# [Spring Boot整合jpa,Shiro进行权限管理](http://blog.csdn.net/liuchuanhong1/article/details/76179601)

标签： [spring boot](http://so.csdn.net/so/search/s.do?q=spring boot&t=blog" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [shiro](http://so.csdn.net/so/search/s.do?q=shiro&t=blog" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

2017年07月27日 12:24:37 2535人阅读 [评论](http://blog.csdn.net/liuchuanhong1/article/details/javascript:void(0);)(9) [收藏](http://blog.csdn.net/liuchuanhong1/article/details/javascript:void(0);" \o "收藏" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [举报](http://blog.csdn.net/liuchuanhong1/article/details/javascript:void(0);" \o "举报)

IMG_256 分类：

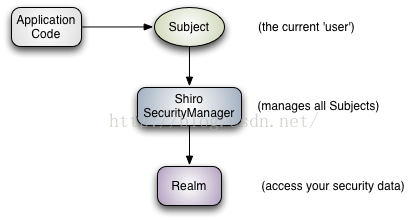
spring boot（45） IMG_257 shiro（3） IMG_258

转:http://412887952-qq-com.iteye.com/blog/2299777

本来想写一篇spring boot整合Shiro实现权限验证的文章，发现这篇写的非常不错，就直接借鉴了！

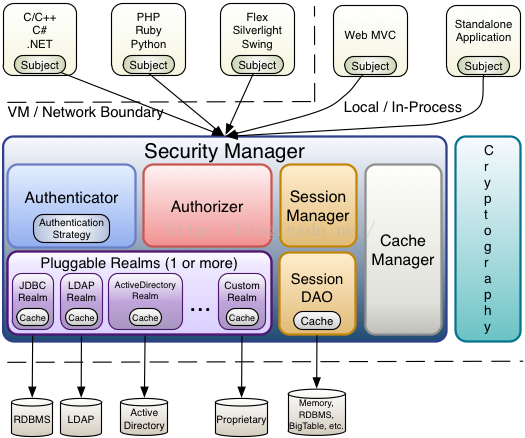
(1). Shiro简单介绍

Shiro是Apache下的一个开源项目，我们称之为Apache Shiro。它是一个很易用与**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)**项目的的安全框架，提供了认证、授权、加密、会话管理，与**[spring](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)** Security 一样都是做一个权限的安全框架，但是与Spring Security 相比，在于 Shiro 使用了比较简单易懂易于使用的授权方式。  
Apache Shiro 的三大核心组件



- Subject 当前用户操作   
- SecurityManager 用于管理所有的Subject   
- Realms 用于进行权限信息的验证，也是我们需要自己实现的。  
我们需要实现Realms的Authentication 和 Authorization。其中 Authentication 是用来验证用户身份，Authorization 是授权访问控制，用于对用户进行的操作授权，证明该用户是否允许进行当前操作，如访问某个链接，某个资源文件等。  
  
Apache Shiro 核心通过 Filter 来实现，就好像SpringMvc 通过DispachServlet 来主控制一样。   
既然是使用 Filter 一般也就能猜到，是通过URL规则来进行过滤和权限校验，所以我们需要定义一系列关于URL的规则和访问权限。  
另外我们可以通过Shiro 提供的会话管理来获取Session中的信息。Shiro 也提供了缓存支持，使用 CacheManager 来管理。  
  
官方网站：http://shiro.apache.org/

完整**[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)**图：



Shiro是很强大的一个安全框架，这里只是抛装引玉下，还有很多的需要大家自己去学习Shiro。

(2). 集成Shiro核心分析  
      集成Shiro的话，我们需要知道Shiro框架大概的一些管理对象。  
第一：ShiroFilterFactory，Shiro过滤器工厂类，具体的实现类是：ShiroFilterFactoryBean，此实现类是依赖于SecurityManager安全管理器。  
第二：SecurityManager,Shiro的安全管理，主要是身份认证的管理，缓存管理，cookie管理，所以在实际开发中我们主要是和SecurityManager进行打交道的，ShiroFilterFactory主要配置好了Filter就可以了。当然SecurityManager并进行身份认证缓存的实现，我们需要进行对应的编码然后进行注入到安全管理器中。  
第三：Realm,用于身份信息权限信息的验证。  
第四：其它的就是缓存管理，记住登录之类的，这些大部分都是需要自己进行简单的实现，然后注入到SecurityManager让Shiro的安全管理器进行管理就好了。  
  
