1. Shiro同Spring集成

本章主要介绍Shiro同Spring在Web应用中的集成。我们需要把握Shiro同Spring在Web应用中集成的步骤，进一步理解Shiro的工作原理和对Shiro安全框架的实现。

目录

[11 Shiro同Spring集成 1](#_Toc494386229)

[11.1 Web应用中同Spring集成 2](#_Toc494386230)

[11.1.1 集成准备 2](#_Toc494386231)

[11.1.2 编写Shiro注解初始化代码 4](#_Toc494386232)

[11.1.3 添加Spring配置文件 8](#_Toc494386233)

[11.1.4 配置web.xml 9](#_Toc494386234)

[11.1.5 测试 10](#_Toc494386235)

[11.2 J2SE独立应用中同Spring集成 10](#_Toc494386236)

[11.3 总结 10](#_Toc494386237)

通过对前面几章的研究，我们理解了Shiro的工作原理和使用方式，同时也初步了解了Shiro在Web应用中的运行使用方式和配置过程，接下来我们介绍Shiro同目前常用的框架Spring的集成。

Spring是容器框架，它能管理好应用中的所有业务逻辑对象，并且通过AOP的方式将这些对象整合在一起，它是我们目前开发Web应用的几个流行框架(SSH、SSM)的核心，能够实现同Spring的集成，就基本上能顺利的实现同这些流行框架的集成。

Shiro同Spring集成有两种情况：一种情况是在J2SE独立应用中同Spring集成；一种情况是在Web应用中同Spring集成。鉴于目前大部分都是Web应用，本章主要讲解Web应用中同Spring的集成，J2SE独立应用的集成不做详解。

* 1. Web应用中同Spring集成
     1. 集成准备

本节主要体现Shiro和Web环境下的Spring的集成，用到的案例是第九章，ShiroTP09样例代码中案例。所以准备工作内容主要分三部分：基于Maven创建Web工程；将ShiroTP09里面的代码引用或复制到新Web工程；在Web工程中添加Spring相关的依赖。创建Maven的Web工程（ShiroTP11），我们在这里就不重复说明了，详细内容请查看随书代码[code/ShiroTP11](../code/ShiroTP11)，这里主要介绍引用和赋值ShiroTP09的代码和添加Spring的相关依赖。

* 移植以前的案例代码

在ShiroTP09中有两种代码，一种是引用的ShiroTP07中自定义的Shiro组件类，包括BitPermission.java， BitAndWildcardPermissionResolver.java， ShiroRolePermissionResolver.java， ShiroRealm.java四个类；一种是ShiroTP09中自己添加的AdminServlet.java， CreateUserServlet.java， UpdateUserServlet.java， DeleteUserServlet.java， QueryUserServlet.java， index.jsp， welcome.jsp， unauthorized.jsp。

前面自定义的四个Shiro组件类，我们通过在ShiroTP11/pom.xml中添加依赖，引用到ShiroTP11工程，样例代码块如下，详细内容请查看随书代码[code/ShiroTP11/pom.xml](../code/ShiroTP11/pom.xml)。

<dependency>

<groupId>ShiroBook</groupId>

<artifactId>ShiroTP07</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>

后面的Servlet代码和jsp，分别原样从ShiroTP09中复制到ShiroTP11中来，详细内容请分别查看随书样例代码目录[code/ShiroTP11/src/main/java/cn/com/shiro/book/  
tp09/servlet](../code/ShiroTP11/src/main/java/cn/com/shiro/book/tp09/servlet) 和 [code/ShiroTP11/src/main/webapp](../code/ShiroTP11/src/main/webapp)。

* 添加Spring和Shiro的依赖

在ShiroTP11/pom.xml中添加Spring相关依赖，样例代码块如下，详细内容请查看随书代码[code/ShiroTP11/pom.xml](../code/ShiroTP11/pom.xml)。

<!-- spring核心包 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${project.build.spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${project.build.spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>${project.build.spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${project.build.spring.version}</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-aspects -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>${project.build.spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${project.build.spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>${project.build.spring.version}</version>

