Shiro的组件都是JavaBean/POJO式的组件，所以非常容易使用Spring进行组件管理，可以非常方便的从ini配置迁移到Spring进行管理，且支持JavaSE应用及Web应用的集成。

在示例之前，需要导入shiro-spring及spring-context依赖，具体请参考pom.xml。

spring-beans.xml配置文件提供了基础组件如DataSource、DAO、Service组件的配置。

## JavaSE应用

spring-shiro.xml提供了普通JavaSE独立应用的Spring配置：

**Java代码  [收藏代码](javascript:void())**

1. <!-- 缓存管理器 使用Ehcache实现 -->
2. <bean id="cacheManager" **class**="org.apache.shiro.cache.ehcache.EhCacheManager">
3. <property name="cacheManagerConfigFile" value="classpath:ehcache.xml"/>
4. </bean>
6. <!-- 凭证匹配器 -->
7. <bean id="credentialsMatcher" **class**="
8. com.github.zhangkaitao.shiro.chapter12.credentials.RetryLimitHashedCredentialsMatcher">
9. <constructor-arg ref="cacheManager"/>
10. <property name="hashAlgorithmName" value="md5"/>
11. <property name="hashIterations" value="2"/>
12. <property name="storedCredentialsHexEncoded" value="true"/>
13. </bean>
15. <!-- Realm实现 -->
16. <bean id="userRealm" **class**="com.github.zhangkaitao.shiro.chapter12.realm.UserRealm">
17. <property name="userService" ref="userService"/>
18. <property name="credentialsMatcher" ref="credentialsMatcher"/>
19. <property name="cachingEnabled" value="true"/>
20. <property name="authenticationCachingEnabled" value="true"/>
21. <property name="authenticationCacheName" value="authenticationCache"/>
22. <property name="authorizationCachingEnabled" value="true"/>
23. <property name="authorizationCacheName" value="authorizationCache"/>
24. </bean>
25. <!-- 会话ID生成器 -->
26. <bean id="sessionIdGenerator"
27. **class**="org.apache.shiro.session.mgt.eis.JavaUuidSessionIdGenerator"/>
28. <!-- 会话DAO -->
29. <bean id="sessionDAO"
30. **class**="org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO">
31. <property name="activeSessionsCacheName" value="shiro-activeSessionCache"/>
32. <property name="sessionIdGenerator" ref="sessionIdGenerator"/>
33. </bean>
34. <!-- 会话验证调度器 -->
35. <bean id="sessionValidationScheduler"
36. **class**="org.apache.shiro.session.mgt.quartz.QuartzSessionValidationScheduler">
37. <property name="sessionValidationInterval" value="1800000"/>
38. <property name="sessionManager" ref="sessionManager"/>
39. </bean>
40. <!-- 会话管理器 -->
41. <bean id="sessionManager" **class**="org.apache.shiro.session.mgt.DefaultSessionManager">
42. <property name="globalSessionTimeout" value="1800000"/>
43. <property name="deleteInvalidSessions" value="true"/>
44. <property name="sessionValidationSchedulerEnabled" value="true"/>
45. <property name="sessionValidationScheduler" ref="sessionValidationScheduler"/>
46. <property name="sessionDAO" ref="sessionDAO"/>
47. </bean>
48. <!-- 安全管理器 -->
49. <bean id="securityManager" **class**="org.apache.shiro.mgt.DefaultSecurityManager">
50. <property name="realms">
51. <list><ref bean="userRealm"/></list>
52. </property>
53. <property name="sessionManager" ref="sessionManager"/>
54. <property name="cacheManager" ref="cacheManager"/>
55. </bean>
56. <!-- 相当于调用SecurityUtils.setSecurityManager(securityManager) -->
57. <bean **class**="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
58. <property name="staticMethod"
59. value="org.apache.shiro.SecurityUtils.setSecurityManager"/>
60. <property name="arguments" ref="securityManager"/>
61. </bean>
62. <!-- Shiro生命周期处理器-->
63. <bean id="lifecycleBeanPostProcessor"
64. **class**="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>