(3). 无Shiro的Spring Boot  
      我们先编写一个无Shiro的简单的框架，在这个框架中我们可以访问到index,login,userInfo,userInfoAdd。  
      这个步骤对于有Spring Boot基础的就应该很简单了，在这里简单的介绍下：  
(a) 新建一个maven **[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)**project,取名为spring-boot-shiro1  
(b) 在pom.xml中引入基本依赖，在这里还没有引入shiro等的依赖：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
4. **<groupId>**com.example**</groupId>**
5. **<artifactId>**spring-boot-shiro1**</artifactId>**
6. **<version>**0.0.1-SNAPSHOT**</version>**
7. **<packaging>**jar**</packaging>**

10. **<properties>**
11. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
12. **</properties>**

15. <!-- Inherit defaults from Spring Boot -->
16. **<parent>**
17. **<groupId>**org.springframework.boot**</groupId>**
18. **<artifactId>**spring-boot-starter-parent**</artifactId>**
19. **<version>**1.4.0.RELEASE**</version>**
20. **</parent>**

23. **<dependencies>**
24. <!-- spring boot web支持：mvc,aop... -->
25. **<dependency>**
26. **<groupId>**org.springframework.boot**</groupId>**
27. **<artifactId>**spring-boot-starter-web**</artifactId>**
28. **</dependency>**
29. <!-- thmleaf模板依赖. -->
30. **<dependency>**
31. **<groupId>**org.springframework.boot**</groupId>**
32. **<artifactId>**spring-boot-starter-thymeleaf**</artifactId>**
33. **</dependency>**
35. <!-- 热部署 -->
36. **<dependency>**
37. **<groupId>**org.springframework.boot**</groupId>**
38. **<artifactId>**spring-boot-devtools**</artifactId>**
39. **<optional>**true**</optional>**
40. **</dependency>**
42. **</dependencies>**
43. **</project>**

(c) 编写网页文件:  
index.html,login.html,userInfo.html,userInfoAdd.html  
这个文件存在在src/main/resouces/templates, 这几个文件中都是简单的代码，只有登录界面中有账号和密码：

index.html

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="UTF-8" **/>**
5. **<title>**Insert title here**</title>**
6. **</head>**
7. **<body>**
8. **<h3>**index**</h3>**
9. **</body>**
10. **</html>**

login.html

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="UTF-8" **/>**
5. **<title>**Insert title here**</title>**
6. **</head>**
7. **<body>**
8. 错误信息：**<h4** th:text="${msg}"**></h4>**
9. **<form** action="" method="post"**>**
10. **<p>**账号：**<input** type="text" name="username" value="admin"**/></p>**
11. **<p>**密码：**<input** type="text" name="password" value="123456"**/></p>**
12. **<p><input** type="submit" value="登录"**/></p>**
13. **</form>**
14. **</body>**
15. **</html>**

userInfo.html,

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** charset="UTF-8" **/>**
5. **<title>**Insert title here**</title>**
6. **</head>**
7. **<body>**
8. **<h3>**用户查询界面**</h3>**
9. **</body>**
10. **</html>**

userInfoAdd.html

<h3>用户添加界面</h3>

(d)编写启动类

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example;
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
6. @SpringBootApplication
7. public class Application {
8. public static void main(String[] args) {
9. SpringApplication.run(Application.class, args);
10. }
11. }

