## Shiro

### shiro基础介绍

#### 什么是shiro？

一个强大且易用的Java安全框架,能够用于身份验证、授权、加密和会话管理。Shiro拥有易于理解的API,您可以快速、轻松地获得任何应用程序——从最小的移动应用程序到最大的网络和企业应用程序。

简而言之，Apache Shiro 是一个强大灵活的开源安全框架，可以完全处理身份验证、授权、加密和会话管理。

#### shrio能做什么呢？

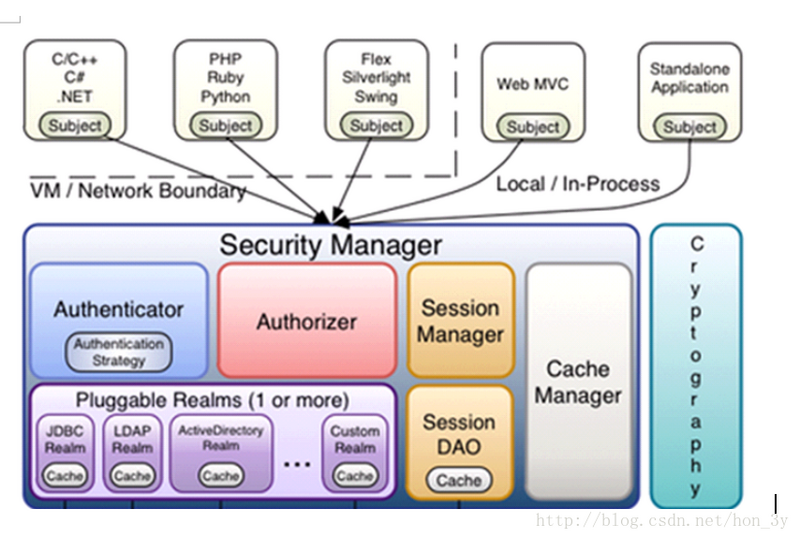
* 验证用户身份
* 用户访问权限控制，比如：1、判断用户是否分配了一定的安全角色。2、判断用户是否被授予完成某个操作的权限
* 在非 Web 或 EJB 容器的环境下可以任意使用Session API
* 可以响应认证、访问控制，或者 Session 生命周期中发生的事件
* 可将一个或以上用户安全数据源数据组合成一个复合的用户 “view”(视图)
* 支持单点登录(SSO)功能
* 支持提供“Remember Me”服务，获取用户关联信息而无需登录

#### 我们为什么要用shiro？

* **易于使用**——易用性是项目的最终目标。应用程序安全非常令人困惑和沮丧,被认为是“不可避免的灾难”。如果你让它简化到新手都可以使用它,它就将不再是一种痛苦了。
* **全面**——没有其他安全框架的宽度范围可以同Apache Shiro一样,它可以成为你的“一站式”为您的安全需求提供保障。
* **灵活**——Apache Shiro可以在任何应用程序环境中工作。虽然在网络工作、EJB和IoC环境中可能并不需要它。但Shiro的授权也没有任何规范,甚至没有许多依赖关系。
* **Web支持**——Apache Shiro拥有令人兴奋的web应用程序支持,允许您基于应用程序的url创建灵活的安全策略和网络协议(例如REST),同时还提供一组JSP库控制页面输出。
* **低耦合**——Shiro干净的API和设计模式使它容易与许多其他框架和应用程序集成。你会看到Shiro无缝地集成Spring这样的框架, 以及Grails, Wicket, Tapestry, Mule, Apache Camel, Vaadin…等。
* **被广泛支持**——Apache Shiro是Apache软件基金会的一部分。项目开发和用户组都有友好的网民愿意帮助。这样的商业公司如果需要Katasoft还提供专业的支持和服务。

### shiro的模块

#### 1、基本介绍



* **subject**：主体，可以是用户也可以是程序，主体要访问系统，系统需要对主体进行认证、授权。
* **securityManager**：安全管理器，主体进行认证和授权都 是通过securityManager进行。
* **authenticator**：认证器，主体进行认证最终通过authenticator进行的。
* **authorizer**：授权器，主体进行授权最终通过authorizer进行的。
* **sessionManager**：web应用中一般是用web容器对session进行管理，shiro也提供一套session管理的方式。
* **SessionDao**： 通过SessionDao管理session数据，针对个性化的session数据存储需要使用sessionDao。
* **cache Manager**：缓存管理器，主要对session和授权数据进行缓存，比如将授权数据通过cacheManager进行缓存管理，和ehcache整合对缓存数据进行管理。
* **realm**：域，领域，相当于数据源，通过realm存取认证、授权相关数据。
* **cryptography**：密码管理，提供了一套加密/解密的组件，方便开发。比如提供常用的散列、加/解密等功能。