可以看出，只要把之前的ini配置翻译为此处的spring xml配置方式即可，无须多解释。LifecycleBeanPostProcessor用于在实现了Initializable接口的Shiro bean初始化时调用Initializable接口回调，在实现了Destroyable接口的Shiro bean销毁时调用 Destroyable接口回调。如UserRealm就实现了Initializable，而DefaultSecurityManager实现了Destroyable。具体可以查看它们的继承关系。

测试用例请参考com.github.zhangkaitao.shiro.chapter12.ShiroTest。

## Web应用

Web应用和普通JavaSE应用的某些配置是类似的，此处只提供一些不一样的配置，详细配置可以参考spring-shiro-web.xml。

**Java代码  [收藏代码](javascript:void())**

1. <!-- 会话Cookie模板 -->
2. <bean id="sessionIdCookie" **class**="org.apache.shiro.web.servlet.SimpleCookie">
3. <constructor-arg value="sid"/>
4. <property name="httpOnly" value="true"/>
5. <property name="maxAge" value="180000"/>
6. </bean>
7. <!-- 会话管理器 -->
8. <bean id="sessionManager"
9. **class**="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
10. <property name="globalSessionTimeout" value="1800000"/>
11. <property name="deleteInvalidSessions" value="true"/>
12. <property name="sessionValidationSchedulerEnabled" value="true"/>
13. <property name="sessionValidationScheduler" ref="sessionValidationScheduler"/>
14. <property name="sessionDAO" ref="sessionDAO"/>
15. <property name="sessionIdCookieEnabled" value="true"/>
16. <property name="sessionIdCookie" ref="sessionIdCookie"/>
17. </bean>
18. <!-- 安全管理器 -->
19. <bean id="securityManager" **class**="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
20. <property name="realm" ref="userRealm"/>
21. <property name="sessionManager" ref="sessionManager"/>
22. <property name="cacheManager" ref="cacheManager"/>
23. </bean>

1、sessionIdCookie是用于生产Session ID Cookie的模板；

2、会话管理器使用用于web环境的DefaultWebSessionManager；

3、安全管理器使用用于web环境的DefaultWebSecurityManager。

**Java代码  [收藏代码](javascript:void())**

1. <!-- 基于Form表单的身份验证过滤器 -->
2. <bean id="formAuthenticationFilter"
3. **class**="org.apache.shiro.web.filter.authc.FormAuthenticationFilter">
4. <property name="usernameParam" value="username"/>
5. <property name="passwordParam" value="password"/>
6. <property name="loginUrl" value="/login.jsp"/>
7. </bean>
8. <!-- Shiro的Web过滤器 -->
9. <bean id="shiroFilter" **class**="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
10. <property name="securityManager" ref="securityManager"/>
11. <property name="loginUrl" value="/login.jsp"/>
12. <property name="unauthorizedUrl" value="/unauthorized.jsp"/>
13. <property name="filters">
14. <util:map>
15. <entry key="authc" value-ref="formAuthenticationFilter"/>
16. </util:map>
17. </property>
18. <property name="filterChainDefinitions">
19. <value>
20. /index.jsp = anon
21. /unauthorized.jsp = anon
22. /login.jsp = authc
23. /logout = logout
24. /\*\* = user
25. </value>
26. </property>
27. </bean>

1、formAuthenticationFilter为基于Form表单的身份验证过滤器；此处可以再添加自己的Filter bean定义；

2、shiroFilter：此处使用ShiroFilterFactoryBean来创建ShiroFilter过滤器；filters属性用于定义自己的过滤器，即ini配置中的[filters]部分；filterChainDefinitions用于声明url和filter的关系，即ini配置中的[urls]部分。

