# 一、绪论

基于Servlet3.0规范和SpringMVC4注解式配置方式，实现零xml配置，弄了个小demo，供交流讨论。

项目说明如下：

1.db.sql是项目中用到的表，数据库使用的是oracle11g

2.该项目使用mvn进行管理，私服为自搭建nexus,项目只用到一个第三方 jar，就是oracle的驱动；

3.默认项目为零配置启动，如果需要更改启动方式，请作如下操作：

<1.使用xml风格测试时，需要先将/src/main/java/web/config下的全部类移动到项目外，并开启web.xml中的相关配置>

<2.使用Bean风格测试时，需要保证/src/main/java/web/config下的全部类都存在，并关闭web.xml中的相关配置>

4.项目特征如下：

1)项目基于SpringMVC4

2)安全控制使用SpringSecurity3.2

3)Hibernate4 + 注解式事务管理

4)基于ASpect的注解式AOP

5)基于EHCache的注解式缓存

6)JSON及XML例子,@RestController及@Controller使用对比

7)单元测试基于MockMvc

8)配置了较完整的组件，贴近实际应用

5.运行环境：tomcat8.0.9 jdk1.7

这里对SpringMVC零配置做一个简单的说明

spring4中提供了大量的注解来支持零配置，简要说明如下：

@Configuration ： 类似于spring配置文件，负责注册bean，对应的提供了@Bean注解。需要

org.springframework.web.context.support.AnnotationConfigWebApplicationContext注册到容器中。

@ComponentScan ： 注解类查找规则定义 <context:component-scan/>

@EnableAspectJAutoProxy ： 激活Aspect自动代理 <aop:aspectj-autoproxy/>

@Import @ImportResource: 关联其它spring配置 <import resource="" />

@EnableCaching ：启用缓存注解 <cache:annotation-driven/>

@EnableTransactionManagement ： 启用注解式事务管理 <tx:annotation-driven />

@EnableWebMvcSecurity ： 启用springSecurity安全验证

Servlet3.0规范，支持将web.xml相关配置也硬编码到代码中[servlet，filter，listener,等等]，并由javax.servlet.ServletContainerInitializer的实现类负责

在容器启动时进行加载，

spring提供了一个实现类org.springframework.web.SpringServletContainerInitializer,

该类会调用所有org.springframework.web.WebApplicationInitializer的实现类的onStartup(ServletContext servletContext)方法，将相关的组件注册到服务器；

spring同时提供了一些WebApplicationInitializer的实现类供我们继承，以简化相关的配置，比如：

org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer ： 注册spring DispatcherServlet

org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer ： 注册springSecurity

同时，spring也提供了一些@Configuration的支持类供我们继承，以简化相关@Configuration的配置，比如：

org.springframework.web.servlet.config.annotation.WebMvcConfigurationSupport：封装了springmvc相关组件，我们可以通过注册新的@Bean和@Override相

关方法，以实现对各个组件的注册；

org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter：封装类springsecurity相关组件

# 二、web.xml文件配置说明

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.java：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.java

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>

编码配置：

@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};

}

}

# 三、应用上下文配置

从spring3.0开始，Spring将JavaConfig整合到核心模块，普通的POJO只需要标注@Configuration注解，就可以成为spring配置类，并通过在方法上标注@Bean注解的方式注入bean。

Xml配置和Java类配置对比如下：

applicationContext-AppConfig.xml

<!-- 激活自动代理功能 参看：web.function.aop.aspect.DemoAspect -->

<aop:aspectj-autoproxy proxy-target-class="true" />

<context:component-scan base-package="web.function">

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

<import resource="classpath:config/context/applicationContext-CachingConfig.xml" />

<import resource="classpath:config/context/applicationContext-DaoConfig.xml" />

AppConfig.java

@Configuration

@ComponentScan(basePackages = "web.function", excludeFilters = { @ComponentScan.Filter(type = FilterType.ANNOTATION, value = { Controller.class }) })

@EnableAspectJAutoProxy(proxyTargetClass=true)

@Import({CachingConfig.class,DaoConfig.class})

public class AppConfig {

}

applicationContext-CachingConfig.xml

<!-- 启用缓存注解 -->

<cache:annotation-driven cache-manager="cacheManager" />

<!-- cacheManager工厂类，指定ehcache.xml的位置 -->

<bean id="cacheManagerFactory"

class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"

p:configLocation="classpath:/config/ehcache.xml" />

<!-- 声明cacheManager -->

<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager"

p:cacheManager-ref="cacheManagerFactory" />

CachingConfig.java

@Configuration

@EnableCaching//<!-- 启用缓存注解 --> <cache:annotation-driven cache-manager="cacheManager" />

public class CachingConfig {

private static final Logger logger = Logger.getLogger(CachingConfig.class);

