[dispatcherservlet与ContextLoaderListener详解](http://blog.csdn.net/lqx1988221/article/details/12620689)

2013-10-11 17:22 22430人阅读 [评论](http://blog.csdn.net/lqx1988221/article/details/12620689#comments)(6) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/lqx1988221/article/details/12620689#report)

目录

1. [DispatcherServlet作用](http://blog.csdn.net/lqx1988221/article/details/12620689#t0)
2. [DispatcherServlet在webxml中的配置](http://blog.csdn.net/lqx1988221/article/details/12620689#t1)
3. [上下文关系](http://blog.csdn.net/lqx1988221/article/details/12620689#t2)
4. [DispatcherServlet初始化顺序](http://blog.csdn.net/lqx1988221/article/details/12620689#t3)

**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**

Java代码  收藏代码

1. <servlet>
2. <servlet-name>chapter2</servlet-name>
3. <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4. <load-on-startup>1</load-on-startup>
5. </servlet>
6. <servlet-mapping>
7. <servlet-name>chapter2</servlet-name>
8. <url-pattern>/</url-pattern>
9. </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。 |

因此我们可以通过添加初始化参数

Java代码  收藏代码

1. <servlet>
2. <servlet-name>chapter2</servlet-name>
3. <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4. <load-on-startup>1</load-on-startup>
5. <init-param>
6. <param-name>contextConfigLocation</param-name>
7. <param-value>classpath:spring-servlet-config.xml</param-value>
8. </init-param>
9. </servlet>

如果使用如上配置，Spring Web MVC框架将加载“classpath:spring-servlet-config.xml”来进行初始化上下文而不是“/WEB-INF/[servlet名字]-servlet.xml”。

**3.3、上下文关系**

集成Web环境的通用配置：

Java代码  收藏代码

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>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
9. </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**