接着需要在web.xml中进行如下配置：

**Java代码  [收藏代码](javascript:void())**

1. <context-param>
2. <param-name>contextConfigLocation</param-name>
3. <param-value>
4. classpath:spring-beans.xml,
5. classpath:spring-shiro-web.xml
6. </param-value>
7. </context-param>
8. <listener>
9. <listener-**class**>
10. org.springframework.web.context.ContextLoaderListener
11. </listener-**class**>
12. </listener>

通过ContextLoaderListener加载contextConfigLocation指定的Spring配置文件。

**Java代码  [收藏代码](javascript:void())**

1. <filter>
2. <filter-name>shiroFilter</filter-name>
3. <filter-**class**>org.springframework.web.filter.DelegatingFilterProxy</filter-**class**>
4. <init-param>
5. <param-name>targetFilterLifecycle</param-name>
6. <param-value>**true**</param-value>
7. </init-param>
8. </filter>
9. <filter-mapping>
10. <filter-name>shiroFilter</filter-name>
11. <url-pattern>/\*</url-pattern>
12. </filter-mapping>

DelegatingFilterProxy会自动到Spring容器中查找名字为shiroFilter的bean并把filter请求交给它处理。

其他配置请参考源代码。

## Shiro权限注解

Shiro提供了相应的注解用于权限控制，如果使用这些注解就需要使用AOP的功能来进行判断，如Spring AOP；Shiro提供了Spring AOP集成用于权限注解的解析和验证。

为了测试，此处使用了Spring MVC来测试Shiro注解，当然Shiro注解不仅仅可以在web环境使用，在独立的JavaSE中也是可以用的，此处只是以web为例了。

在spring-mvc.xml配置文件添加Shiro Spring AOP权限注解的支持：

**Java代码  [收藏代码](javascript:void())**

1. <aop:config proxy-target-**class**="true"></aop:config>
2. <bean **class**="
3. org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
4. <property name="securityManager" ref="securityManager"/>
5. </bean>

如上配置用于开启Shiro Spring AOP权限注解的支持；<aop:config proxy-target-class="true">表示代理类。

接着就可以在相应的控制器（AnnotationController）中使用如下方式进行注解：

**Java代码  [收藏代码](javascript:void())**

1. @RequiresRoles("admin")
2. @RequestMapping("/hello2")
3. **public** String hello2() {
4. **return** "success";
5. }

访问hello2方法的前提是当前用户有admin角色。

当验证失败，其会抛出UnauthorizedException异常，此时可以使用Spring的ExceptionHandler（DefaultExceptionHandler）来进行拦截处理：

**Java代码  [收藏代码](javascript:void())**

1. @ExceptionHandler({UnauthorizedException.**class**})
2. @ResponseStatus(HttpStatus.UNAUTHORIZED)
3. **public** ModelAndView processUnauthenticatedException(NativeWebRequest request, UnauthorizedException e) {
4. ModelAndView mv = **new** ModelAndView();
5. mv.addObject("exception", e);
6. mv.setViewName("unauthorized");
7. **return** mv;
8. }

如果集成Struts2，需要注意《Shiro+Struts2+Spring3 加上@RequiresPermissions 后@Autowired失效》问题：

<http://jinnianshilongnian.iteye.com/blog/1850425>

**权限注解**

**Java代码  [收藏代码](javascript:void())**

1. @RequiresAuthentication

表示当前Subject已经通过login进行了身份验证；即Subject. isAuthenticated()返回true。

**Java代码  [收藏代码](javascript:void())**

1. @RequiresUser

表示当前Subject已经身份验证或者通过记住我登录的。

**Java代码  [收藏代码](javascript:void())**

1. @RequiresGuest

表示当前Subject没有身份验证或通过记住我登录过，即是游客身份。

**Java代码  [收藏代码](javascript:void())**

1. @RequiresRoles(value={“admin”, “user”}, logical= Logical.AND)

表示当前Subject需要角色admin和user。

**Java代码  [收藏代码](javascript:void())**

1. @RequiresPermissions (value={“user:a”, “user:b”}, logical= Logical.OR)

表示当前Subject需要权限user:a或user:b。