(e)编写HomeController类  
新建HomeController类

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.controller;
3. import java.util.Map;
5. import javax.servlet.http.HttpServletRequest;
7. import org.apache.shiro.authc.IncorrectCredentialsException;
8. import org.apache.shiro.authc.UnknownAccountException;
9. import org.springframework.stereotype.Controller;
10. import org.springframework.web.bind.annotation.RequestMapping;
11. import org.springframework.web.bind.annotation.RequestMethod;
13. @Controller
14. public class HomeController {
15. @RequestMapping({ "/", "index" })
16. public String index() {
17. return "/index";
18. }
20. @RequestMapping(value = "/login", method = RequestMethod.GET)
21. public String login() {
22. return "/login";
23. }
24. }

集成shiro大概分这么一个步骤：  
(a) pom.xml中添加Shiro依赖；  
(b) 注入Shiro Factory和SecurityManager。  
(c) 身份认证  
(d) 权限控制

(a) pom.xml中添加Shiro依赖；

要使用Shiro进行权限控制，那么很明显的就需要添加对Shiro的依赖包，在pom.xml中加入如下配置：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. <!-- shiro spring. -->
2. **<dependency>**
3. **<groupId>**org.apache.shiro**</groupId>**
4. **<artifactId>**shiro-spring**</artifactId>**
5. **<version>**1.2.3**</version>**
6. **</dependency>**

(b) 注入Shiro Factory和SecurityManager。  
      在Spring中注入类都是使用配置文件的方式，在Spring Boot中是使用注解的方式，那么应该如何进行实现呢？  
      Shiro几个核心的类，第一就是ShiroFilterFactory,第二就是SecurityManager，那么最简单的配置就是注入这两个类就ok了，那么如何注入呢？看如下代码：

      新建ShiroConfiguration.java

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.config.shiro;
3. import java.util.LinkedHashMap;
4. import java.util.Map;
6. import org.apache.shiro.authc.credential.HashedCredentialsMatcher;
7. import org.apache.shiro.mgt.SecurityManager;
8. import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
9. import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
10. import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
11. import org.springframework.context.annotation.Bean;
12. import org.springframework.context.annotation.Configuration;
14. @Configuration
15. public class ShiroConfiguration {
17. @Bean
18. public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {
19. System.out.println("ShiroConfiguration.shiroFilter()");
20. ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
22. // 必须设置SecuritManager
23. shiroFilterFactoryBean.setSecurityManager(securityManager);
25. // 拦截器
26. Map**<String**, String**>** filterChainDefinitionMap = new LinkedHashMap**<String**, String**>**();
28. // 配置退出过滤器,其中的具体代码Shiro已经替我们实现了
29. filterChainDefinitionMap.put("/logout", "logout");
31. // <!-- 过滤链定义，从上向下顺序执行，一般将 /\*\*放在最为下边 -->:这是一个坑呢，一不小心代码就不好使了;
32. // <!-- authc:所有url都必须认证通过才可以访问; anon:所有url都都可以匿名访问-->
34. filterChainDefinitionMap.put("/\*\*", "authc");
36. // 如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面
37. shiroFilterFactoryBean.setLoginUrl("/login");
38. // 登录成功后要跳转的链接
39. shiroFilterFactoryBean.setSuccessUrl("/index");
40. // 未授权界面;
41. shiroFilterFactoryBean.setUnauthorizedUrl("/403");
43. shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
44. return shiroFilterFactoryBean;
46. }
48. @Bean
49. public SecurityManager securityManager() {
50. DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
52. return securityManager;
53. }


57. }

   这里说下：ShiroFilterFactory中已经由Shiro官方实现的过滤器：  
Shiro内置的FilterChain

|  |  |
| --- | --- |
| ****Filter Name**** | ****Class**** |
| anon | org.apache.shiro.web.filter.authc.AnonymousFilter |
| authc | org.apache.shiro.web.filter.authc.FormAuthenticationFilter |
| authcBasic | org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter |
| perms | org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter |
| port | org.apache.shiro.web.filter.authz.PortFilter |
| rest | org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter |
| roles | org.apache.shiro.web.filter.authz.RolesAuthorizationFilter |
| ssl | org.apache.shiro.web.filter.authz.SslFilter |
| user | org.apache.shiro.web.filter.authc.UserFilter |