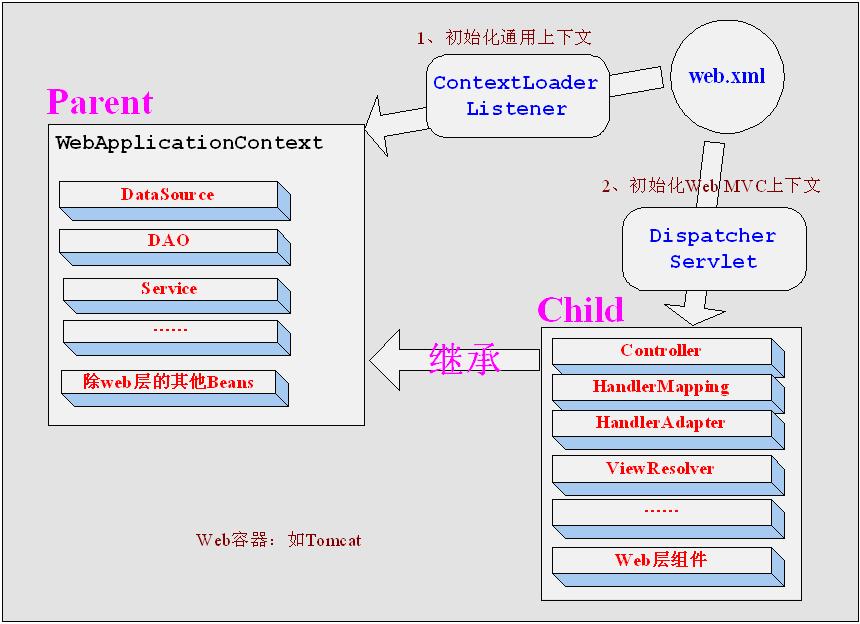


图3-1

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

：：：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。

Java代码  收藏代码

1. public abstract class HttpServletBean extends HttpServlet implements EnvironmentAware{
2. @Override
3. public final void init() throws ServletException {
4. //省略部分代码
5. //1、如下代码的作用是将Servlet初始化参数设置到该组件上
6. //如contextAttribute、contextClass、namespace、contextConfigLocation；
7. try {
8. PropertyValues pvs = new ServletConfigPropertyValues(getServletConfig(), this.requiredProperties);
9. BeanWrapper bw = PropertyAccessorFactory.forBeanPropertyAccess(this);
10. ResourceLoader resourceLoader = new ServletContextResourceLoader(getServletContext());
11. bw.registerCustomEditor(Resource.class, new ResourceEditor(resourceLoader, this.environment));
12. initBeanWrapper(bw);
13. bw.setPropertyValues(pvs, true);
14. }
15. catch (BeansException ex) {
16. //…………省略其他代码
17. }
18. //2、提供给子类初始化的扩展点，该方法由FrameworkServlet覆盖
19. initServletBean();
20. if (logger.isDebugEnabled()) {
21. logger.debug("Servlet '" + getServletName() + "' configured successfully");
22. }
23. }
24. //…………省略其他代码
25. }

**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；

Java代码  收藏代码

1. public abstract class FrameworkServlet extends HttpServletBean {
2. @Override
3. protected final void initServletBean() throws ServletException {
4. //省略部分代码
5. try {
6. //1、初始化Web上下文
7. this.webApplicationContext = initWebApplicationContext();
8. //2、提供给子类初始化的扩展点
9. initFrameworkServlet();
10. }
11. //省略部分代码
12. }
13. }

Java代码  收藏代码

1. protected WebApplicationContext initWebApplicationContext() {
2. //ROOT上下文（ContextLoaderListener加载的）
3. WebApplicationContext rootContext =
4. WebApplicationContextUtils.getWebApplicationContext(getServletContext());
5. WebApplicationContext wac = null;
6. if (this.webApplicationContext != null) {
7. // 1、在创建该Servlet注入的上下文
8. wac = this.webApplicationContext;
9. if (wac instanceof ConfigurableWebApplicationContext) {
10. ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) wac;
11. if (!cwac.isActive()) {
12. if (cwac.getParent() == null) {
13. cwac.setParent(rootContext);
14. }
15. configureAndRefreshWebApplicationContext(cwac);
16. }
17. }
18. }
19. if (wac == null) {
20. //2、查找已经绑定的上下文
21. wac = findWebApplicationContext();
22. }
23. if (wac == null) {
24. //3、如果没有找到相应的上下文，并指定父亲为ContextLoaderListener
25. wac = createWebApplicationContext(rootContext);
26. }
27. if (!this.refreshEventReceived) {
28. //4、刷新上下文（执行一些初始化）
29. onRefresh(wac);
30. }
31. if (this.publishContext) {
32. // Publish the context as a servlet context attribute.
33. String attrName = getServletContextAttributeName();
34. getServletContext().setAttribute(attrName, wac);
35. //省略部分代码
36. }
37. return wac;
38. }

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）。

最后调用了onRefresh()方法执行容器的一些初始化，这个方法由子类实现，来进行扩展。

**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：

Java代码  收藏代码

1. public class DispatcherServlet extends FrameworkServlet {
2. //实现子类的onRefresh()方法，该方法委托为initStrategies()方法。
3. @Override
4. protected void onRefresh(ApplicationContext context) {
5. initStrategies(context);
6. }
8. //初始化默认的Spring Web MVC框架使用的策略（如HandlerMapping）
9. protected void initStrategies(ApplicationContext context) {
10. initMultipartResolver(context);
11. initLocaleResolver(context);
12. initThemeResolver(context);
13. initHandlerMappings(context);
14. initHandlerAdapters(context);
15. initHandlerExceptionResolvers(context);
16. initRequestToViewNameTranslator(context);
17. initViewResolvers(context);
18. initFlashMapManager(context);
19. }
20. }

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，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。

 转载出处：http://www.iteye.com/topic/1125421