**3.1、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。 具体请参考第二章的图2-1。

  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。 |

从以上我们可以看出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）。

**3.2、DispatcherServlet在web.xml中的配置**

<url-pattern>/</url-pattern>

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：

<init-param>

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

<param-value>classpath:spring-servlet-config.xml</param-value>

</init-param>

|  |  |
| --- | --- |
| **参数** | **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。 |

**3.3、上下文关系**

集成Web环境的通用配置：

<context-param>

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

<param-value>

classpath:spring-common-config.xml,

classpath:spring-budget-config.xml

</param-value>

</context-param>

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

</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3-1**

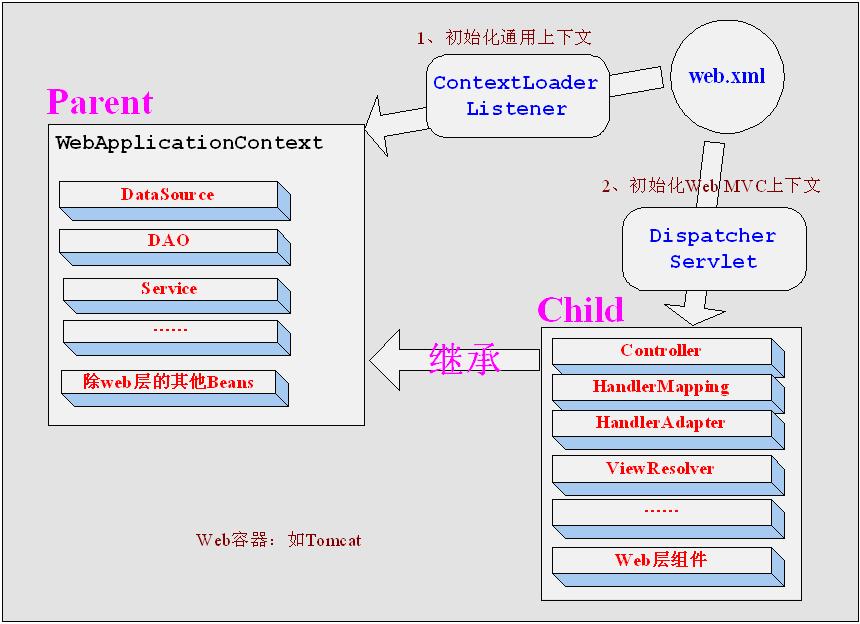


图3-1

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。

**3.4、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：

|  |
| --- |
| // 省略日志及异常代码  **publicfinalvoid** init() **throws** ServletException {  //1、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；  **try** {  PropertyValues pvs = **new** ServletConfigPropertyValues(getServletConfig(), **this**.requiredProperties);  BeanWrapper bw = PropertyAccessorFactory.forBeanPropertyAccess(**this**);  ResourceLoader resourceLoader = **new** ServletContextResourceLoader(getServletContext());  bw.registerCustomEditor(Resource.**class**, **new** ResourceEditor(resourceLoader, **this**.environment));  initBeanWrapper(bw);  bw.setPropertyValues(pvs, **true**);  }  //2、提供给子类初始化的扩展点，该方法由FrameworkServlet覆盖  initServletBean();  } |

**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；

|  |
| --- |
| // 省略日志及异常代码  **protectedfinalvoid** initServletBean() **throws** ServletException {  **try** {  //1、初始化Web上下文  **this**.webApplicationContext = initWebApplicationContext();  //2、提供给子类初始化的扩展点  initFrameworkServlet();  }  } |
| **protected**WebApplicationContext initWebApplicationContext() {  //ROOT上下文（ContextLoaderListener加载的）  WebApplicationContextrootContext = WebApplicationContextUtils.getWebApplicationContext(getServletContext());  WebApplicationContextwac = **null**;  **if** (**this**.webApplicationContext != **null**) {  // 1、在创建该Servlet注入的上下文  wac = **this**.webApplicationContext;  **if** (wac**instanceof**ConfigurableWebApplicationContext) {  ConfigurableWebApplicationContextcwac = (ConfigurableWebApplicationContext) wac;  **if** (!(cwac.isActive())) {  **if** (cwac.getParent() == **null**) {  cwac.setParent(rootContext);  }  configureAndRefreshWebApplicationContext(cwac);  }  }  }  //2、查找已经绑定的上下文  **if** (wac == **null**) {  wac = findWebApplicationContext();  }  //3、如果没有找到相应的上下文，则指定父亲为ContextLoaderListener  **if** (wac == **null**) {  wac = createWebApplicationContext(rootContext);  }  //4、刷新上下文（执行容器的一些初始化）,这个方法由子类实现，来进行扩展  **if** (!(**this**.refreshEventReceived)) {  onRefresh(wac);  }  **if** (**this**.publishContext) {  String attrName = getServletContextAttributeName();  getServletContext().setAttribute(attrName, wac);  }  **return**wac;  } |