anon:所有url都都可以匿名访问;  
authc: 需要认证才能进行访问;  
user:配置记住我或认证通过可以访问；  
这几个是我们会用到的，在这里说明下，其它的请自行查询文档进行学习。  
这时候我们运行程序，访问/index页面我们会发现自动跳转到了login页面，当然这个时候输入账号和密码是无法进行访问的。下面这才是重点：任何身份认证，如何权限控制。  
(c) 身份认证  
      在认证、授权内部实现机制中都有提到，最终处理都将交给Real进行处理。因为在Shiro中，最终是通过Realm来获取应用程序中的用户、角色及权限信息的。通常情况下，在Realm中会直接从我们的数据源中获取Shiro需要的验证信息。可以说，Realm是专用于安全框架的DAO.  
认证实现  
Shiro的认证过程最终会交由Realm执行，这时会调用Realm的getAuthenticationInfo(token)方法。  
该方法主要执行以下操作:  
1、检查提交的进行认证的令牌信息  
2、根据令牌信息从数据源(通常为数据库)中获取用户信息  
3、对用户信息进行匹配验证。  
4、验证通过将返回一个封装了用户信息的AuthenticationInfo实例。  
5、验证失败则抛出AuthenticationException异常信息。  
而在我们的应用程序中要做的就是自定义一个Realm类，继承AuthorizingRealm抽象类，重载doGetAuthenticationInfo ()，重写获取用户信息的方法。  
既然需要进行身份权限控制，那么少不了创建用户实体类，权限实体类。  
      在权限管理系统中，有这么几个角色很重要，这个要是不清楚的话，那么就很难理解，我们为什么这么编码了。第一是用户表：在用户表中保存了用户的基本信息，账号、密码、姓名，性别等；第二是：权限表（资源+控制权限）：这个表中主要是保存了用户的URL地址，权限信息；第三就是角色表：在这个表重要保存了系统存在的角色；第四就是关联表：用户-角色管理表（用户在系统中都有什么角色，比如admin，vip等），角色-权限关联表（每个角色都有什么权限可以进行操作）。依据这个理论，我们进行来进行编码，很明显的我们第一步就是要进行实体类的创建。在这里我们使用Mysql和JPA进行操作数据库。  
那么我们先在pom.xml中引入mysql和JPA的依赖：

UserInfo.java、SysRole.java、SysPermission.java至于之前的关联表我们使用JPA进行自动生成。

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. <!-- Spirng data JPA依赖; -->
2. **<dependency>**
3. **<groupId>**org.springframework.boot**</groupId>**
4. **<artifactId>**spring-boot-starter-data-jpa**</artifactId>**
5. **</dependency>**
7. <!-- mysql驱动; -->
8. **<dependency>**
9. **<groupId>**mysql**</groupId>**
10. **<artifactId>**mysql-connector-java**</artifactId>**
11. **</dependency>**

配置src/main/resouces/application.properties配置数据库和jpa(application.properties新建一个即可):

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. ########################################################
2. ###datasource
3. ########################################################
4. spring.datasource.url = jdbc:mysql://localhost:3306/test
5. spring.datasource.username = root
6. spring.datasource.password = root
7. spring.datasource.driverClassName = com.mysql.jdbc.Driver
8. spring.datasource.max-active=20
9. spring.datasource.max-idle=8
10. spring.datasource.min-idle=8
11. spring.datasource.initial-size=10


15. ########################################################
16. ### Java Persistence Api
17. ########################################################
18. # Specify the DBMS
19. spring.jpa.database = MYSQL
20. # Show or not log for each sql query
21. spring.jpa.show-sql = true
22. # Hibernate ddl auto (create, create-drop, update)
23. spring.jpa.hibernate.ddl-auto = update
24. # Naming strategy
25. #[org.hibernate.cfg.ImprovedNamingStrategy | org.hibernate.cfg.DefaultNamingStrategy]
26. spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.DefaultNamingStrategy
27. # stripped before adding them to the entity manager)
28. spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