</dependency>

在ShiroTP11/pom.xml中添加Shiro相关依赖，样例代码块如下，详细内容请查看随书代码[code/ShiroTP11/pom.xml](../code/ShiroTP11/pom.xml)。

<parent>

<groupId>ShiroBook</groupId>

<artifactId>ShiroPOM</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

… …

<dependencies>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-cas</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-ehcache</artifactId>

<version>${shiro.version}</version>

</dependency>

<!-- ehcache -->

<dependency>

<groupId>net.sf.ehcache</groupId>

<artifactId>ehcache-core</artifactId>

<version>2.5.3</version>

</dependency>

</ dependencies>

* + 1. 编写Shiro注解初始化代码

创建ShiroWebConfiguration类，用编码方式完成Spring中Shiro组件对象的创建和装配，样例代码块如下，详细内容请参考随书代码[code/ShiroTP11/src/main/java/cn/com/shiro/  
book/tp11/config/ShiroWebConfiguration.java](../code/ShiroTP11/src/main/java/cn/com/shiro/book/tp11/config/ShiroWebConfiguration.java)。

// 支持注解

**@Configuration**

// 注解扫描包

**@ComponentScan("cn.com.shiro.book")**

public class ShiroWebConfiguration {

**@Bean(name = "shiroFilter") ①**

public ShiroFilterFactoryBean createShiroFilterFactoryBean() {

System.out

.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*create shiroFilter \*\*\*\*\*\*\*\*\*\*\*\*\*");

ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();

// 设置安全管理器

shiroFilter.setSecurityManager(this.createSecurityManager());

String chainDefinitions = "/index.jsp=anon\r\n" + "#/welcome.jsp=authc"

+ "#/unauthorized.jsp=authc\r\n" + "/login=authc\r\n"

+ "/admin/adminServlet=authc,roles[admin]\r\n"

+ "/user/createUserServlet=authc, roles[role1]\r\n"

+ "/user/deleteUserServlet=authc, roles[role1,role2]\r\n"

+ "/user/updateUserServlet=authc, perms[1:4]\r\n"

+ "/user/queryUserServlet=authcBasic, perms[1:8]\r\n"

+ "/logout=logout\r\n" + "/\*=authc";

// 设置url同filter拦截器的关系

shiroFilter.setFilterChainDefinitions(chainDefinitions);

**// 初始化filter拦截器**

**this.setFilterAttributes(shiroFilter);**

return shiroFilter;

}

// 初始化authc、logout、roles、perms和authcBasic拦截器

**private void setFilterAttributes(ShiroFilterFactoryBean shiroFilter) { ②**

Map<String, Filter> filters = shiroFilter.getFilters();

FormAuthenticationFilter authc = (FormAuthenticationFilter) filters

.get("authc");

if (authc == null) {

authc = new FormAuthenticationFilter();

filters.put("authc", authc);

}

authc.setSuccessUrl("/welcome.jsp");

authc.setLoginUrl("/login");

authc.setUsernameParam("userName");

authc.setPasswordParam("password");

LogoutFilter logout = (LogoutFilter) filters.get("logout");

if (logout == null) {

logout = new LogoutFilter();

filters.put("logout", logout);

}

logout.setRedirectUrl("/login");

RolesAuthorizationFilter roles = (RolesAuthorizationFilter) filters

.get("roles");

if (roles == null) {

roles = new RolesAuthorizationFilter();

filters.put("roles", roles);

}

roles.setUnauthorizedUrl("/unauthorized.jsp");

PermissionsAuthorizationFilter perms = (PermissionsAuthorizationFilter) filters

.get("perms");

if (perms == null) {

perms = new PermissionsAuthorizationFilter();

filters.put("perms", perms);

}

perms.setUnauthorizedUrl("/unauthorized.jsp");

BasicHttpAuthenticationFilter authcBasic = (BasicHttpAuthenticationFilter) filters

