servlet3.0+规范后，允许servlet，filter，listener不必声明在web.xml中，而是以硬编码的方式存在，实现容器的零配置。

****ServletContainerInitializer：启动容器时负责加载相关配置****

package javax.servlet;

import java.util.Set;

public interface ServletContainerInitializer {

public void onStartup(Set<Class<?>> c, ServletContext ctx)

throws ServletException;

}

 容器启动时会自动扫描当前服务中ServletContainerInitializer的实现类，并调用其onStartup方法，其参数Set<Class<?>> c，可通过在实现类上声明注解javax.servlet.annotation.HandlesTypes(xxx.class)注解自动注入，@HandlesTypes会自动扫描项目中所有的xxx.class的实现类，并将其全部注入Set。

Spring为其提供了一个实现类：****SpringServletContainerInitializer****

package org.springframework.web;

import java.lang.reflect.Modifier;

import java.util.LinkedList;

import java.util.List;

import java.util.ServiceLoader;

import java.util.Set;

import javax.servlet.ServletContainerInitializer;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.annotation.HandlesTypes;

import org.springframework.core.annotation.AnnotationAwareOrderComparator;

@HandlesTypes(WebApplicationInitializer.class)

public class SpringServletContainerInitializer implements ServletContainerInitializer {

@Override

public void onStartup(Set<Class<?>> webAppInitializerClasses, ServletContext servletContext)

throws ServletException {

List<WebApplicationInitializer> initializers = new LinkedList<WebApplicationInitializer>();

if (webAppInitializerClasses != null) {

for (Class<?> waiClass : webAppInitializerClasses) {

// Be defensive: Some servlet containers provide us with invalid classes,

// no matter what @HandlesTypes says...

if (!waiClass.isInterface() && !Modifier.isAbstract(waiClass.getModifiers()) &&

WebApplicationInitializer.class.isAssignableFrom(waiClass)) {

try {

initializers.add((WebApplicationInitializer) waiClass.newInstance());

}

catch (Throwable ex) {

throw new ServletException("Failed to instantiate WebApplicationInitializer class", ex);

}

}

}

}

if (initializers.isEmpty()) {

servletContext.log("No Spring WebApplicationInitializer types detected on classpath");

return;

}

AnnotationAwareOrderComparator.sort(initializers);

servletContext.log("Spring WebApplicationInitializers detected on classpath: " + initializers);

for (WebApplicationInitializer initializer : initializers) {

initializer.onStartup(servletContext);

}

}

}

 从中可以看出，WebApplicationInitializer才是我们需要关心的接口，我们只需要将相应的servlet，filter，listener等硬编码到该接口的实现类中即可。比如：

xml配置：

<!-- Log4jConfigListener -->

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>classpath:config/properties/log4j.properties</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

<!-- OpenSessionInViewFilter -->

<filter>

<filter-name>hibernateFilter</filter-name>

<filter-class>

org.springframework.orm.hibernate4.support.OpenSessionInViewFilter

</filter-class>

</filter>

<filter-mapping>

<filter-name>hibernateFilter</filter-name>

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

</filter-mapping>

<!-- DemoServlet -->

<servlet>

<servlet-name>demoServlet</servlet-name>

<servlet-class>web.function.servlet.DemoServlet</servlet-class>

<load-on-startup>2</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>demoServlet</servlet-name>

<url-pattern>/demo\_servlet</url-pattern>

</servlet-mapping>

@Order(1)

public class CommonInitializer implements WebApplicationInitializer{

@Override

public void onStartup(ServletContext servletContext)

throws ServletException {

//Log4jConfigListener

servletContext.setInitParameter("log4jConfigLocation", "classpath:config/properties/log4j.properties");

servletContext.addListener(Log4jConfigListener.class);

//OpenSessionInViewFilter

OpenSessionInViewFilter hibernateSessionInViewFilter = new OpenSessionInViewFilter();

FilterRegistration.Dynamic filterRegistration = servletContext.addFilter(

"hibernateFilter", hibernateSessionInViewFilter);

filterRegistration.addMappingForUrlPatterns(

EnumSet.of(DispatcherType.REQUEST, DispatcherType.FORWARD, DispatcherType.INCLUDE), false, "/");

//DemoServlet

DemoServlet demoServlet = new DemoServlet();

ServletRegistration.Dynamic dynamic = servletContext.addServlet(

"demoServlet", demoServlet);

dynamic.setLoadOnStartup(2);

dynamic.addMapping("/demo\_servlet");

}

}

Spring为我们提供了一些WebApplicationInitializer的抽象类，我们只需要继承并按需修改即可，比如：

****1）org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer ： SpringSecurity相关配置****

xml配置：

<listener>

<listener-class>org.springframework.security.web.session.HttpSessionEventPublisher</listener-class>

</listener>

<filter>

<filter-name>springSecurityFilterChain</filter-name>

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

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

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

</filter-mapping>

编码配置：

@Order(2)

public class WebAppSecurityInitializer extends AbstractSecurityWebApplicationInitializer

{

//servletContext.addListener("org.springframework.security.web.session.HttpSessionEventPublisher");

//session监听器

@Override

protected boolean enableHttpSessionEventPublisher() {

return true;

}

}

****2）org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer：MVC相关配置，比如加载spring配置文件，声明DispatcherServlet等等，参看下面的对比：****

xml配置：

<context-param>

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

<param-value>

classpath:config/context/applicationContext-AppConfig.xml,

classpath:config/context/applicationContext-SpringSecurityConfig.xml

</param-value>

</context-param>

<listener>

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

</listener>

<filter>

<filter-name>Set Character Encoding</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>Set Character Encoding</filter-name>

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

</filter-mapping>

<servlet>

<servlet-name>webmvc</servlet-name>

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

<init-param>

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

<param-value>classpath:config/context/applicationContext-MvcConfig.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>webmvc</servlet-name>

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

</servlet-mapping>

编码方式：

@Order(3)

//spring DispatcherServlet的配置,其它servlet和监听器等需要额外声明，用@Order注解设定启动顺序

public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

/\*

\* DispatcherServlet的映射路径

\*/

@Override

protected String[] getServletMappings() {

return new String[]{"/"};

}

/\*

\* 应用上下文，除web部分

\*/

@SuppressWarnings({ "unchecked", "rawtypes" })

@Override

protected Class[] getRootConfigClasses() {

//加载配置文件类，这里与上面的xml配置是对应的，需要使用@Configuration注解进行标注，稍后介绍

return new Class[] {AppConfig.class, SpringSecurityConfig.class};

}

/\*

\* web上下文

\*/

@SuppressWarnings({ "unchecked", "rawtypes" })

@Override

protected Class[] getServletConfigClasses() {

return new Class[] {MvcConfig.class};

}

/\*

\* 注册过滤器，映射路径与DispatcherServlet一致，路径不一致的过滤器需要注册到另外的WebApplicationInitializer中

\*/

@Override

protected Filter[] getServletFilters() {

CharacterEncodingFilter characterEncodingFilter = new CharacterEncodingFilter();

characterEncodingFilter.setEncoding("UTF-8");

characterEncodingFilter.setForceEncoding(true);

return new Filter[] {characterEncodingFilter};

}

}