@Bean

public EhCacheManagerFactoryBean ehCacheManagerFactoryBean() {

EhCacheManagerFactoryBean ehCacheManagerFactoryBean = new EhCacheManagerFactoryBean();

ehCacheManagerFactoryBean.setConfigLocation(new ClassPathResource(

"config/ehcache.xml"));

return ehCacheManagerFactoryBean;

}

@Bean

public CacheManager cacheManager() {

logger.info("EhCacheCacheManager");

EhCacheCacheManager cacheManager = new EhCacheCacheManager();

cacheManager.setCacheManager(ehCacheManagerFactoryBean().getObject());

return cacheManager;

}

}

applicationContext-DaoConfig.xml

<import resource="classpath:config/context/applicationContext-DataSourceConfig.xml" />

<bean id="sessionFactory"

class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="packagesToScan">

<list>

<value>web.function.\*\*.model.oracle</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">

${hibernate.dialect}

</prop>

<prop key="hibernate.show\_sql">${hibernate.show\_sql}</prop>

<prop key="hibernate.current\_session\_context\_class">org.springframework.orm.hibernate4.SpringSessionContext</prop>

</props>

</property>

</bean>

<!-- hibernateDAO -->

<bean id="hibernateDAO" class="web.dao.hibernate.impl.CP\_Hibernate4DAOImpl">

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>

<bean id="transactionManager"

class="org.springframework.orm.hibernate4.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!-- 启用事务注解功能 -->

<tx:annotation-driven transaction-manager="transactionManager" />

DaoConfig.java

@Configuration

//启用注解事务管理，使用CGLib代理

@EnableTransactionManagement(proxyTargetClass = true)

@Import({DataSourceConfig.class})

public class DaoConfig {

private static final Logger logger = Logger.getLogger(DaoConfig.class);

@Value("${hibernate.dialect}")

String hibernate\_dialect;

@Value("${hibernate.show\_sql}")

String hibernate\_show\_sql;

/\*\*

\* 描述 : <负责解析资源文件>. <br>

\*<p>

<这个类必须有，而且必须声明为static，否则不能正常解析>

</p>

\* @return

\*/

@Bean

public static PropertySourcesPlaceholderConfigurer placehodlerConfigurer() {

logger.info("PropertySourcesPlaceholderConfigurer");

return new PropertySourcesPlaceholderConfigurer();

}

@Resource(name="dataSource")

public DataSource dataSource;

@Bean(name = "sessionFactory")

public LocalSessionFactoryBean localSessionFactoryBean() {

logger.info("sessionFactory");

LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();

sessionFactory.setDataSource(dataSource);

String[] packagesToScan = new String[] { "web.function.\*\*.model.oracle" };

sessionFactory.setPackagesToScan(packagesToScan);

Properties hibernateProperties = new Properties();

hibernateProperties.setProperty("hibernate.dialect", hibernate\_dialect);

hibernateProperties.setProperty("hibernate.show\_sql",

hibernate\_show\_sql);

hibernateProperties.setProperty(

"hibernate.current\_session\_context\_class",

"org.springframework.orm.hibernate4.SpringSessionContext");

sessionFactory.setHibernateProperties(hibernateProperties);

return sessionFactory;

}

@Bean(name = "hibernateDAO")

public CP\_Hibernate4DAOImpl hibernate4Dao() {

logger.info("hibernateDAO");

CP\_Hibernate4DAOImpl dao = new CP\_Hibernate4DAOImpl();

dao.setSessionFactory(localSessionFactoryBean().getObject());

return dao;

}

@Bean(name = "transactionManager")

public HibernateTransactionManager hibernateTransactionManager() {

logger.info("transactionManager");

HibernateTransactionManager hibernateTransactionManager = new HibernateTransactionManager();

hibernateTransactionManager.setSessionFactory(localSessionFactoryBean()

.getObject());

return hibernateTransactionManager;