准备工作准备好之后，那么就可以编写实体类了：

UserInfo.java

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.domain;
3. import java.io.Serializable;
4. import java.util.List;
6. import javax.persistence.Column;
7. import javax.persistence.Entity;
8. import javax.persistence.FetchType;
9. import javax.persistence.GeneratedValue;
10. import javax.persistence.Id;
11. import javax.persistence.JoinColumn;
12. import javax.persistence.JoinTable;
13. import javax.persistence.ManyToMany;
15. /\*\*
16. \* 用户信息.
17. \* @author Administrator
18. \*
19. \*/
20. @Entity
21. public class UserInfo implements Serializable {
23. /\*\*
24. \*
25. \*/
26. private static final long serialVersionUID = 1L;
28. @Id
29. @GeneratedValue
30. private long uid;// 用户id
32. @Column(unique = true)
33. private String username;// 帐号
35. private String name;// 名称（昵称或者真实姓名，不同系统不同定义）
37. private String password; // 密码;
38. private String salt;// 加密密码的盐
40. private byte state;// 用户状态,0:创建未认证（比如没有激活，没有输入验证码等等）--等待验证的用户 ,
41. // 1:正常状态,2：用户被锁定.
43. @ManyToMany(fetch = FetchType.EAGER) // 立即从数据库中进行加载数据
44. @JoinTable(name = "SysUserRole", joinColumns = { @JoinColumn(name = "uid") }, inverseJoinColumns = {
45. @JoinColumn(name = "roleId") })
46. private List**<SysRole>** roleList;// 一个用户具有多个角色
48. public long getUid() {
49. return uid;
50. }
52. public void setUid(long uid) {
53. this.uid = uid;
54. }
56. public String getUsername() {
57. return username;
58. }
60. public void setUsername(String username) {
61. this.username = username;
62. }
64. public String getName() {
65. return name;
66. }
68. public void setName(String name) {
69. this.name = name;
70. }
72. public String getPassword() {
73. return password;
74. }
76. public void setPassword(String password) {
77. this.password = password;
78. }
80. public String getSalt() {
81. return salt;
82. }
84. public void setSalt(String salt) {
85. this.salt = salt;
86. }
88. public byte getState() {
89. return state;
90. }
92. public void setState(byte state) {
93. this.state = state;
94. }
96. public List**<SysRole>** getRoleList() {
97. return roleList;
98. }
100. public void setRoleList(List**<SysRole>** roleList) {
101. this.roleList = roleList;
102. }
104. /\*\*
105. \* 密码盐.
106. \*
107. \* @return
108. \*/
109. public String getCredentialsSalt() {
110. return this.username + this.salt;
111. }
113. @Override
114. public String toString() {
115. return "UserInfo [uid=" + uid + ", username=" + username + ", name=" + name + ", password=" + password
116. + ", salt=" + salt + ", state=" + state + "]";
117. }
119. }

在这里salt主要是用来进行密码加密的，当然也可以使用明文进行编码**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)**，实际开发中还是建议密码进行加密。  
getCredentialsSalt()  
这个方法重新对盐重新进行了定义，用户名+salt，这样就更加不容易被破解了