#### 几个模块的例子

配置文件：重要 多敲！

1. @Configuration
2. **public** **class** ShiroConfig {
3. //1.配置ShiroFilterFactoryBean
4. @Bean
5. **public** ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager){
6. //1.1）定义一个ShiroFitlerFactoryBean对象
7. ShiroFilterFactoryBean filterFactoryBean = **new** ShiroFilterFactoryBean();
8. filterFactoryBean.setSecurityManager(securityManager);
9. //1.1.1)设置一系列的登录参数
10. filterFactoryBean.setLoginUrl("/login");            //设置登录地址
11. filterFactoryBean.setSuccessUrl("/user/listmenu");  //登录成功后跳转的页面
12. filterFactoryBean.setUnauthorizedUrl("/refuse.jsp");//授权失败后跳转的页面
14. //1.2) 设置过滤器链的定义(这里需要定义的过滤器是有序的，所以选择LinkedHashMap)
15. Map<String, String> filterChainMap = **new** LinkedHashMap<>();
16. //1.3)向过滤器链中添加对指定资源的过滤逻辑
17. filterChainMap.put("/plugins/\*\*","anon");
18. filterChainMap.put("/validatecode.jsp","anon");
19. filterChainMap.put("/refuse.jsp","anon");
20. filterChainMap.put("/logout","logout");
21. //1.4)配置通过“记住我”功能进行放行的资源
22. filterChainMap.put("/user/\*\*","user");
23. filterChainMap.put("/student/\*\*","user");
24. //拦截其它资源
25. filterChainMap.put("/\*\*","authc");
26. //1.4)将绑定filterChainMap与filterFactoryBean进行绑定
27. filterFactoryBean.setFilterChainDefinitionMap(filterChainMap);
28. //1.5）配置自定义过滤器
29. Map<String, Filter> filters = **new** LinkedHashMap<>();
30. filters.put("authc",**new** CustomFormFilter());
31. //1.6)将自定义的过滤器与filterFactoryBean
32. filterFactoryBean.setFilters(filters);
33. //返回
34. **return** filterFactoryBean;
35. }
36. //2.配置SecurityManager
37. @Bean
38. **public** SecurityManager securityManager(CustomRealm customRealm, CacheManager cacheManager, RememberMeManager meManager){
39. //2.1)定义SecurityManager对象
40. DefaultWebSecurityManager securityManager = **new** DefaultWebSecurityManager();
41. //2.2)配置自定义Realm
42. securityManager.setRealm(customRealm);
43. //2.3)配置缓存管理器
44. securityManager.setCacheManager(cacheManager);
45. //2.4)配置rememberMeManager
46. securityManager.setRememberMeManager(meManager);
47. //返回
48. **return** securityManager;
49. }
50. //3.配置自定义Realm对象
51. @Bean
52. **public** CustomRealm customRealm(CredentialsMatcher credentialsMatcher){
53. //3.1)构造自定义realm对象
54. CustomRealm customRealm = **new** CustomRealm();
55. //3.2)将凭证匹配器与自定义Realm进行绑定
56. customRealm.setCredentialsMatcher(credentialsMatcher);
57. //返回
58. **return** customRealm;
59. }
60. //4.配置凭证匹配器对象
61. @Bean
62. **public** CredentialsMatcher credentialsMatcher(){
63. //4.1)构造一个凭证匹配器对象
64. HashedCredentialsMatcher credentialsMatcher = **new** HashedCredentialsMatcher();
65. //4.2)设置参数
66. credentialsMatcher.setHashAlgorithmName("md5");
67. credentialsMatcher.setHashIterations(1);
68. //返回
69. **return** credentialsMatcher;
70. }
71. //5.配置缓存管理器
72. @Bean
73. **public** CacheManager cacheManager(){
74. //5.1)定义一个缓存管理器
75. EhCacheManager cacheManager = **new** EhCacheManager();
76. //5.2)设置缓存管理器的位置
77. cacheManager.setCacheManagerConfigFile("classpath:cache/shiro-cache.xml");
78. //返回
79. **return** cacheManager;
80. }
81. //6.配置rememberMeManager
82. @Bean
83. **public** RememberMeManager meManager(Cookie cookie){
84. CookieRememberMeManager rememberMeManager = **new** CookieRememberMeManager();
85. rememberMeManager.setCookie(cookie);
86. **return** rememberMeManager;
87. }
88. //7.定义Cookie对象
89. @Bean
90. **public** Cookie cookie(){
91. Cookie cookie = **new** SimpleCookie();
92. cookie.setMaxAge(3600); //设置到期时间
93. cookie.setName("rememberMe");    //设置此cookie名称与表单中复选框名字一致
94. **return** cookie;
95. }
96. /\*\*
97. \* Spring的一个bean , 由Advisor决定对哪些类的方法进行AOP代理
98. \* @return
99. \*/
100. @Bean
101. **public** DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator(){
102. DefaultAdvisorAutoProxyCreator proxyCreator = **new** DefaultAdvisorAutoProxyCreator();
103. proxyCreator.setProxyTargetClass(**true**);
104. **return** proxyCreator;
105. }
106. //配置AuthorizationAttributeSourceAdvisor
107. @Bean
108. **public** AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){
109. AuthorizationAttributeSourceAdvisor sourceAdvisor = **new** AuthorizationAttributeSourceAdvisor();
110. sourceAdvisor.setSecurityManager(securityManager);
111. **return** sourceAdvisor;
112. }
113. //配置springmvc能一般简单的异常的处理
114. @Bean
115. **public** SimpleMappingExceptionResolver simpleMappingExceptionResolver(){
116. SimpleMappingExceptionResolver exceptionResolver = **new** SimpleMappingExceptionResolver();
117. Properties mappings = **new** Properties();
118. mappings.put("UnauthorizedException","refuse"); //参数1：异常类名 参数2：逻辑视图名
119. mappings.put("UnauthenticatedException","refuse"); //参数1：异常类名 参数2：逻辑视图名
120. exceptionResolver.setExceptionMappings(mappings);
121. //返回
122. **return** exceptionResolver;
123. }
125. }

如果启动的时候报了异常 绝大部分可能是配置文件（或者用springboot的时候的配置信息的类@configuration的类）中的配置写错了

配置的总流程详解：

* **配置ShiroFilterFactoryBean** 看看它有什么属性：用它的对象去点set方法
* **根据**ShiroFilterFactoryBean来设置相应的Bean 每次设置Bean都用如上的方法
* **然后配置**SecurityManager (同样这是在上面的属性中有的也是shiro的必须组件)
* **在**SecurityManager中又有realm属性 所以接下来我们要配置Realm 的Bean
* **在realm中又有了 凭证匹配器的属性**
* **根据上面我们开始配置凭证匹配器**CredentialsMatcher
* **在凭证匹配器中又有了三个属性 这三个都是直接写死的直接设置就行**
* **在**SecurityManager还有一个缓存管理器**CacheManager属性 我们也需要配置所以这一步是配置缓存管理器**
* **再就是关于cookie放的验证码的信息 我们需要设置一个cookie对象**
* **配置RememberMe 管理器 这个是session来设置用户登录如果点击了 记住我 那么它的登录信息在session中存在一定的时间 下次打开浏览器直接访问不用登录**

