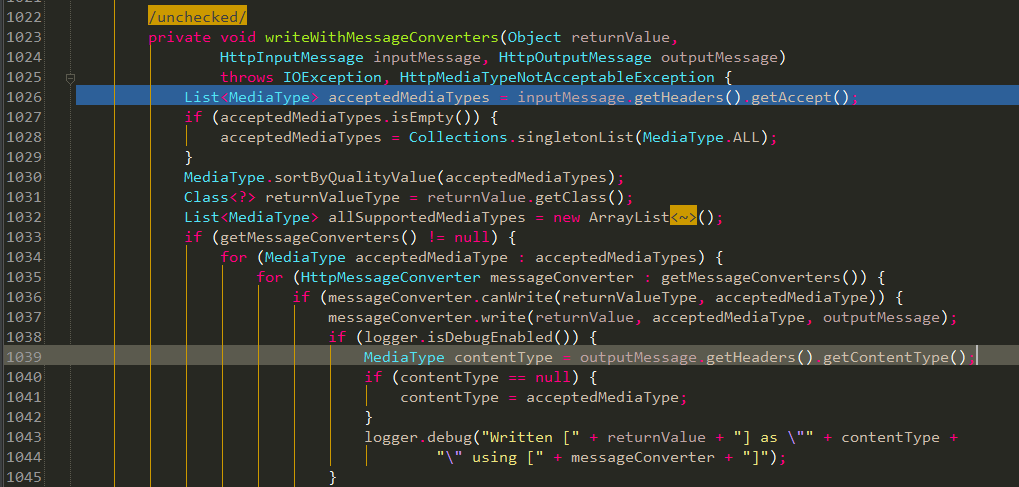
当前方法是带ResponseBody注解的，所以执行到这里：



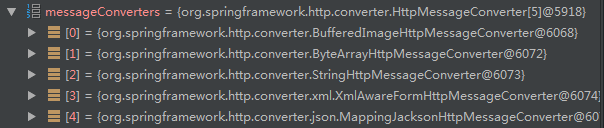
进去方法：



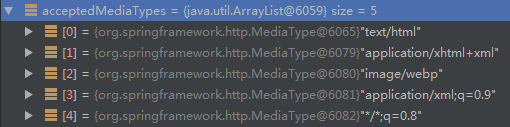
这里是根据http类型做出相应的输出：



用户配置的：



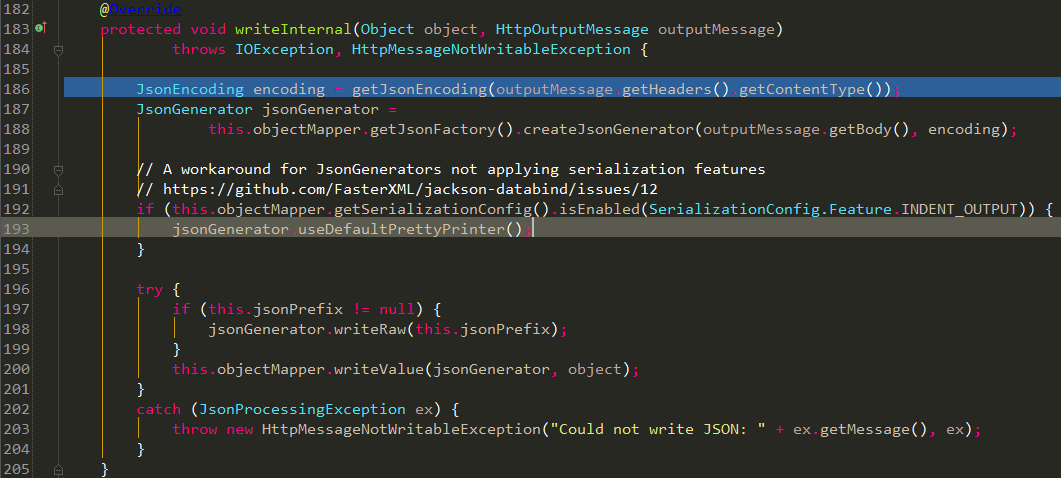
接收的类型：



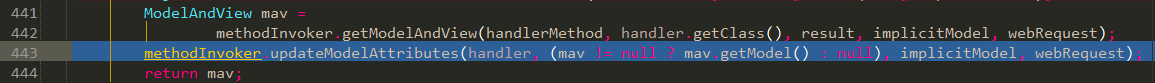