}

}

applicationContext-DataSourceConfig.xml

<context:property-placeholder

location="classpath:/config/properties/db.properties" />

<!-- 数据源 -->

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="${jdbc.driver}" />

<property name="url" value="${jdbc.url}" />

<property name="username" value="${jdbc.username}" />

<property name="password" value="${jdbc.password}" />

</bean>

DataSourceConfig.java

@Configuration

//加载资源文件

@PropertySource({"classpath:/config/properties/db.properties"})

public class DataSourceConfig {

private static final Logger logger = Logger.getLogger(DataSourceConfig.class);

/\*

\* 绑定资源属性

\*/

@Value("${jdbc.driver}")

String driverClass;

@Value("${jdbc.url}")

String url;

@Value("${jdbc.username}")

String userName;

@Value("${jdbc.password}")

String passWord;

@Bean(name = "dataSource")

public DataSource dataSource() {

logger.info("DataSource");

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName(driverClass);

dataSource.setUrl(url);

dataSource.setUsername(userName);

dataSource.setPassword(passWord);

return dataSource;

}

}

DataSource如果使用JNDI的方式如下：

xml配置：

<jee:jndi-lookup id="dataSource" jndi-name="jdbc/demoDB" />

Java类配置：

@Bean

public JndiObjectFactoryBean jndiObjectFactoryBean(){

JndiObjectFactoryBean factory = new JndiObjectFactoryBean();

// factory.setJndiName("java:comp/env/jdbc/demoDB"); //两种方式均可，spring会自动补齐

factory.setJndiName("jdbc/demoDB");

return factory;

}

@Bean(name = "dataSource")

public DataSource dataSource() throws Exception{

logger.info("DataSourceJNDI");

return (DataSource)jndiObjectFactoryBean().getObject();

}

# 四、SpringSecurity相关配置

SpringSecurity的配置相对来说有些复杂，如果是完整的bean配置，则需要配置大量的bean，所以xml配置时使用了命名空间来简化配置，同样，spring为我们提供了一个抽象类WebSecurityConfigurerAdapter和一个注解@EnableWebMvcSecurity，达到同样减少bean配置的目的，如下：

applicationContext-SpringSecurityConfig.xml

<http security="none" pattern="/static/\*\*" />

<http security="none" pattern="/\*\*/\*.jsp" />

<http auto-config='true' access-decision-manager-ref="accessDecisionManager" access-denied-page="/login"

use-expressions="true">

<logout logout-url="/logout" invalidate-session="true"

logout-success-url="/login" />

<form-login login-page="/login" authentication-failure-url="/login?error=1"

login-processing-url="/j\_spring\_security\_check" password-parameter="j\_password"

username-parameter="j\_username" />

<intercept-url pattern="/\*\*/\*.do\*" access="hasRole('ROLE\_USER')" />

<intercept-url pattern="/\*\*/\*.htm" access="hasRole('ROLE\_ADMIN')" />

<session-management session-fixation-protection="changeSessionId">

<concurrency-control max-sessions="1"

expired-url="/access/sameLogin.do" />

</session-management>

<remember-me key="webmvc#FD637E6D9C0F1A5A67082AF56CE32485"

remember-me-parameter="remember-me" />

</http>

<!-- 启用表达式 为了后面的投票器做准备 -->

<beans:bean

class="org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler"

id="expressionHandler" />

<beans:bean

class="org.springframework.security.web.access.expression.WebExpressionVoter"

id="expressionVoter">

<beans:property name="expressionHandler" ref="expressionHandler" />

</beans:bean>

<!-- Automatically receives AuthenticationEvent messages -->

<beans:bean id="loggerListener"

class="org.springframework.security.authentication.event.LoggerListener" />

<beans:bean id="authorizationListener"

class="org.springframework.security.access.event.LoggerListener" />

<!-- 认证管理器，使用自定义的UserDetailsService，并对密码采用md5加密 -->

<authentication-manager>

<authentication-provider user-service-ref="userService">

<password-encoder hash="md5" />

</authentication-provider>

</authentication-manager>

<beans:bean id="userService" class="web.security.CP\_UserDetailsService" />

<beans:bean id="accessDecisionManager"

class="org.springframework.security.access.vote.AffirmativeBased">

<beans:property name="decisionVoters">

<beans:list>

<beans:bean class="org.springframework.security.access.vote.RoleVoter" />

<beans:bean

class="org.springframework.security.access.vote.AuthenticatedVoter" />

<beans:ref bean="expressionVoter" />

</beans:list>

</beans:property>

</beans:bean>

SpringSecurityConfig.java

@Configuration

@EnableWebMvcSecurity

public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

private static final Logger logger = Logger

.getLogger(SpringSecurityConfig.class);