从initWebApplicationContext（）方法可以看出，基本上如果ContextLoaderListener加载了上下文将作为根上下文（DispatcherServlet的父容器）。

**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：

|  |
| --- |
| **protectedvoid** onRefresh(ApplicationContext context) {  initStrategies(context);  }  //初始化默认的Spring Web MVC框架使用的策略（如HandlerMapping）  **protectedvoid** initStrategies(ApplicationContext context) {  initMultipartResolver(context);  initLocaleResolver(context);  initThemeResolver(context);  initHandlerMappings(context);  initHandlerAdapters(context);  initHandlerExceptionResolvers(context);  initRequestToViewNameTranslator(context);  initViewResolvers(context);  initFlashMapManager(context);  } |

 从如上代码可以看出，DispatcherServlet启动时会进行我们需要的Web层Bean的配置，如HandlerMapping、HandlerAdapter等，而且如果我们没有配置，还会给我们提供默认的配置。

从如上代码我们可以看出DispatcherServlet初始化的整个过程中做了些什么事情，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。

**服务器启动时的日志分析**（此处加上了ContextLoaderListener从而启动ROOT上下文容器）：

|  |
| --- |
| 信息: Initializing Spring root WebApplicationContext **//由ContextLoaderListener启动ROOT上下文**    2012-03-12 13:33:55 [main] INFO  **org.springframework.web.context.ContextLoader** - Root WebApplicationContext: initialization started  2012-03-12 13:33:55 [main] INFO  org.springframework.web.context.support.XmlWebApplicationContext - Refreshing Root WebApplicationContext: startup date [Mon Mar 12 13:33:55 CST 2012]; root of context hierarchy  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.DefaultBeanDefinitionDocumentReader - Loading bean definitions  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 0 bean definitions from location pattern **[/WEB-INF/ContextLoaderListener.xml]**  2012-03-12 13:33:55 [main] DEBUG org.springframework.web.context.support.XmlWebApplicationContext - Bean factory for Root WebApplicationContext: org.springframework.beans.factory.support.DefaultListableBeanFactory@1c05ffd: defining beans []; root of factory hierarchy  2012-03-12 13:33:55 [main] DEBUG org.springframework.web.context.support.XmlWebApplicationContext - Bean factory for Root WebApplicationContext:  **2012-03-12 13:33:55 [main] DEBUG org.springframework.web.context.ContextLoader - Published root WebApplicationContext as ServletContext attribute with name [org.springframework.web.context.WebApplicationContext.ROOT] //将ROOT上下文绑定到ServletContext**  2012-03-12 13:33:55 [main] INFO  org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization completed in 438 ms**//到此ROOT上下文启动完毕**     2012-03-12 13:33:55 [main] DEBUG org.springframework.web.servlet.DispatcherServlet - Initializing servlet 'chapter2'  信息: Initializing Spring FrameworkServlet 'chapter2'  **//开始初始化FrameworkServlet对应的Web上下文**  2012-03-12 13:33:55 [main] INFO  org.springframework.web.servlet.DispatcherServlet - FrameworkServlet 'chapter2': initialization started  2012-03-12 13:33:55 [main] DEBUG org.springframework.web.servlet.DispatcherServlet - Servlet with name 'chapter2' will try to**create custom WebApplicationContext context** of class 'org.springframework.web.context.support.XmlWebApplicationContext', **using parent context [Root WebApplicationContext**: startup date [Mon Mar 12 13:33:55 CST 2012]; root of context hierarchy]  **//此处使用Root WebApplicationContext作为父容器。**  2012-03-12 13:33:55 [main] INFO  org.springframework.web.context.support.XmlWebApplicationContext - Refreshing WebApplicationContext for namespace 'chapter2-servlet': startup date [Mon Mar 12 13:33:55 CST 2012]; parent: Root WebApplicationContext  2012-03-12 13:33:55 [main] INFO  org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from ServletContext resource [**/WEB-INF/chapter2-servlet.xml]**  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.DefaultBeanDefinitionDocumentReader - Loading bean definitions  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.BeanDefinitionParserDelegate - Neither XML 'id' nor 'name' specified - using generated