**自定义realm**

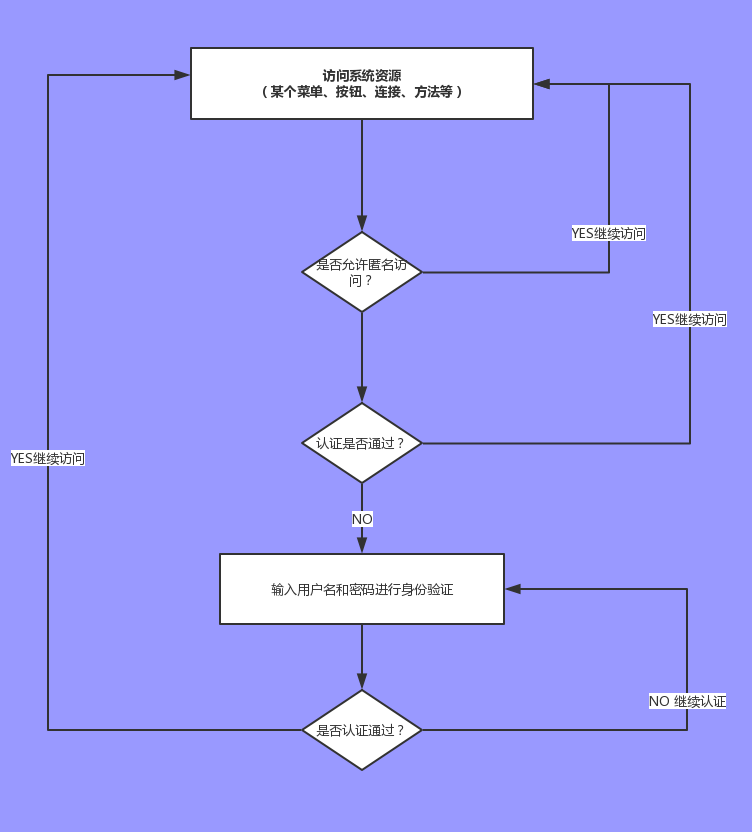
**下面是关于自定义realm的例子：**

1. **public** **class** CustomRealm **extends** AuthorizingRealm {
2. @Autowired
3. **private** UserService userService;
5. //1.授权
6. @Override
7. **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
8. //1.1)得到当前登录的用户对
9. SysUser user = (SysUser) principalCollection.getPrimaryPrincipal();
10. //1.2)判断是否为null
11. **if**(user != **null**){
12. //1.3)得到用户的权限列表
13. List<SysPermission> permissions = userService.findPermissionByUserCode(user.getUsercode());
14. //1.4)将上面的权限列表转换为字符串这种类型 的权限码列表
15. List<String> permissionList = **new** ArrayList<>();
16. //1.5)遍历权限列表，并且将权限列表放到permissionList这个集合中
17. **for** (SysPermission permission : permissions) {
18. permissionList.add(permission.getPercode());
19. }
20. System.out.println("正在访问数据库的权限列表....");
21. //1.6)定义返回的授权对象   //注意如果我们要的东西是一个接口或者抽象类那么就去找它的实现类 ctrl+h 查看继承关系
22. SimpleAuthorizationInfo simpleAuthenticationInfo = **new** SimpleAuthorizationInfo();
23. simpleAuthenticationInfo.addStringPermissions(permissionList);
24. //1.7)返回
25. **return** simpleAuthenticationInfo;
26. }
27. **return** **null**;
28. }
29. //2.认证
30. @Override
31. **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) **throws** AuthenticationException {
32. //2.1)得到当前登录的用户名
33. String userCode = (String) authenticationToken.getPrincipal();
34. //2.2)在数据库中根据usercode查询用户对象
35. SysUser user = userService.findUserByUserCode(userCode);
36. //2.3)判断user是否为null
37. **if**(user == **null**){
38. **return** **null**;
39. }
40. //2.4)如果不为null，就得到盐值
41. String salt = user.getSalt();
42. //2.5）为当前的用户指定菜单列表
43. user.setMenus(userService.findMenusByUsercode(userCode));
44. //2.5)返回
45. **return** **new** SimpleAuthenticationInfo(user,user.getPassword(),ByteSource.Util.bytes(salt),"aaa");
46. }
47. }

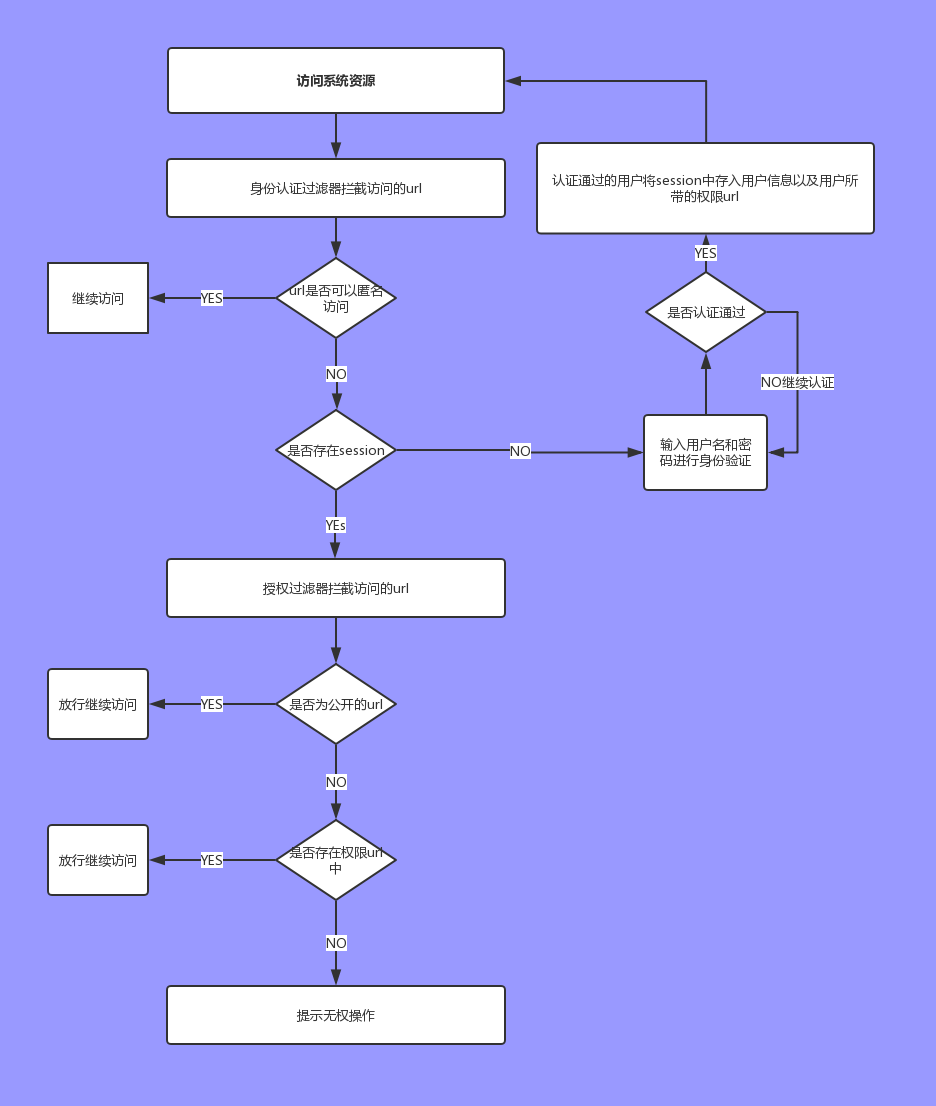
### shiro的认证授权流程

#### 授权流程：

* Shiro先会设置一个匿名的url列表 所有人都可以访问（不用登录也行）；
* 如果你输入了其他的url那么shiro就会进行拦截，让你跳转到登录页面或者是无权限访问；
* 如果你登录了就是授权问题了；
* 首先会判断是是否在公开的url（就是登录了的用户都可以访问的地方）；
* 如果你进向访问其他的资源那么shiro会进行拦截；
* 如果你通过了 授权 那么shiro就会放行；
* 后面的以此类推；