SysRole.java

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.domain;
3. import java.io.Serializable;
4. import java.util.List;
6. import javax.persistence.Entity;
7. import javax.persistence.FetchType;
8. import javax.persistence.GeneratedValue;
9. import javax.persistence.Id;
10. import javax.persistence.JoinColumn;
11. import javax.persistence.JoinTable;
12. import javax.persistence.ManyToMany;
14. /\*\*
15. \* 系统角色实体类;
16. \*
17. \* @author Administrator
18. \*
19. \*/
20. @Entity
21. public class SysRole implements Serializable {
22. private static final long serialVersionUID = 1L;
23. @Id
24. @GeneratedValue
25. private Long id; // 编号
26. private String role; // 角色标识程序中判断使用,如"admin",这个是唯一的:
27. private String description; // 角色描述,UI界面显示使用
28. private Boolean available = Boolean.FALSE; // 是否可用,如果不可用将不会添加给用户
30. // 角色 -- 权限关系：多对多关系;
31. @ManyToMany(fetch = FetchType.EAGER)
32. @JoinTable(name = "SysRolePermission", joinColumns = { @JoinColumn(name = "roleId") }, inverseJoinColumns = {
33. @JoinColumn(name = "permissionId") })
34. private List**<SysPermission>** permissions;
36. // 用户 - 角色关系定义;
37. @ManyToMany
38. @JoinTable(name = "SysUserRole", joinColumns = { @JoinColumn(name = "roleId") }, inverseJoinColumns = {
39. @JoinColumn(name = "uid") })
40. private List**<UserInfo>** userInfos;// 一个角色对应多个用户
42. public List**<UserInfo>** getUserInfos() {
43. return userInfos;
44. }
46. public void setUserInfos(List**<UserInfo>** userInfos) {
47. this.userInfos = userInfos;
48. }
50. public Long getId() {
51. return id;
52. }
54. public void setId(Long id) {
55. this.id = id;
56. }
58. public String getRole() {
59. return role;
60. }
62. public void setRole(String role) {
63. this.role = role;
64. }
66. public String getDescription() {
67. return description;
68. }
70. public void setDescription(String description) {
71. this.description = description;
72. }
74. public Boolean getAvailable() {
75. return available;
76. }
78. public void setAvailable(Boolean available) {
79. this.available = available;
80. }
82. public List**<SysPermission>** getPermissions() {
83. return permissions;
84. }
86. public void setPermissions(List**<SysPermission>** permissions) {
87. this.permissions = permissions;
88. }
90. @Override
91. public String toString() {
92. return "SysRole [id=" + id + ", role=" + role + ", description=" + description + ", available=" + available
93. + ", permissions=" + permissions + "]";
94. }
95. }

SysPermission.java

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.domain;
3. import java.io.Serializable;
4. import java.util.List;
6. import javax.persistence.Column;
7. import javax.persistence.Entity;
8. import javax.persistence.FetchType;
9. import javax.persistence.GeneratedValue;
10. import javax.persistence.Id;
11. import javax.persistence.JoinColumn;
12. import javax.persistence.JoinTable;
13. import javax.persistence.ManyToMany;
15. /\*\*
16. \* 权限实体类;
17. \*
18. \*/
19. @Entity
20. public class SysPermission implements Serializable {
21. private static final long serialVersionUID = 1L;
23. @Id
24. @GeneratedValue
25. private long id;// 主键.
26. private String name;// 名称.
28. @Column(columnDefinition = "enum('menu','button')")
29. private String resourceType;// 资源类型，[menu|button]
30. private String url;// 资源路径.
31. private String permission; // 权限字符串,menu例子：role:\*，button例子：role:create,role:update,role:delete,role:view
32. private Long parentId; // 父编号
33. private String parentIds; // 父编号列表
34. private Boolean available = Boolean.FALSE;
36. //  @ManyToMany(fetch = FetchType.LAZY)
37. //  @JoinTable(name = "SysRolePermission", joinColumns = { @JoinColumn(name = "permissionId") }, inverseJoinColumns = {
38. //          @JoinColumn(name = "roleId") })
39. //  private List**<SysRole>** roles;
41. public long getId() {
42. return id;
43. }
45. public void setId(long id) {
46. this.id = id;
47. }
49. public String getName() {
50. return name;
51. }
53. public void setName(String name) {
54. this.name = name;
55. }
57. public String getResourceType() {
58. return resourceType;
59. }
61. public void setResourceType(String resourceType) {
62. this.resourceType = resourceType;
63. }
65. public String getUrl() {
66. return url;
67. }
69. public void setUrl(String url) {
70. this.url = url;
71. }
73. public String getPermission() {
74. return permission;
75. }
77. public void setPermission(String permission) {
78. this.permission = permission;
79. }
81. public Long getParentId() {
82. return parentId;
83. }
85. public void setParentId(Long parentId) {
86. this.parentId = parentId;
87. }
89. public String getParentIds() {
90. return parentIds;
91. }
93. public void setParentIds(String parentIds) {
94. this.parentIds = parentIds;
95. }
97. public Boolean getAvailable() {
98. return available;
99. }
101. public void setAvailable(Boolean available) {
102. this.available = available;
103. }
105. //  public List**<SysRole>** getRoles() {
106. //      return roles;
107. //  }
108. //
109. //  public void setRoles(List**<SysRole>** roles) {
110. //      this.roles = roles;
111. //  }
113. @Override
114. public String toString() {
115. return "SysPermission [id=" + id + ", name=" + name + ", resourceType=" + resourceType + ", url=" + url
116. + ", permission=" + permission + ", parentId=" + parentId + ", parentIds=" + parentIds + ", available="
117. + available + "]";
118. }
120. }