@Override

public void configure(WebSecurity web) throws Exception {

// 设置不拦截规则

web.ignoring().antMatchers("/static/\*\*", "/\*\*/\*.jsp");

}

@Override

protected void configure(HttpSecurity http) throws Exception {

// 设置拦截规则

// 自定义accessDecisionManager访问控制器,并开启表达式语言

http.authorizeRequests().accessDecisionManager(accessDecisionManager())

.expressionHandler(webSecurityExpressionHandler())

.antMatchers("/\*\*/\*.do\*").hasRole("USER")

.antMatchers("/\*\*/\*.htm").hasRole("ADMIN").and()

.exceptionHandling().accessDeniedPage("/login");

// 开启默认登录页面

// http.formLogin();

// 自定义登录页面

http.csrf().disable().formLogin().loginPage("/login")

.failureUrl("/login?error=1")

.loginProcessingUrl("/j\_spring\_security\_check")

.usernameParameter("j\_username")

.passwordParameter("j\_password").permitAll();

// 自定义注销

http.logout().logoutUrl("/logout").logoutSuccessUrl("/login")

.invalidateHttpSession(true);

// session管理

http.sessionManagement().sessionFixation().changeSessionId()

.maximumSessions(1).expiredUrl("/");

// RemeberMe

http.rememberMe().key("webmvc#FD637E6D9C0F1A5A67082AF56CE32485");

}

@Override

protected void configure(AuthenticationManagerBuilder auth)

throws Exception {

// 自定义UserDetailsService

auth.userDetailsService(userDetailsService()).passwordEncoder(

new Md5PasswordEncoder());

}

@Bean

public CP\_UserDetailsService userDetailsService() {

logger.info("CP\_UserDetailsService");

CP\_UserDetailsService userDetailsService = new CP\_UserDetailsService();

return userDetailsService;

}

@Bean

public LoggerListener loggerListener() {

logger.info("org.springframework.security.authentication.event.LoggerListener");

LoggerListener loggerListener = new LoggerListener();

return loggerListener;

}

@Bean

public org.springframework.security.access.event.LoggerListener eventLoggerListener() {

logger.info("org.springframework.security.access.event.LoggerListener");

org.springframework.security.access.event.LoggerListener eventLoggerListener = new org.springframework.security.access.event.LoggerListener();

return eventLoggerListener;

}

/\*

\*

\* 这里可以增加自定义的投票器

\*/

@SuppressWarnings("rawtypes")

@Bean(name = "accessDecisionManager")

public AccessDecisionManager accessDecisionManager() {

logger.info("AccessDecisionManager");

List<AccessDecisionVoter> decisionVoters = new ArrayList<AccessDecisionVoter>();

decisionVoters.add(new RoleVoter());

decisionVoters.add(new AuthenticatedVoter());

decisionVoters.add(webExpressionVoter());// 启用表达式投票器

AffirmativeBased accessDecisionManager = new AffirmativeBased(

decisionVoters);

return accessDecisionManager;

}

/\*

\* 表达式控制器

\*/

@Bean(name = "expressionHandler")

public DefaultWebSecurityExpressionHandler webSecurityExpressionHandler() {

logger.info("DefaultWebSecurityExpressionHandler");

DefaultWebSecurityExpressionHandler webSecurityExpressionHandler = new DefaultWebSecurityExpressionHandler();

return webSecurityExpressionHandler;

}

/\*

\* 表达式投票器

\*/

@Bean(name = "expressionVoter")

public WebExpressionVoter webExpressionVoter() {

logger.info("WebExpressionVoter");

WebExpressionVoter webExpressionVoter = new WebExpressionVoter();

webExpressionVoter.setExpressionHandler(webSecurityExpressionHandler());

return webExpressionVoter;

}

}

# 五、Web上下文配置

与SpringSecurity的配置类似，spring同样为我们提供了一个实现类WebMvcConfigurationSupport和一个注解@EnableWebMvc以帮助我们减少bean的声明。

applicationContext-MvcConfig.xml

<!-- 启用注解，并定义组件查找规则 ，mvc层只负责扫描@Controller -->

<context:component-scan base-package="web.function"

use-default-filters="false">

<context:include-filter type="annotation"

expression="org.springframework.stereotype.Controller" />

</context:component-scan>

<!-- 视图处理器 -->

<bean id="viewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/jsp/function/" />