.get("authcBasic");

if (authcBasic == null) {

authcBasic = new BasicHttpAuthenticationFilter();

filters.put("authcBasic", authcBasic);

}

authcBasic.setApplicationName("ShiroBook 登录");

}

// shiro生命周期处理器

**@Bean(name = "lifecycleBeanPostProcessor") ③**

public LifecycleBeanPostProcessor createLifecycleBeanPostProcessor() {

return new LifecycleBeanPostProcessor();

}

// 调用SecurityUtils注册全局SecurityManager对象

**@Bean**

**public MethodInvokingFactoryBean createMethodInvokingFactoryBean() { ④**

MethodInvokingFactoryBean factoryBean = new MethodInvokingFactoryBean();

factoryBean

.setStaticMethod("org.apache.shiro.SecurityUtils.setSecurityManager");

factoryBean.setArguments(new Object[] { this.createSecurityManager() });

return factoryBean;

}

**@Bean(name = "securityManager") ⑤**

public org.apache.shiro.mgt.SecurityManager createSecurityManager() {

DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();

securityManager.setAuthorizer(this.createModularRealmAuthorizer());

securityManager.setRealms(Arrays.asList(this.createRealm()));

return securityManager;

}

@Bean(name = "shiroRealm")

public Realm createRealm() {

ShiroRealm realm = new ShiroRealm();

realm.setPasswordService(this.createPasswordService());

realm.setCredentialsMatcher(this.createPasswordMatcher());

return realm;

}

@Bean(name = "authorizer")

public ModularRealmAuthorizer createModularRealmAuthorizer() {

ModularRealmAuthorizer authorizer = new ModularRealmAuthorizer();

authorizer.setPermissionResolver(this.createPermissionResolver());

authorizer.setRolePermissionResolver(this

.createRolePermissionResolver());

return authorizer;

}

@Bean(name = "rolePermissionResolver")

public RolePermissionResolver createRolePermissionResolver() {

RolePermissionResolver rolePermResolver = new ShiroRolePermissionResovler();

return rolePermResolver;

}

@Bean(name = "permResolver")

public PermissionResolver createPermissionResolver() {

PermissionResolver permResolver = new BitAndWildcardPermissionResolver();

return permResolver;

}

@Bean(name = "passwordMatcher")

public PasswordMatcher createPasswordMatcher() {

PasswordMatcher passwordMatcher = new PasswordMatcher();

passwordMatcher.setPasswordService(this.createPasswordService());

;

return passwordMatcher;

}

@Bean(name = "passwordService")

public PasswordService createPasswordService() {

DefaultPasswordService passwordService = new DefaultPasswordService();

passwordService.setHashFormat(this.createHashFormat());

passwordService.setHashFormatFactory(this.createHashFormatFactory());

passwordService.setHashService(this.createHashService());

return passwordService;

}

@Bean(name = "hashFormatFactory")

public HashFormatFactory createHashFormatFactory() {

return new DefaultHashFormatFactory();

}

@Bean(name = "hashFormat")

public HashFormat createHashFormat() {

return new Shiro1CryptFormat();

}

@Bean(name = "hashService")

public HashService createHashService() {

HashService hashService = new DefaultHashService();

return hashService;

}

}

如上代码的作用，同第九章中[code/ShiroTP09/src/main/resources/shiro\_shiroRealm.ini](../code/ShiroTP09/src/main/resources/shiro_shiroRealm.ini)的作用一样，都是初始化Shiro安全框架的对象，只是以前是用配置文件描述，由Shiro安全框架自己读配置文件初始化，如上内容是用代码创建并且装配好Shiro安全框架的对象，最后交给Spring管理。同shiro\_shiroRealm.ini相同的内容，这里就不重复了，接下来着重介绍不同的部分。