写的方法：



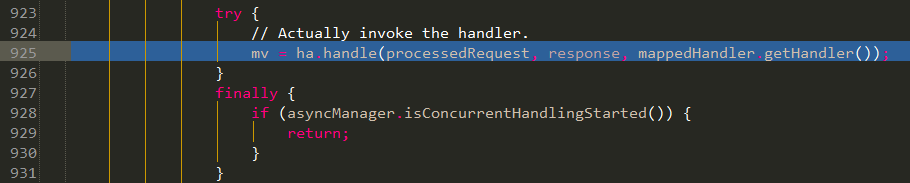
这里配置的json格式，所以会进入JSON方法：



写入ResponseBody后，返回mav：



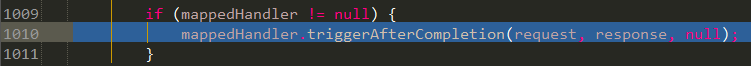
最后返回之前调用handler的地方，之后会有一些不同种类的拦截器方法：



拦截器如：

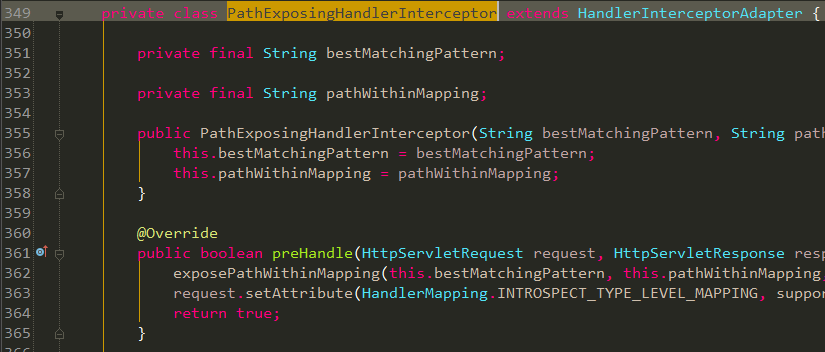
http://img.blog.csdn.net/20140310212215156

