<https://www.cnblogs.com/wkrbky/p/5929943.html>

# [Spring MVC的web.xml配置详解(转)](http://www.cnblogs.com/wkrbky/p/5929943.html)

出处http://blog.csdn.net/u010796790

1、[spring](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "_blank) 框架解决字符串编码问题：过滤器 **CharacterEncodingFilter**（filter-name）   
2、在web.xml配置监听器**ContextLoaderListener**（listener-class）   
ContextLoaderListener的作用就是启动Web容器时，自动装配ApplicationContext的配置信息。因为它实现了ServletContextListener这个接口，在web.xml配置这个监听器，启动容器时，就会默认执行它实现的方法。   
3、部署applicationContext的xml文件：**contextConfigLocation**（context-param下的param-name）   
4、DispatcherServlet是前置控制器，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据某某规则分发到目标Controller(我们写的Action)来处理。   
**DispatcherServlet**（servlet-name、servlet-class、init-param、param-name(contextConfigLocation)、param-value）   
在DispatcherServlet的初始化过程中，框架会在web应用的 WEB-INF文件夹下寻找名为[servlet-name]-servlet.xml 的配置文件，生成文件中定义的bean

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd">

<!-- 在Spring框架中是如何解决从页面传来的字符串的编码问题的呢？

下面我们来看看Spring框架给我们提供过滤器CharacterEncodingFilter

这个过滤器就是针对于每次浏览器请求进行过滤的，然后再其之上添加了父类没有的功能即处理字符编码。

其中encoding用来设置编码格式，forceEncoding用来设置是否理会 request.getCharacterEncoding()方法，设置为true则强制覆盖之前的编码格式。-->

<filter>

<filter-name>characterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 项目中使用Spring 时，applicationContext.xml配置文件中并没有BeanFactory，要想在业务层中的class 文件中直接引用Spring容器管理的bean可通过以下方式-->

<!--1、在web.xml配置监听器ContextLoaderListener-->

<!--ContextLoaderListener的作用就是启动Web容器时，自动装配ApplicationContext的配置信息。因为它实现了ServletContextListener这个接口，在web.xml配置这个监听器，启动容器时，就会默认执行它实现的方法。

在ContextLoaderListener中关联了ContextLoader这个类，所以整个加载配置过程由ContextLoader来完成。

它的API说明

第一段说明ContextLoader可以由 ContextLoaderListener和ContextLoaderServlet生成。

如果查看ContextLoaderServlet的API，可以看到它也关联了ContextLoader这个类而且它实现了HttpServlet这个接口

第二段，ContextLoader创建的是 XmlWebApplicationContext这样一个类，它实现的接口是WebApplicationContext->ConfigurableWebApplicationContext->ApplicationContext->

BeanFactory这样一来spring中的所有bean都由这个类来创建

IUploaddatafileManager uploadmanager = (IUploaddatafileManager) ContextLoaderListener.getCurrentWebApplicationContext().getBean("uploadManager");

-->

<listener>

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

</listener>

<!--2、部署applicationContext的xml文件-->

<!--如果在web.xml中不写任何参数配置信息，默认的路径是"/WEB-INF/applicationContext.xml，

在WEB-INF目录下创建的xml文件的名称必须是applicationContext.xml。

如果是要自定义文件名可以在web.xml里加入contextConfigLocation这个context参数：

在<param-value> </param-value>里指定相应的xml文件名，如果有多个xml文件，可以写在一起并以“,”号分隔。

也可以这样applicationContext-\*.xml采用通配符，比如这那个目录下有applicationContext-ibatis-base.xml，

applicationContext-action.xml，applicationContext-ibatis-dao.xml等文件，都会一同被载入。

在ContextLoaderListener中关联了ContextLoader这个类，所以整个加载配置过程由ContextLoader来完成。-->

<context-param>

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

<param-value>classpath:spring/applicationContext.xml</param-value>

</context-param>

<!--如果你的DispatcherServlet拦截"/"，为了实现REST风格，拦截了所有的请求，那么同时对\*.js,\*.jpg等静态文件的访问也就被拦截了。-->

<!--方案一：激活Tomcat的defaultServlet来处理静态文件-->

<!--要写在DispatcherServlet的前面， 让 defaultServlet先拦截请求，这样请求就不会进入Spring了，我想性能是最好的吧。-->

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.css</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.swf</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.gif</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.jpg</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.png</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.js</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.xml</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.json</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.map</url-pattern>

</servlet-mapping>

<!--使用Spring MVC,配置DispatcherServlet是第一步。DispatcherServlet是一个Servlet,,所以可以配置多个DispatcherServlet-->

<!--DispatcherServlet是前置控制器，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据某某规则分发到目标Controller(我们写的Action)来处理。-->

<servlet>

<servlet-name>DispatcherServlet</servlet-name><!--在DispatcherServlet的初始化过程中，框架会在web应用的 WEB-INF文件夹下寻找名为[servlet-name]-servlet.xml 的配置文件，生成文件中定义的bean。-->

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!--指明了配置文件的文件名，不使用默认配置文件名，而使用dispatcher-servlet.xml配置文件。-->

<init-param>

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

<!--其中<param-value>\*\*.xml</param-value> 这里可以使用多种写法-->

<!--1、不写,使用默认值:/WEB-INF/<servlet-name>-servlet.xml-->

<!--2、<param-value>/WEB-INF/classes/dispatcher-servlet.xml</param-value>-->

<!--3、<param-value>classpath\*:dispatcher-servlet.xml</param-value>-->

<!--4、多个值用逗号分隔-->

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

</init-param>

<load-on-startup>1</load-on-startup><!--是启动顺序，让这个Servlet随Servletp容器一起启动。-->

</servlet>

<servlet-mapping>

<!--这个Servlet的名字是dispatcher，可以有多个DispatcherServlet，是通过名字来区分的。每一个DispatcherServlet有自己的WebApplicationContext上下文对象。同时保存的ServletContext中和Request对象中.-->

<!--ApplicationContext是Spring的核心，Context我们通常解释为上下文环境，我想用“容器”来表述它更容易理解一些，ApplicationContext则是“应用的容器”了:P，Spring把Bean放在这个容器中，在需要的时候，用getBean方法取出-->

<servlet-name>DispatcherServlet</servlet-name>

<!--Servlet拦截匹配规则可以自已定义，当映射为@RequestMapping("/user/add")时，为例,拦截哪种URL合适？-->

<!--1、拦截\*.do、\*.htm， 例如：/user/add.do,这是最传统的方式，最简单也最实用。不会导致静态文件（jpg,js,css）被拦截。-->

<!--2、拦截/，例如：/user/add,可以实现现在很流行的REST风格。很多互联网类型的应用很喜欢这种风格的URL。弊端：会导致静态文件（jpg,js,css）被拦截后不能正常显示。 -->

<url-pattern>/</url-pattern> <!--会拦截URL中带“/”的请求。-->

</servlet-mapping>

<welcome-file-list><!--指定欢迎页面-->

<welcome-file>login.html</welcome-file>

</welcome-file-list>

<error-page> <!--当系统出现404错误，跳转到页面nopage.html-->

<error-code>404</error-code>

<location>/nopage.html</location>

</error-page>

<error-page> <!--当系统出现java.lang.NullPointerException，跳转到页面error.html-->

<exception-type>java.lang.NullPointerException</exception-type>

<location>/error.html</location>

</error-page>

<session-config><!--会话超时配置，单位分钟-->

<session-timeout>360</session-timeout>

</session-config>

</web-app>

[复制代码](javascript:void(0);)