<property name="suffix" value=".jsp" />

</bean>

<!-- 定义国际化资源文件查找规则 ，各种messages.properties -->

<bean id="messageSource"

class="org.springframework.context.support.ResourceBundleMessageSource"

p:basename="config.messages.messages">

</bean>

<!-- servlet适配器，这里必须明确声明，因为spring默认没有初始化该适配器 -->

<bean id="servletHandlerAdapter"

class="org.springframework.web.servlet.handler.SimpleServletHandlerAdapter" />

<!-- 定义文件上传处理器 -->

<bean id="multipartResolver"

class="org.springframework.web.multipart.commons.CommonsMultipartResolver"

p:defaultEncoding="UTF-8" />

<!-- 异常处理器 -->

<bean id="exceptionResolver" class="web.core.CP\_SimpleMappingExceptionResolver">

<property name="defaultErrorView" value="common\_error" />

<property name="exceptionAttribute" value="exception" />

<property name="exceptionMappings">

<props>

<prop key="java.lang.RuntimeException">common\_error</prop>

</props>

</property>

</bean>

<!-- 定义公共参数初始化拦截器 -->

<bean id="initInterceptor" class="web.core.CP\_InitializingInterceptor" />

<!-- 本地化资源处理器 -->

<bean id="localeResolver"

class="org.springframework.web.servlet.i18n.CookieLocaleResolver" />

<!-- 定义本地化变更拦截器 -->

<bean id="localeChangeInterceptor"

class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor" />

<!-- 请求拦截器，每一个用户请求都会被拦截 -->

<mvc:interceptors>

<ref bean="localeChangeInterceptor" />

<ref bean="initInterceptor" />

</mvc:interceptors>

<!-- 定义注解驱动Controller方法处理适配器 ,注：该适配器必须声明在<mvc:annotation-driven />之前，否则不能正常处理参数类型的转换 -->

<bean

class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">

<property name="webBindingInitializer">

<bean class="web.core.CP\_PropertyEditorRegistrar">

<property name="format" value="yyyy-MM-dd"></property>

</bean>

</property>

<property name="messageConverters">

<list>

<bean

class="org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter" />

<bean

class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter" />

</list>

</property>

</bean>

<!-- 会自动注册RequestMappingHandlerMapping与RequestMappingHandlerAdapter

两个bean,是spring MVC为@Controllers分发请求所必须的。 并提供了：数据绑定支持，@NumberFormatannotation支持，@DateTimeFormat支持，@Valid支持，读写XML的支持（JAXB），读写JSON的支持（Jackson） -->

<mvc:annotation-driven />

<!-- 资源访问处理器 -->

<mvc:resources mapping="/static/\*\*" location="/WEB-INF/static/" />

MvcConfig.java

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = "web.function", useDefaultFilters = false, includeFilters = {

@ComponentScan.Filter(type = FilterType.ANNOTATION, value = {Controller.class})

})

public class MvcConfig extends WebMvcConfigurationSupport {

private static final Logger logger = Logger

.getLogger(MvcConfig.class);

/\*\*

\* 描述 : <注册试图处理器>. <br>

\*<p>

<使用方法说明>

</p>

\* @return

\*/

@Bean

public ViewResolver viewResolver() {

logger.info("ViewResolver");

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setPrefix("/WEB-INF/views/jsp/function/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

/\*\*

\* 描述 : <注册消息资源处理器>. <br>

\*<p>

<使用方法说明>

</p>

\* @return

\*/

@Bean

public MessageSource messageSource() {

logger.info("MessageSource");

ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();

messageSource.setBasename("config.messages.messages");

return messageSource;

}

/\*\*

\* 描述 : <注册servlet适配器>. <br>

\*<p>

<只需要在自定义的servlet上用@Controller("映射路径")标注即可>

</p>

\* @return

\*/

@Bean

public HandlerAdapter servletHandlerAdapter(){

logger.info("HandlerAdapter");

return new SimpleServletHandlerAdapter();

}

/\*\*

\* 描述 : <本地化拦截器>. <br>

\*<p>

<使用方法说明>

</p>

\* @return

\*/

@Bean

public LocaleChangeInterceptor localeChangeInterceptor(){

logger.info("LocaleChangeInterceptor");

return new LocaleChangeInterceptor();