http://img.blog.csdn.net/20140310212220828

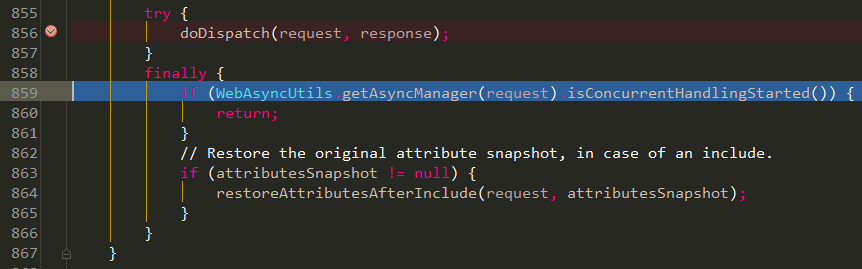


还有一些处理不同异常情况的拦截器。

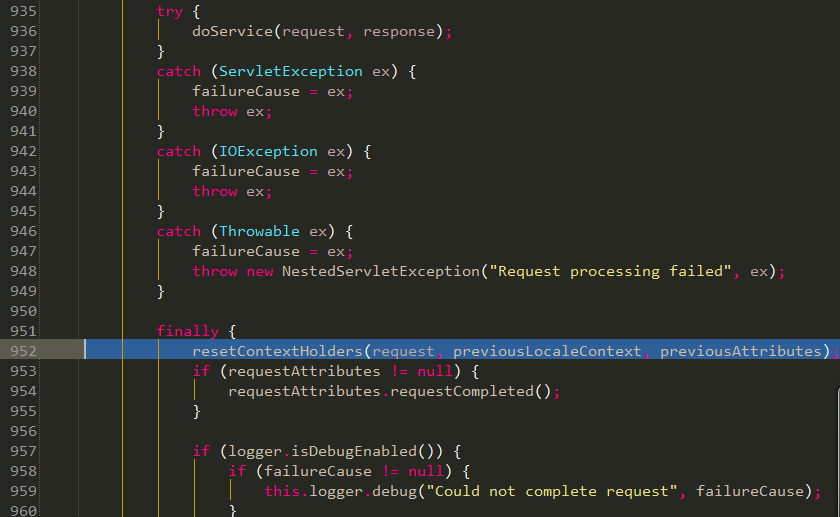
系统默认都会去执行一个拦截器，这个拦截器基本上都是空方法，是一个private类：



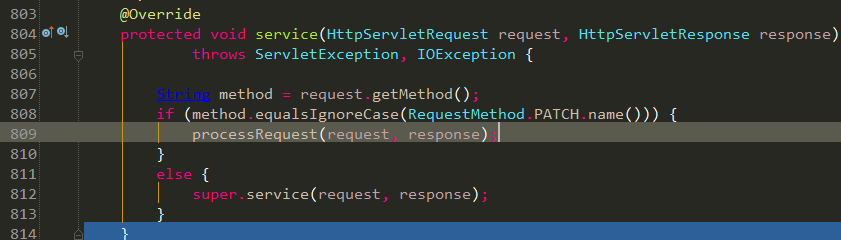
返回调用doDIspatch的地方：



返回到doService的地方：



最后回到httpservlet的service方法：



到这里就完成了一次完整的调用过程。

可以发现，整个过程的流程是比较清晰，程序启动时会根据mvc的配置和spring配置来处理配置信息和注解的类。

Servlet处理请求，通过request（主要是url）来获取handler，之后最主要的一个部分就是获取需要注入的参数，最后调用用户方法，处理返回结果。

整个过程中麻烦的地方就是在一些细节的处理上，这些细节未必一开始就有的，一开始应该是一个主要的流程，后续发现问题或者为了通用性做的改进。

# [Spring—DispatcherServlet的详解](http://blog.csdn.net/jbjwpzyl3611421/article/details/18728227)

2014-01-24 09:46 3144人阅读 [评论](http://blog.csdn.net/jbjwpzyl3611421/article/details/18728227#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/jbjwpzyl3611421/article/details/18728227#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

编程语言（97） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg SPRING（19） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg JAVA编程（88） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

目录[(?)[+]](http://blog.csdn.net/jbjwpzyl3611421/article/details/18728227)

学习Servlet编程，就是继承HttpServlet，根据需要重写一下doGet，doPost方法，跳转到定义好的jsp页面。Servlet类编写完之后在web.xml里注册这个Servlet类。

启动web服务器，在浏览器中输入地址，就可以看到浏览器上输出写好的页面。为了更好的理解上面这个过程，需要学习关于Servlet生命周期的三个阶段，就是所谓的“init-service-destroy”。

以上的知识，对于理解Spring MVC的设计思想，已经足够。Spring MVC当然可以称得上是一个复杂的框架，但是同时它又遵循Servlet世界里最简单的法则，那就是“init-service-destroy”。对于分析SpringMVC的初始化流程，其实就是分析DispatcherServlet类的init()方法。

## DispatcherServlet作用

DispatcherServlet是前端控制器设计模式的实现，提供Spring Web MVC的集中访问点，而且负责职责的分派，而且与Spring IoC容器无缝集成，从而可以获得Spring的所有好处。