bean name**[org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping#0]  //我们配置的HandlerMapping**  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.BeanDefinitionParserDelegate - Neither XML 'id' nor 'name' specified - using generated bean name**[org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter#0] //我们配置的HandlerAdapter**  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.BeanDefinitionParserDelegate - Neither XML 'id' nor 'name' specified - using generated bean name [org.springframework.web.servlet.view.InternalResourceViewResolver#0] **//我们配置的ViewResolver**  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.BeanDefinitionParserDelegate - No XML 'id' specified - using '/hello' as bean name and [] as aliases  **//我们的处理器（HelloWorldController）**  2012-03-12 13:33:55 [main] DEBUG org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loaded 4 bean definitions from location pattern [/WEB-INF/chapter2-servlet.xml]  2012-03-12 13:33:55 [main] DEBUG org.springframework.web.context.support.XmlWebApplicationContext - Bean factory for WebApplicationContext for namespace 'chapter2-servlet': org.springframework.beans.factory.support.DefaultListableBeanFactory@1372656: defining beans [org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping#0,org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter#0,org.springframework.web.servlet.view.InternalResourceViewResolver#0,/hello]; parent: org.springframework.beans.factory.support.DefaultListableBeanFactory@1c05ffd  **//到此容器注册的Bean初始化完毕**    2012-03-12 13:33:56 [main] DEBUG org.springframework.web.servlet.DispatcherServlet - Unable to locate MultipartResolver with name 'multipartResolver': no multipart request handling provided    2012-03-12 13:33:56 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating instance of bean 'org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver'  **//默认的LocaleResolver注册**  2012-03-12 13:33:56 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating instance of bean 'org.springframework.web.servlet.theme.FixedThemeResolver'  **//默认的ThemeResolver注册**    2012-03-12 13:33:56 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping#0'  **//发现我们定义的HandlerMapping 不再使用默认的HandlerMapping。**    2012-03-12 13:33:56 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter#0'  **//发现我们定义的HandlerAdapter 不再使用默认的HandlerAdapter。**    2012-03-12 13:33:56 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating instance of bean 'org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerExceptionResolver'  **//异常处理解析器ExceptionResolver**  2012-03-12 13:33:56 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating instance of bean 'org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerExceptionResolver'    2012-03-12 13:33:56 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Returning cached instance of singleton bean 'org.springframework.web.servlet.view.InternalResourceViewResolver#0'    2012-03-12 13:33:56 [main] DEBUG org.springframework.web.servlet.DispatcherServlet - Published WebApplicationContext of servlet 'chapter2' as ServletContext attribute with name [org.springframework.web.servlet.FrameworkServlet.CONTEXT.chapter2]  **//绑定FrameworkServlet初始化的Web上下文到ServletContext**  2012-03-12 13:33:56 [main] INFO  org.springframework.web.servlet.DispatcherServlet - FrameworkServlet 'chapter2': initialization completed in  297 ms  2012-03-12 13:33:56 [main] DEBUG org.springframework.web.servlet.DispatcherServlet - Servlet 'chapter2' configured successfully  **//到此完整流程结束** |

从如上日志我们也可以看出，DispatcherServlet会进行一些默认的配置。接下来我们看一下默认配置吧。

**3.5、DispatcherServlet默认配置**

DispatcherServlet的默认配置在**DispatcherServlet.properties**（和DispatcherServlet类在同一个包下）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。

**3.6、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。

到此DispatcherServlet我们已经了解了，接下来我们就需要把上边提到的特殊Bean挨个击破，那首先从控制器开始吧。