}

/\*\*

\* 描述 : <基于cookie的本地化资源处理器>. <br>

\*<p>

<使用方法说明>

</p>

\* @return

\*/

@Bean(name="localeResolver")

public CookieLocaleResolver cookieLocaleResolver(){

logger.info("CookieLocaleResolver");

return new CookieLocaleResolver();

}

/\*\*

\* 描述 : <注册自定义拦截器>. <br>

\*<p>

<使用方法说明>

</p>

\* @return

\*/

@Bean

public CP\_InitializingInterceptor initializingInterceptor(){

logger.info("CP\_InitializingInterceptor");

return new CP\_InitializingInterceptor();

}

/\*\*

\* 描述 : <RequestMappingHandlerMapping需要显示声明，否则不能注册自定义的拦截器>. <br>

\*<p>

<这个比较奇怪,理论上应该是不需要的>

</p>

\* @return

\*/

@Bean

public RequestMappingHandlerMapping requestMappingHandlerMapping() {

logger.info("RequestMappingHandlerMapping");

return super.requestMappingHandlerMapping();

}

/\*\*

\* 描述 : <添加拦截器>. <br>

\*<p>

<使用方法说明>

</p>

\* @param registry

\*/

@Override

protected void addInterceptors(InterceptorRegistry registry) {

// TODO Auto-generated method stub

logger.info("addInterceptors start");

registry.addInterceptor(localeChangeInterceptor());

registry.addInterceptor(initializingInterceptor());

logger.info("addInterceptors end");

}

/\*\*

\* 描述 : <HandlerMapping需要显示声明，否则不能注册资源访问处理器>. <br>

\*<p>

<这个比较奇怪,理论上应该是不需要的>

</p>

\* @return

\*/

@Bean

public HandlerMapping resourceHandlerMapping() {

logger.info("HandlerMapping");

return super.resourceHandlerMapping();

}

/\*\*

\* 描述 : <资源访问处理器>. <br>

\*<p>

<可以在jsp中使用/static/\*\*的方式访问/WEB-INF/static/下的内容>

</p>

\* @param registry

\*/

@Override

protected void addResourceHandlers(ResourceHandlerRegistry registry) {

logger.info("addResourceHandlers");

registry.addResourceHandler("/static/\*\*").addResourceLocations("/WEB-INF/static/");

}

/\*\*

\* 描述 : <文件上传处理器>. <br>

\*<p>

<使用方法说明>

</p>

\* @return

\*/

@Bean(name="multipartResolver")

public CommonsMultipartResolver commonsMultipartResolver(){

logger.info("CommonsMultipartResolver");

return new CommonsMultipartResolver();

}

/\*\*

\* 描述 : <异常处理器>. <br>

\*<p>

<系统运行时遇到指定的异常将会跳转到指定的页面>

</p>

\* @return

\*/

@Bean(name="exceptionResolver")

public CP\_SimpleMappingExceptionResolver simpleMappingExceptionResolver(){

logger.info("CP\_SimpleMappingExceptionResolver");

CP\_SimpleMappingExceptionResolver simpleMappingExceptionResolver= new CP\_SimpleMappingExceptionResolver();

simpleMappingExceptionResolver.setDefaultErrorView("common\_error");

simpleMappingExceptionResolver.setExceptionAttribute("exception");

Properties properties = new Properties();

properties.setProperty("java.lang.RuntimeException", "common\_error");

simpleMappingExceptionResolver.setExceptionMappings(properties);

return simpleMappingExceptionResolver;

}

/\*\*

\* 描述 : <RequestMappingHandlerAdapter需要显示声明，否则不能注册通用属性编辑器>. <br>

\*<p>

<这个比较奇怪,理论上应该是不需要的>

</p>

\* @return

\*/

@Bean

public RequestMappingHandlerAdapter requestMappingHandlerAdapter() {

logger.info("RequestMappingHandlerAdapter");

return super.requestMappingHandlerAdapter();

}

/\*\*

\* 描述 : <注册通用属性编辑器>. <br>

\*<p>

<这里只增加了字符串转日期和字符串两边去空格的处理>

</p>

\* @return

\*/

@Override

protected ConfigurableWebBindingInitializer getConfigurableWebBindingInitializer() {

logger.info("ConfigurableWebBindingInitializer");

ConfigurableWebBindingInitializer initializer = super.getConfigurableWebBindingInitializer();

CP\_PropertyEditorRegistrar register = new CP\_PropertyEditorRegistrar();

register.setFormat("yyyy-MM-dd");

initializer.setPropertyEditorRegistrar(register);

return initializer;

}

}