DispatcherServlet主要用作职责调度工作，本身主要用于控制流程，主要职责如下：

1、文件上传解析，如果请求类型是multipart将通过MultipartResolver进行文件上传解析；

2、通过HandlerMapping，将请求映射到处理器（返回一个HandlerExecutionChain，它包括一个处理器、多个HandlerInterceptor拦截器）；

3、  通过HandlerAdapter支持多种类型的处理器(HandlerExecutionChain中的处理器)；

4、通过ViewResolver解析逻辑视图名到具体视图实现；

5、本地化解析；

6、渲染具体的视图等；

7、如果执行过程中遇到异常将交给HandlerExceptionResolver来解析。

从以上可以看出DispatcherServlet主要负责流程的控制（而且在流程中的每个关键点都是很容易扩展的）

## DispatcherServlet在web.xml中的配置

**[html]** [view plaincopy](http://blog.csdn.net/java_wliang/article/details/17953465)[在CODE上查看代码片](https://code.csdn.net/snippets/145376)

1. **<servlet>**
2. **<servlet-name>**dispatcher**</servlet-name>**
3. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
4. **<init-param>**
5. **<param-name>**contextConfigLocation**</param-name>**
6. **<param-value>**/WEB-INF/classes/SpringDispatcherServlet.xml**</param-value>**
7. **</init-param>**
8. **<load-on-startup>**1**</load-on-startup>**
9. **</servlet>**
10. **<servlet-mapping>**
11. **<servlet-name>**dispatcher**</servlet-name>**
12. **<url-pattern>**\*.do**</url-pattern>**
13. **</servlet-mapping>**

**load-on-startup：**表示启动容器时初始化该Servlet；

**url-pattern：**表示哪些请求交给Spring Web MVC处理， “/” 是用来定义默认servlet映射的。也可以如“\*.html”表示拦截所有以html为扩展名的请求。

该DispatcherServlet默认使用WebApplicationContext作为上下文，Spring默认配置文件为“/WEB-INF/[servlet名字]-servlet.xml”。

DispatcherServlet也可以配置自己的初始化参数，覆盖默认配置：

摘自Spring Reference

|  |  |
| --- | --- |
| **参数** | **描述** |
| contextClass | 实现WebApplicationContext接口的类，当前的servlet用它来创建上下文。如果这个参数没有指定， 默认使用XmlWebApplicationContext。 |
| contextConfigLocation | 传给上下文实例（由contextClass指定）的字符串，用来指定上下文的位置。这个字符串可以被分成多个字符串（使用逗号作为分隔符） 来支持多个上下文（在多上下文的情况下，如果同一个bean被定义两次，后面一个优先）。 |
| namespace | WebApplicationContext命名空间。默认值是[server-name]-servlet。 |

如果使用如上配置，Spring Web MVC框架通过contextConfigLocation参数的情况来加载DispatcherServlet的配置文件信息

**[html]** [view plaincopy](http://blog.csdn.net/java_wliang/article/details/17953465)[在CODE上查看代码片](https://code.csdn.net/snippets/145376)

1. **<context-param>**
2. **<param-name>**contextConfigLocation**</param-name>**
3. **<param-value>**
4. classpath:spring-common-config.xml,
5. classpath:spring-budget-config.xml
6. **</param-value>**
7. **</context-param>**
8. **<listener>**
9. **<listener-class>**
10. org.springframework.web.context.ContextLoaderListener
11. **</listener-class>**
12. **</listener>**