#### 2、授权流程:

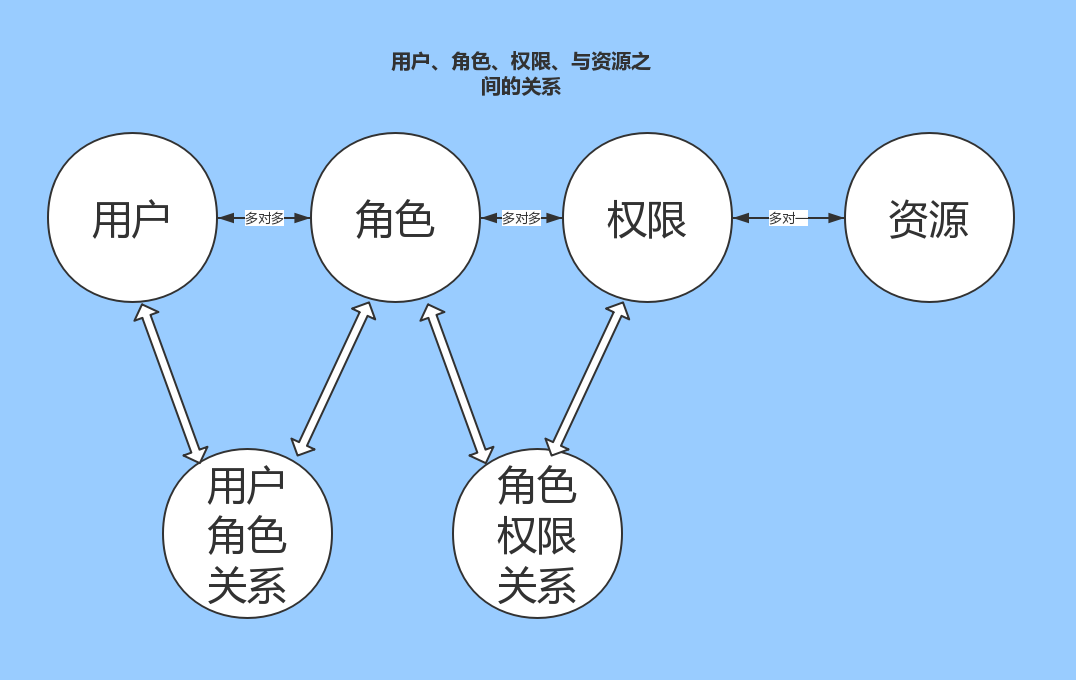


**流程分析：**

**第一步**：subject进行授权，调用isPermitted(“授权串”)方法进行。

**第二步**：SecurityManager执行授权，调用ModularRealmAuthorizer查找自定义realm，由自定义的realm在数据库中进行查找相应的权限串，由PermissionResolver（权限解析器）进行权限串的比（将isPermitted后的“权限串”与数据库中找到的权限串进行比对），如果存在则授权成功，否则，失败，报异常。

#### 3、用户、角色、权限、与资源之间的关系：



### ssm框架与shiro框架的整合

1. 配置文件

这次用xml文件距离配置文件其原理也是一样

#### spring的配置文件

头文件自己可以去官网查找

配置文件根据自己的需要 需要什么配置什么 但是不能少了必须的配置

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"**>**
6. <!--1.读取属性文件-->  //这是配置数据源的文件
7. **<context:property-placeholder** location="classpath\*:properties/\*.properties"**/>**
8. <!--1.1)配置包扫描（扫描四个注解位置在与springboot启动器同一个层次下的包）-->
9. **<context:component-scan** base-package="com.zelin"**/>**
10. <!--2.定义连接池-->
11. **<bean** id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"**>**
12. **<property** name="driverClassName" value="${jdbc.driver}"**/>**
13. **<property** name="url" value="${jdbc.url}"**/>**
14. **<property** name="username" value="${jdbc.user}"**/>**
15. **<property** name="password" value="${jdbc.password}"**/>**
16. **</bean>**
17. <!--3.定义SqlSessionFactoryBean--> //mybatis的整合相关
18. **<bean** class="org.mybatis.spring.SqlSessionFactoryBean"**>**
19. **<property** name="dataSource" ref="dataSource"**/>**
20. **<property** name="mapperLocations" value="classpath\*:mapper/\*.xml"**/>**
21. **<property** name="typeAliasesPackage" value="com.zelin.pojo"**/>**
22. **<property** name="plugins"**>**
23. **<bean** class="com.github.pagehelper.PageHelper"**>**
24. **<property** name="properties"**>**
25. **<value>**
26. dialect=mysql
27. **</value>**
28. **</property>**
29. **</bean>**
30. **</property>**
31. **</bean>**
32. <!--4.定义MapperScannerConfigurer-->  //mybatis通用接口位置 被扫描
33. **<bean** class="tk.mybatis.spring.mapper.MapperScannerConfigurer"**>**
34. <!--4.1）定义mapper接口的扫描位置-->
35. **<property** name="basePackage" value="com.zelin.mapper"**/>**
36. **</bean>**
37. **</beans>**

#### springmvc的配置文件

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"
4. xmlns:context="http://www.springframework.org/schema/context"
5. xmlns:aop="http://www.springframework.org/schema/aop"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd"**>**
7. <!--0.控制器扫描-->
8. **<context:component-scan** base-package="com.zelin.controller"**/>**
9. <!--1.开启aop注解-->
10. **<aop:config** proxy-target-class="true"**/>**
11. <!--1.相当于配置了处理器映射器和处理器适配器-->
12. **<mvc:annotation-driven/>**
13. <!--2.配置视图解析器-->
14. **<bean** id="resourceViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
15. **<property** name="prefix" value="/WEB-INF/jsp/"**/>**
16. **<property** name="suffix" value=".jsp"**/>**
17. **</bean>**
18. <!--3.springmvc处理一般的异常映射-->
19. **<bean** class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver"**>**
20. **<property** name="exceptionMappings"**>**
21. **<props>**
22. **<prop** key="org.apache.shiro.authz.UnauthorizedException"**>**refuse**</prop>**
23. **<prop** key="org.apache.shiro.authz.UnauthenticatedException"**>**refuse**</prop>**
24. **</props>**
25. **</property>**
26. **</bean>**
27. **</beans>**