1. ShiroWebConfiguration类上面添加了@Configuration和@ComponentScan两个注解。@Configuration是告诉Spring容器这是个初始化配置文件，需要Spring容器在初始化的时候，依据该类中的代码，创建Bean并且管理到容器中；@ComponentScan告诉Spring容器在初始化的时候，自动扫描指定包(包含所有子包)下的所有类，识别里面的Spring的相关注解，并做好有注解标记的Bean的初始化。
2. **①**标记的注解对应的方法，创建一个ShiroFilterFactoryBean对象，该对象用来初始化拦截url，进入Shiro内部进行安全控制的ShiroFilter对象。这个对象在Spring容器中的名称推荐使用shiroFilter。可以自定义另外一个名称，但必须同在11.1.3中介绍的web.xml中的过滤器名称一样。
3. **②**标记的方法初始化Shiro安全框架中用到的拦截器对象，在createShiroFilterFactoryBean中被调用。
4. **③**标记的方法创建Shiro生命周期处理器对象
5. **④**标记的方法创建MethodInvokingFactoryBean对象，使用SecurityUtils的setSecurityManager方法，注册全局的SecurityManager对象。
6. **⑤**标记的方法创建一个SecurityManager对象，因为是Web环境下，需要创建WebSecurityManager类型的对象。
   * 1. 添加Spring配置文件

Spring容器启动的时候，需要根据配置文件进行初始化，样例代码如下，详细内容请查看随书代码[code/ShiroTP11/src/main/applicationContext.xml](../code/ShiroTP11/src/main/resources/applicationContext.xml)。

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

<beans xmlns="http://www.springframework.org/schema/beans"

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

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.0.xsd">

<context:component-scan

base-package="cn.com.shiro.book"></context:component-scan>

</beans>

<context:component-scan base-package=”cn.com.shiro.book”></context:component-scan>指定Spring容器初始化的时候，自动要扫描的基本包。

* + 1. 配置web.xml

在web.xml中，我们主要做两个方面的配置：一是配置ContextLoaderListener初始化Spring容器；一是配置DelegatingFilterProxy从Spring容器中获取Shiro对url进行安全控制的入口ShiroFilter对象。详细内容请参考随书代码[code/ShiroTP11/src/  
main/webapp/WEB-INF/web.xml](../code/ShiroTP11/src/main/webapp/WEB-INF/web.xml)。

* 配置ContextLoaderListener

样例代码块如下：

<context-param>

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

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

</context-param>

<listener>

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

</listener>

context-param配置ContextLoaderListener监听器类要加载的文件。param-name的值是固定的，param-value是spring要加载的文件。

* 配置DelegatingFilterProxy

样例代码块如下：

<filter>

<filter-name>**shiroFilter**</filter-name>

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

<init-param>

<param-name>targetFilterLifecycle</param-name>

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

</init-param>

</filter>

<filter-mapping>

<filter-name>shiroFilter</filter-name>

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

</filter-mapping>

通过DelegatingFilterProxy，Web应用就可以从Spring容器中，根据DelegatingFilterProxy的filter-name为名称查找ShiroFilter对象。filter-name的值，必须同11.1.2小节里面，使用createShiroFilterFactoryBean方法创建的bean的名称保持一致。

* + 1. 测试

同第九章的测试一样，发布Web应用到Web服务器后，在浏览器的地址栏中输入admin/adminServlet, welcome.jsp, user/createUserServlet等请求，就可以体验到Shiro安全框架在做的安全控制。

* 1. J2SE独立应用中同Spring集成

同11.1小节类似，创建一个同ShiroWebConfiguration.java类似的类ShiroJ2SEConfiguration.java，初始化Spring容器中要管理的Shiro安全框架组件，再创建一个一样的applicationContext.xml文件就行。

ShiroJ2SEConfiguration.java里面的内容，格式同ShiroWebConfiguration.java一样，创建的对象，对比第七章的代码[code/ShiroTP07/src/main/resources/shiro\_shiroRealm.ini](../code/ShiroTP07/src/main/resources/shiro_shiroRealm.ini)中描述的对象创建就行。

* 1. 总结

本章没有新的知识点，主要是实现Shiro同Web环境下的Spring集成，注意集成需要编写的代码、配置文件和步骤，理解各自的作用（最好对比Shiro安全框架运行流程）。