如上配置是Spring集成Web环境的通用配置；一般用于加载除Web层的Bean（如DAO、Service等），以便于与其他任何Web框架集成。

contextConfigLocation：表示用于加载Bean的配置文件；

contextClass：表示用于加载Bean的ApplicationContext实现类，默认WebApplicationContext。

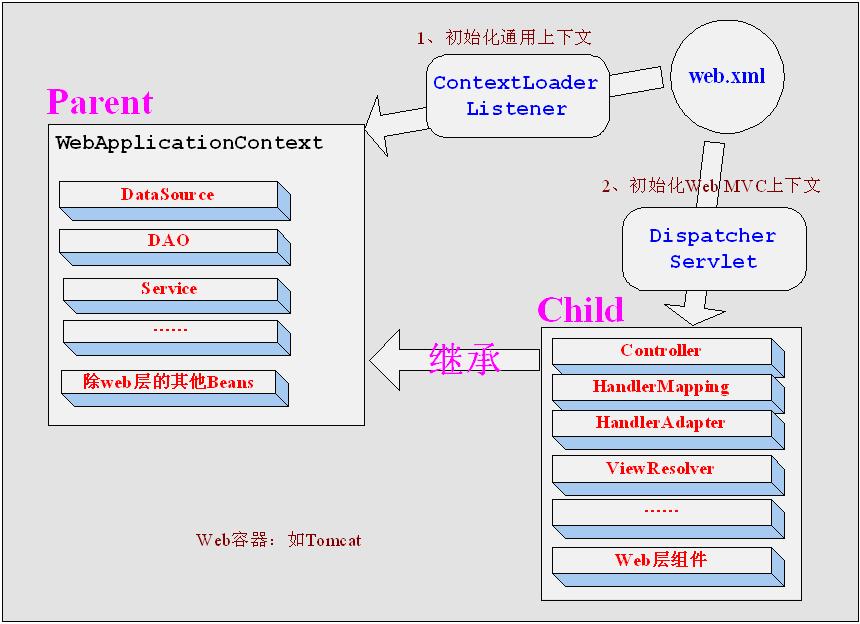
创建完毕后会将该上下文放在ServletContext：

servletContext.setAttribute(

WebApplicationContext.ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE,

this.context);

**ContextLoaderListener初始化的上下文和DispatcherServlet初始化的上下文关系，如图**



从图中可以看出：

ContextLoaderListener初始化的上下文加载的Bean是对于整个应用程序共享的，不管是使用什么表现层技术，一般如DAO层、Service层Bean；

DispatcherServlet初始化的上下文加载的Bean是只对Spring Web MVC有效的Bean，如Controller、HandlerMapping、HandlerAdapter等等，该初始化上下文应该只加载Web相关组件

## DispatcherServlet初始化顺序

**继承体系结构如下所示：**



1、**HttpServletBean继承HttpServlet，**因此在Web容器启动时将调用它的init方法，该初始化方法的主要作用

将Servlet初始化参数（init-param）设置到该组件上（如contextAttribute、contextClass、namespace、contextConfigLocation），通过BeanWrapper简化设值过程，方便后续使用；

提供给子类初始化扩展点，initServletBean()，该方法由FrameworkServlet覆盖。

**2、FrameworkServlet继承HttpServletBean，**通过initServletBean()进行Web上下文初始化，该方法主要覆盖一下两件事情：

    初始化web上下文；

    提供给子类初始化扩展点；

3、DispatcherServlet继承FrameworkServlet，并实现了onRefresh()方法提供一些前端控制器相关的配置：

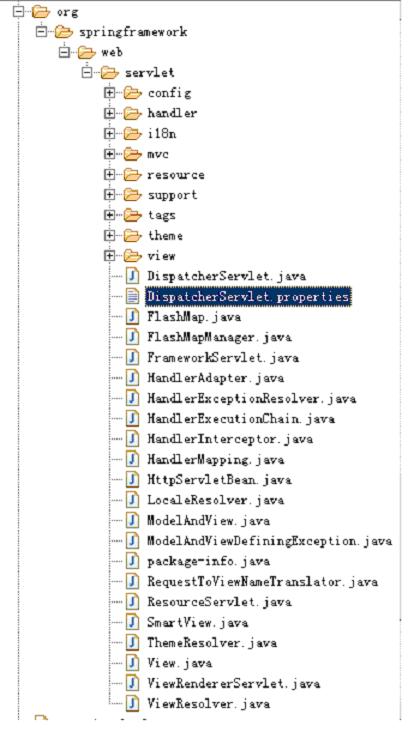
可以看出，整个DispatcherServlet初始化的过程和做了些什么事情，具体主要做了如下两件事情：