#### shiro的配置

这里还是列出来一遍吧 跟上面写的配置类差不多；

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<beans** xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"**>**
6. <!--shiro自带的常用过滤器如下：-->
7. <!--anon:例子/admins/\*\*=anon 没有参数，表示可以匿名使用。-->
8. <!--authc:例如/admins/user/\*\*=authc表示需要认证(登录)才能使用，FormAuthenticationFilter是表单认证，没有参数-->
9. <!--roles：例子/admins/user/\*\*=roles[admin],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，当有多个参数时，例如admins/user/\*\*=roles["admin,guest"],每个参数通过才算通过，相当于hasAllRoles()方法。-->
10. <!--perms：例子/admins/user/\*\*=perms[user:add:\*],参数可以写多个，多个时必须加上引号，并且参数之间用逗号分割，例如/admins/user/\*\*=perms["user:add:\*,user:modify:\*"]，当有多个参数时必须每个参数都通过才通过，想当于isPermitedAll()方法。-->
11. <!--user:例如/admins/user/\*\*=user没有参数表示必须存在用户, 身份认证通过或通过记住我认证通过的可以访问，当登入操作时不做检查-->
12. <!--Logout:代表退出过滤器，此url的action可以不定义，由shiro自动销毁session-->
13. <!--0.开启shiro对注解的支持-->
14. **<bean**  class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor"**>**
15. **<property** name="securityManager" ref="securityManager"**/>**
16. **</bean>**
17. <!--1.配置shiroFilterFactoryBean-->
18. **<bean** id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean"**>**
19. <!--1.1)配置安全管理器-->
20. **<property** name="securityManager" ref="securityManager"**/>**
21. <!--1.2)配置登录的url地址-->
22. **<property** name="loginUrl" value="/login.do"**/>**
23. <!--1.3)授权不通过时的访问页面-->
24. **<property** name="unauthorizedUrl" value="/refuse.jsp"**/>**
25. <!--1.4)认证通过后,跳转到的页面-->
26. **<property** name="successUrl" value="/user/listmenu.do"**/>**
27. <!--1.5)配置过滤器链（调用系统自带的默认过滤器）-->
28. **<property** name="filterChainDefinitions"**>**
29. **<value>**
30. <!--1.5.1)对静态资源进行放行-->
31. /plugins/\*\* = anon
32. <!--1.5.2)对验证码进行放行-->
33. /validatecode.jsp = anon
34. <!--1.5.3)为注消功能映射一个地址-->
35. /logoutlogout.do = logout
36. <!--配置授权过滤器, 第一种授权方式-->
37. <!--/user/userlistxx.do = perms[user:list]-->
38. <!--/user/userlist.do = perms[user:userlistxx]-->
39. <!--1.5.4)除了上面的三种情况外，对其它的资源访问必须要经过认证方可-->
40. /\*\* = authc
41. **</value>**
42. **</property>**
43. <!--1.6)配置自定义过滤器-->
44. **<property** name="filters"**>**
45. **<map>**
46. **<entry** key="authc" value-ref="customFormFilter"**/>**
47. **</map>**
48. **</property>**
49. **</bean>**
50. <!--2.配置安全管理器-->
51. **<bean** id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager"**>**
52. <!--2.1)自定义realm的配置-->
53. **<property** name="realm" ref="customRealm"**/>**
54. <!--2.2)配置缓存管理器-->
55. **<property** name="cacheManager" ref="cacheManager"**/>**
56. **</bean>**
57. <!--3.配置自定义的realm对象-->
58. **<bean** id="customRealm" class="com.zelin.realm.CustomRealm"**>**
59. <!--3.1)配置凭证匹配器-->
60. **<property** name="credentialsMatcher" ref="credentialsMatcher"**/>**
61. **</bean>**
62. <!--4.配置凭证匹配器对象-->
63. **<bean** id="credentialsMatcher" class="org.apache.shiro.authc.credential.HashedCredentialsMatcher"**>**
64. <!--4.1)配置加密的次数-->
65. **<property** name="hashIterations" value="1"**/>**
66. <!--4.2)配置加密算法-->
67. **<property** name="hashAlgorithmName" value="md5"**/>**
68. **</bean>**
69. <!--5.定义缓存管理器-->
70. **<bean** id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager"**>**
71. **<property** name="cacheManagerConfigFile" value="classpath:cache/shiro-cache.xml"**/>**
72. **</bean>**
73. **</beans>**

#### Maven中的依赖

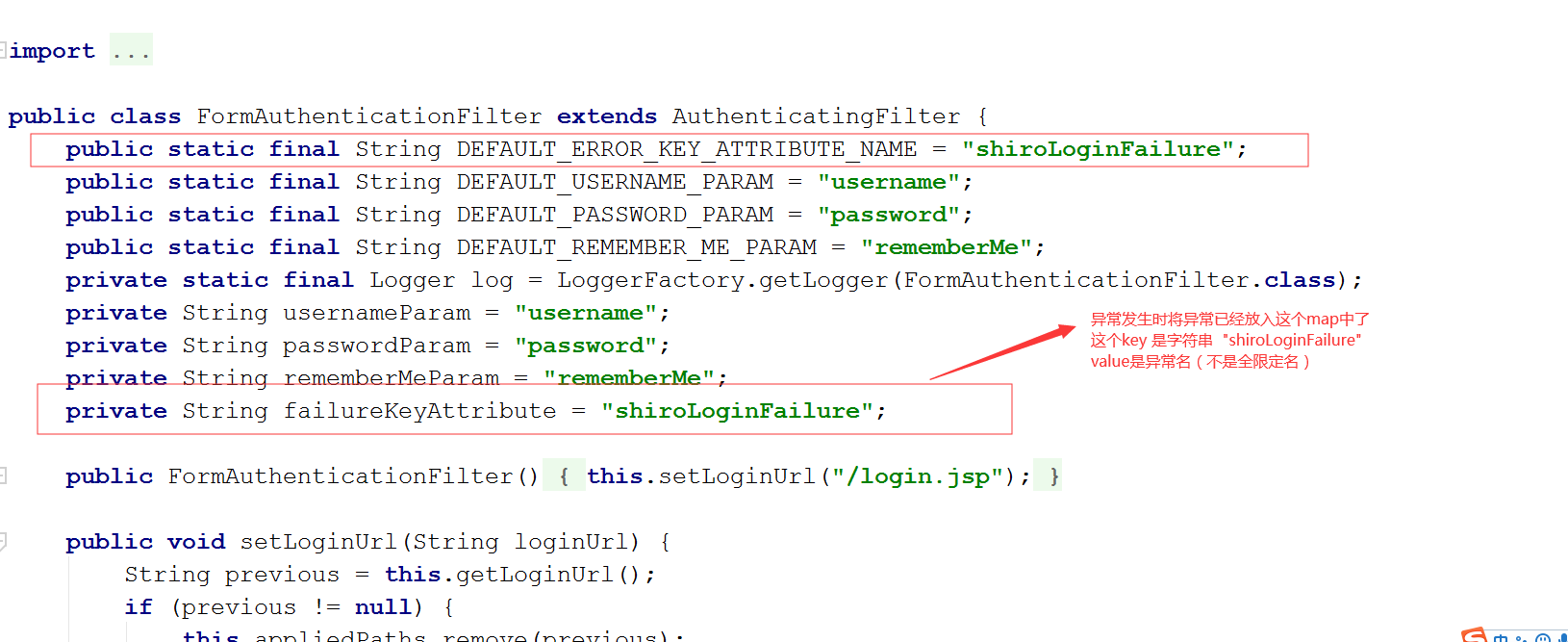
<**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**maven.compiler.source**>1.7</**maven.compiler.source**>  
 <**maven.compiler.target**>1.7</**maven.compiler.target**>  
 *<!-- 集中定义依赖版本号 -->* <**junit.version**>4.12</**junit.version**>  
 <**spring.version**>4.2.4.RELEASE</**spring.version**>  
 <**mybatis.version**>3.2.8</**mybatis.version**>  
 <**mybatis.spring.version**>1.2.2</**mybatis.spring.version**>  
 <**mybatis.paginator.version**>1.2.15</**mybatis.paginator.version**>  
 <**mysql.version**>5.1.32</**mysql.version**>  
 <**slf4j.version**>1.6.4</**slf4j.version**>  
 <**jackson.version**>2.4.2</**jackson.version**>  
 <**druid.version**>1.0.9</**druid.version**>  
 <**jstl.version**>1.2</**jstl.version**>  
 <**servlet-api.version**>2.5</**servlet-api.version**>  
 <**jsp-api.version**>2.0</**jsp-api.version**>  
 <**commons-lang3.version**>3.3.2</**commons-lang3.version**>  
 <**commons-io.version**>1.3.2</**commons-io.version**>  
 <**pagehelper.version**>4.0.0</**pagehelper.version**>  
 <**jsqlparser.version**>0.9.1</**jsqlparser.version**>  
 <**commons-fileupload.version**>1.3.1</**commons-fileupload.version**>  
</**properties**>  
  
  
<**dependencies**>  
 *<!-- Apache工具组件 -->* <**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-lang3</**artifactId**>  
 <**version**>${commons-lang3.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-io</**artifactId**>  
 <**version**>${commons-io.version}</**version**>  
 </**dependency**>  
 *<!-- Jackson Json处理工具包 -->* <**dependency**>  
 <**groupId**>com.fasterxml.jackson.core</**groupId**>  
 <**artifactId**>jackson-databind</**artifactId**>  
 <**version**>${jackson.version}</**version**>  
 </**dependency**>  
  
 *<!-- 单元测试 -->* <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>${junit.version}</**version**>  
 </**dependency**>  
 *<!-- 日志处理 -->* <**dependency**>  
 <**groupId**>org.slf4j</**groupId**>  
 <**artifactId**>slf4j-log4j12</**artifactId**>  
 <**version**>${slf4j.version}</**version**>  
 </**dependency**>  
 *<!-- Mybatis -->* <**dependency**>  
 <**groupId**>org.mybatis</**groupId**>  
 <**artifactId**>mybatis</**artifactId**>  
 <**version**>${mybatis.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.mybatis</**groupId**>  
 <**artifactId**>mybatis-spring</**artifactId**>  
 <**version**>${mybatis.spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.github.miemiedev</**groupId**>  
 <**artifactId**>mybatis-paginator</**artifactId**>  
 <**version**>${mybatis.paginator.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>com.github.pagehelper</**groupId**>  
 <**artifactId**>pagehelper</**artifactId**>  
 <**version**>${pagehelper.version}</**version**>  
 </**dependency**>  
 *<!-- MySql -->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>${mysql.version}</**version**>  
 </**dependency**>  
 *<!-- 连接池 -->* <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 <**version**>${druid.version}</**version**>  
 </**dependency**>  
 *<!-- Spring -->* <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-beans</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jdbc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-aspects</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jms</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context-support</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
 *<!-- JSP相关 -->* <**dependency**>  
 <**groupId**>jstl</**groupId**>  
 <**artifactId**>jstl</**artifactId**>  
 <**version**>${jstl.version}</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>servlet-api</**artifactId**>  
 <**version**>${servlet-api.version}</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>jsp-api</**artifactId**>  
 <**version**>${jsp-api.version}</**version**>  
 <**scope**>provided</**scope**>  
 </**dependency**>  
  
 *<!-- 关于shiro相关 -->* <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-core</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-web</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-spring</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-ehcache</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>tk.mybatis</**groupId**>  
 <**artifactId**>mapper</**artifactId**>  
 <**version**>4.1.5</**version**>  
 </**dependency**>  
</**dependencies**>  
<**build**>  
 <**finalName**>${project.artifactId}</**finalName**>  
 <**pluginManagement**>  
 <**plugins**>  
 *<!-- 配置Tomcat插件 -->* <**plugin**>  
 <**groupId**>org.apache.tomcat.maven</**groupId**>  
 <**artifactId**>tomcat7-maven-plugin</**artifactId**>  
 <**configuration**>  
 <**contextReloadable**>true</**contextReloadable**>  
 <**port**>9000</**port**>  
 <**path**>/</**path**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
 </**pluginManagement**>  
</**build**>