1、初始化Spring Web MVC使用的Web上下文，并且可能指定父容器为（ContextLoaderListener加载了根上下文）；

2、初始化DispatcherServlet使用的策略，如HandlerMapping、HandlerAdapter等。

## DispatcherServlet默认配置

DispatcherServlet的默认配置在DispatcherServlet.properties（和DispatcherServlet类在一个包下）中，而且是当Spring配置文件中没有指定配置时使用的默认策略：



org.springframework.web.servlet.LocaleResolver=org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver

org.springframework.web.servlet.ThemeResolver=org.springframework.web.servlet.theme.FixedThemeResolver

org.springframework.web.servlet.HandlerMapping=org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping,\

    org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping

org.springframework.web.servlet.HandlerAdapter=org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter,\

    org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter,\

    org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter

org.springframework.web.servlet.HandlerExceptionResolver=org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerExceptionResolver,\

    org.springframework.web.servlet.mvc.annotation.ResponseStatusExceptionResolver,\

    org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver

org.springframework.web.servlet.RequestToViewNameTranslator=org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator

org.springframework.web.servlet.ViewResolver=org.springframework.web.servlet.view.InternalResourceViewResolver

org.springframework.web.servlet.FlashMapManager=org.springframework.web.servlet.support.SessionFlashMapManager

从如上配置可以看出DispatcherServlet在启动时会自动注册这些特殊的Bean，无需我们注册，如果我们注册了，默认的将不会注册。

因此如第二章的BeanNameUrlHandlerMapping、SimpleControllerHandlerAdapter是不需要注册的，DispatcherServlet默认会注册这两个Bean。

从DispatcherServlet.properties可以看出有许多特殊的Bean，那接下来我们就看看Spring Web MVC主要有哪些特殊的Bean。

## DispatcherServlet中使用的特殊的Bean

DispatcherServlet默认使用WebApplicationContext作为上下文，因此我们来看一下该上下文中有哪些特殊的Bean：

**1、Controller：**处理器/页面控制器，做的是MVC中的C的事情，但控制逻辑转移到前端控制器了，用于对请求进行处理；

**2、HandlerMapping：**请求到处理器的映射，如果映射成功返回一个HandlerExecutionChain对象（包含一个Handler处理器（页面控制器）对象、多个HandlerInterceptor拦截器）对象；如BeanNameUrlHandlerMapping将URL与Bean名字映射，映射成功的Bean就是此处的处理器；

**3、HandlerAdapter：**HandlerAdapter将会把处理器包装为适配器，从而支持多种类型的处理器，即适配器设计模式的应用，从而很容易支持很多类型的处理器；如SimpleControllerHandlerAdapter将对实现了Controller接口的Bean进行适配，并且掉处理器的handleRequest方法进行功能处理；

**4、ViewResolver：**ViewResolver将把逻辑视图名解析为具体的View，通过这种策略模式，很容易更换其他视图技术；如InternalResourceViewResolver将逻辑视图名映射为jsp视图；

**5、LocalResover：**本地化解析，因为Spring支持国际化，因此LocalResover解析客户端的Locale信息从而方便进行国际化；

**6、ThemeResovler：**主题解析，通过它来实现一个页面多套风格，即常见的类似于软件皮肤效果；

**7、MultipartResolver：**文件上传解析，用于支持文件上传；

**8、HandlerExceptionResolver：**处理器异常解析，可以将异常映射到相应的统一错误界面，从而显示用户友好的界面（而不是给用户看到具体的错误信息）；

**9、RequestToViewNameTranslator：**当处理器没有返回逻辑视图名等相关信息时，自动将请求URL映射为逻辑视图名；

**10、FlashMapManager：**用于管理FlashMap的策略接口，FlashMap用于存储一个请求的输出，当进入另一个请求时作为该请求的输入，通常用于重定向场景，后边会细述。