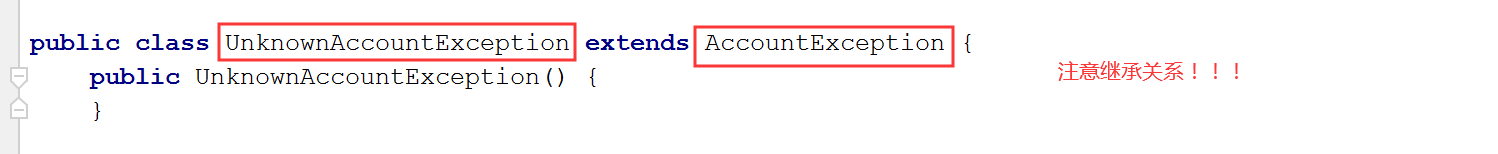
#### 5、异常的处理

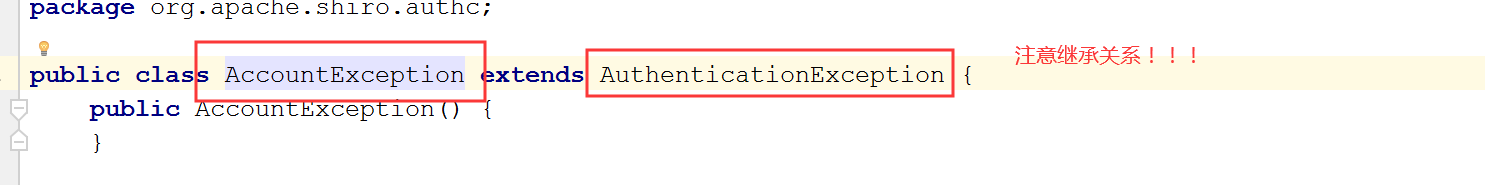
关于异常这里说的是shiro进行验证时产生的异常

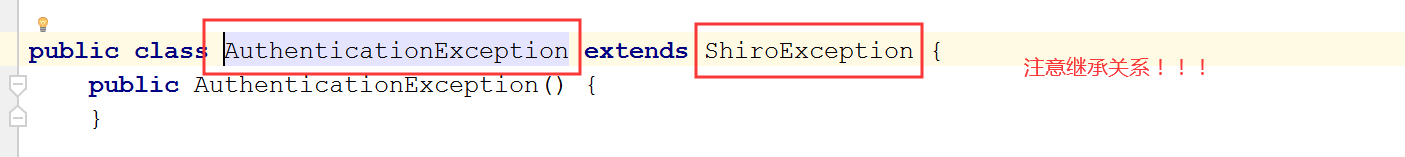
**步骤：**

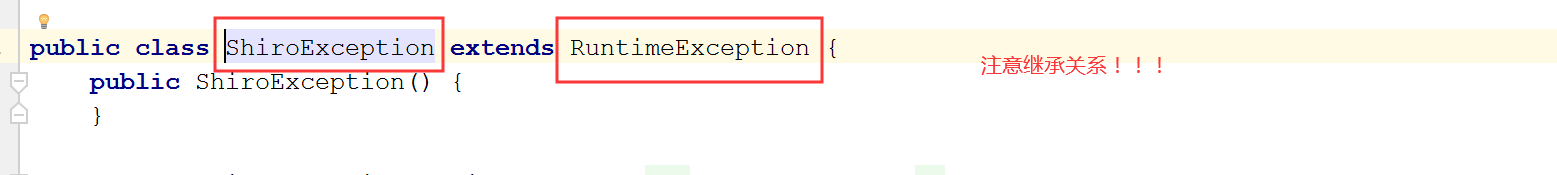
**第一步：**我们自定义一个异常类让它继承RunTimeException 这样我们就可以在异常发生时进行捕获 因为shrio框架抛出异常时 会将异常信息通过ModularRealmAuthenticator 类的方法抛出异常











异常抛出给shrio的异常类之后由我们捕获到 然后定义一个全局异常处理类 来处理这个异常；

自定义的异常类：

**public class** MyException **extends** RuntimeException {  
 **private** String **message**;  
  
 @Override  
 **public** String getMessage() {  
 **return message**;  
 }  
  
 **public void** setMessage(String message) {  
 **this**.**message** = message;  
 }  
  
 **public** MyException(String message) {  
 **this**.**message** = message;  
 }  
}

全局异常处理类：

1. @ControllerAdvice
2. **public** **class** MyExceptionHandler {
3. //1.下面的注解代表，只处理MyException这个异常
4. @ExceptionHandler(MyException.**class**)
5. **public** String error(Model model, Exception ex){
6. //1.1)定义出错的异常信息
7. String message = "";
8. //1.2)如果当前的异常存在，就得到错语信息
9. **if**(ex != **null**){
10. message = ex.getMessage();
11. }
12. //1.3)将这个信息放到model中
13. model.addAttribute("message",message);
14. //1.4)返回逻辑视图
15. **return** "error";
16. }
17. }

#### 6、我在跑项目的时候遇到的问题以及处理的办法

**第一个**

在给在遍历一个集合时不要对集合 进行修改操作 要不然会报

concurrentModifyException 因为在遍历的时候需要集合的size 而修改可能会改变集合的size 这样jvm就不知道集合的size 而遍历 又需要 ；

**第二个**

启动是报错提示创建Bean失败 这里就是Spring的配置出了问题 注意依赖关系 以及注入的时候id有没有完全对应！！！

### shrio与springboot整合

#### Maven中的依赖

*<!--1.配置下面的依赖所用到的父工程-->*<**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.1.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->*</**parent**>  
<**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**java.version**>1.8</**java.version**>  
</**properties**>  
<**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter</**artifactId**>  
 </**dependency**>  
 *<!--2.与junit进行整合的starter-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-test</**artifactId**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 *<!--3.代表springboot与web的整合的starter-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
 *<!--4.添加数据库-->* <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.1.47</**version**>  
 </**dependency**>  
 *<!--5.添加mybatis与springboot整合的依赖-->* <**dependency**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**version**>2.0.1</**version**>  
 </**dependency**>  
 *<!-- 6.添加mybatis分页插件与springboot整合的依赖 -->* <**dependency**>  
 <**groupId**>com.github.pagehelper</**groupId**>  
 <**artifactId**>pagehelper-spring-boot-starter</**artifactId**>  
 <**version**>1.2.10</**version**>  
 </**dependency**>  
 *<!-- 7.添加通过mapper与springboot整合的依赖 -->* <**dependency**>  
 <**groupId**>tk.mybatis</**groupId**>  
 <**artifactId**>mapper-spring-boot-starter</**artifactId**>  
 <**version**>2.1.5</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-lang3</**artifactId**>  
 <**version**>3.3.2</**version**>  
 </**dependency**>  
 *<!-- 关于shiro相关 -->* <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-core</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-web</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-spring</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.shiro</**groupId**>  
 <**artifactId**>shiro-ehcache</**artifactId**>  
 <**version**>1.3.2</**version**>  
 </**dependency**>  
 *<!--在springboot中添加热布署功能-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 </**dependency**>  
 *<!--添加对jstl的依赖-->* <**dependency**>  
 <**groupId**>jstl</**groupId**>  
 <**artifactId**>jstl</**artifactId**>  
 <**version**>1.2</**version**>  
 </**dependency**>  
 *<!-- 使用jsp引擎，springboot内置tomcat没有此依赖 -->* <**dependency**>  
 <**groupId**>org.apache.tomcat.embed</**groupId**>  
 <**artifactId**>tomcat-embed-jasper</**artifactId**>  
 <**scope**>provided</**scope**>  
 </**dependency**>

<?--小辣椒用来生成set get方法 以及构造方法-->  
 <**dependency**>  
 <**groupId**>org.projectlombok</**groupId**>  
 <**artifactId**>lombok</**artifactId**>  
 </**dependency**>  
</**dependencies**>  
<**build**>  
 <**plugins**>  
 *<!--如果使用springboot的热布署功能的话，需要添加如下的配置-->* <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 <**configuration**>  
 <**fork**>true</**fork**>  
 </**configuration**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

#### springboot的配置

**//定义端口**

**server:  
 port:** 9000  
*#定义数据源***spring:  
 datasource:  
 driver-class-name:** com.mysql.jdbc.Driver  
 **url:** jdbc:mysql:///shiro  
 **username:** root  
 **password:** root  
*#定义springmvc的视图解析器* **mvc:  
 view:  
 prefix:** /WEB-INF/jsp/  
 **suffix:** .jsp  
*#配置mybatis***mybatis:  
 mapper-locations:** mapper/\*.xml

#### shiro的配置类

@Configuration  
**public class** Myconfig {  
 *//配置shiroFilterFactoryBean;* @Bean  
 **public** ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager){  
 ShiroFilterFactoryBean shiroFilterFactoryBean = **new** ShiroFilterFactoryBean();  
 shiroFilterFactoryBean.setSecurityManager(securityManager);  
 shiroFilterFactoryBean.setLoginUrl(**"/login"**);  
 shiroFilterFactoryBean.setSuccessUrl(**"/user/listmenu"**);  
 shiroFilterFactoryBean.setUnauthorizedUrl(**"/refuse.jsp"**);  
 *//设置过滤器链* LinkedHashMap<String, String > filterChainMap = **new** LinkedHashMap<>();  
 *//向过滤器链中添加一系列的资源过滤  
 //放行静态资源* filterChainMap.put(**"/plugins/\*\*"**,**"anon"**);  
 *//放行验证码资源* filterChainMap.put(**"/validatecode.jsp"**,**"anon"**);  
 *//放行拒绝登陆页面的资源* filterChainMap.put(**"/refuse.jsp"**,**"anon"**);  
 *//放行登出页面的资源* filterChainMap.put(**"/logout"**,**"logout"**);  
 *//放行用户登陆后 的状态 能访问 /user/\*\* 的资源* filterChainMap.put(**"/user/\*\*"**,**"user"**);  
 *//其余所有的资源要进行拦截验证* filterChainMap.put(**"/\*\*"**,**"authc"**);  
 *//设置过滤器链 到FilterFactoryBean* shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainMap);  
  
 *//配置自定义的过滤器* LinkedHashMap<String, Filter> filterLinkedHashMap = **new** LinkedHashMap<>();  
 filterLinkedHashMap.put(**"authc"**,**new** MyFilter());  
 *//返回* **return** shiroFilterFactoryBean;  
 }  
 *//配置securityManager* @Bean  
 **public** SecurityManager securityManager(MyRealm realm,CacheManager cacheManager,RememberMeManager meManager){  
 DefaultWebSecurityManager securityManager = **new** DefaultWebSecurityManager();  
 *//设置各项属性  
 //设置自定义realm* securityManager.setRealm(realm);  
 *//设置缓存管理器* securityManager.setCacheManager(cacheManager);  
 *//设置记住我管理器* securityManager.setRememberMeManager(meManager);  
 *//返回* **return** securityManager;  
 }  
 *//配置自定义的Realm* @Bean  
 **public** MyRealm realm(CredentialsMatcher credentialsMatcher){  
 MyRealm realm = **new** MyRealm();  
 *//设置凭证匹配器* realm.setCredentialsMatcher(credentialsMatcher);  
 **return** realm;  
 }  
 *//配置缓存管理器* @Bean  
 **public** CacheManager cacheManager(){  
 EhCacheManager cacheManager = **new** EhCacheManager();  
 *//设置缓存管理器的配置文件所在的位置* cacheManager.setCacheManagerConfigFile(**"classpath:cache/shiro-cache.xml"**);  
 **return** cacheManager;  
 }  
 *//配置记住我管理器* @Bean  
 **public** RememberMeManager rememberMeManager(Cookie cookie){  
 CookieRememberMeManager meManager = **new** CookieRememberMeManager();  
 *//为记住我管理器设置Cookid值* meManager.setCookie(cookie);  
 **return** meManager;  
 }  
 *//配置Cookie* @Bean  
 **public** Cookie cookie(){  
 SimpleCookie cookie = **new** SimpleCookie();  
 *//设置到期时间 一个小时* cookie.setMaxAge(3600);  
 *//设置cookie的名字 与页面标签上的name属性一致 这样才能被验证* cookie.setName(**"rememberMe"**);  
 **return** cookie;  
 }  
  
 *//配置自定义的realm需要的凭证匹配器* @Bean  
 **public** CredentialsMatcher credentialsMatcher(){  
 HashedCredentialsMatcher credentialsMatcher = **new** HashedCredentialsMatcher();  
 *//为凭证匹配器对象设置属性  
 //算法名* credentialsMatcher.setHashAlgorithmName(**"md5"**);  
 *//加密次数* credentialsMatcher.setHashIterations(1);  
 **return** credentialsMatcher;  
 }  
  
 */\*\*  
 \* Spring的一个bean , 由Advisor决定对哪些类的方法进行AOP代理  
 \** ***@return*** *\*/* @Bean  
 **public** DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator(){  
 DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator = **new** DefaultAdvisorAutoProxyCreator();  
 defaultAdvisorAutoProxyCreator.setProxyTargetClass(**true**);  
 **return** defaultAdvisorAutoProxyCreator;  
 }  
  
  
 *//配置验证AuthorizationAttributeSourceAdvisor* @Bean  
 **public** AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){  
 AuthorizationAttributeSourceAdvisor advisor = **new** AuthorizationAttributeSourceAdvisor();  
 advisor.setSecurityManager(securityManager);  
 **return** advisor;  
 }  
  
 *//配置springmvc能进行一般的简单异常处理* @Bean  
 **public** SimpleMappingExceptionResolver simpleMappingExceptionResolver(){  
 SimpleMappingExceptionResolver resolver = **new** SimpleMappingExceptionResolver();  
 *//定义一个properties集合来接收* Properties properties = **new** Properties();  
 *//逐条设置属性* properties.put(**"UnauthorizedException"**,**"refuse"**);  
 properties.put(**"UnauthenticatedException"**,**"refuse"**);  
 resolver.setExceptionMappings(properties);  
 **return** resolver;  
 }  
}

#### shiro整合springboot的验证步骤

shiro会放行匿名访问地址，如登陆界面等等

点击登陆时 进行验证跳转到登陆控制器，同时shiro表单验证过滤器开始工作，经过处理，如果有异常我们的表单验证过滤器会将异常通过全局异常处理来进行处理（将异常信息发送到一个errorView界面）

如果验证通过，验证成功后shrio会跳转到成功页面！！

shrio对当前主体对象进行权限检查，权限检查通过时，才允许当前主体访问对应的资源

如果权限检查不通过，则shrio也会抛出一个无权限异常，又被我们的全局异常处器处理发送到相应的errorView界面

如果都通过则允许用户访问相应的资源。