ok，到这里实体类就编码完毕了，这时候运行Application，就会自动建表

**[MySQL](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)**> show tables;  
+---------------------+  
| Tables\_in\_test      |  
+---------------------+  
| sys\_permission      |  
| sys\_role            |  
| sys\_role\_permission |  
| sys\_user\_role       |  
| user\_info           |  
+---------------------+  
5 rows in set (0.08 sec)  
  
**[mysql](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)**>

sql

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. INSERT INTO `sys\_permission`(name,parent\_id,parent\_ids,available,permission,resource\_type,url)
2. VALUES ('用户管理',0,'0/' ,1,'userInfo:view', 'menu', 'userInfo/userList');
4. INSERT INTO `sys\_permission`(name,parent\_id,parent\_ids,available,permission,resource\_type,url)
5. VALUES ('用户添加',1,'0/1',1,'userInfo:add', 'button', 'userInfo/userAdd');
7. INSERT INTO `sys\_permission`(name,parent\_id,parent\_ids,available,permission,resource\_type,url)
8. VALUES ('用户删除',1,'0/1',1,'userInfo:del', 'button', 'userInfo/userDel');
10. INSERT INTO `sys\_role`(available,description,role) VALUES (1,'管理员','admin');
11. INSERT INTO `sys\_role`(available,description,role) VALUES (1,'VIP会员','vip');
13. INSERT INTO `sys\_role\_permission`(permission\_id,role\_id) VALUES ('1', '1');
14. INSERT INTO `sys\_role\_permission`(permission\_id,role\_id) VALUES ('2', '1');
16. INSERT INTO `user\_info`(name,password,salt,state,username) VALUES ('管理员', 'd3c59d25033dbf980d29554025c23a75', '8d78869f470951332959580424d4bf4f', '0', 'admin');
18. INSERT INTO `sys\_user\_role`(uid,role\_id) VALUES (1,1);
19. INSERT INTO `sys\_user\_role`(uid,role\_id) VALUES (1,2);

这时候数据都准备完毕了，那么接下来就应该编写Repository进行访问数据了

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.repository;
3. import org.springframework.data.repository.CrudRepository;
5. import com.example.domain.UserInfo;
7. /\*\*
8. \* UserInfo持久化类
9. \*
10. \* @author Administrator
11. \*
12. \*/
13. public interface UserInfoRepository extends CrudRepository**<UserInfo**, Long**>** {
14. /\*\* 通过username查找用户信息 \*\*/
15. public UserInfo findByUsername(String username);
17. }

在这里你会发现我们只编写了UserInfo的数据库操作，那么我们怎么获取我们的权限信息了，通过userInfo.getRoleList()可以获取到对应的角色信息，然后在通过对应的角色可以获取到权限信息，当然这些都是JPA帮我们实现了，我们也可以进行直接获取到权限信息，只要写一个关联查询然后过滤掉重复的权限即可，这里不进行实现。  
编写一个业务处理类UserInfoService>

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.service;
3. import com.example.domain.UserInfo;
5. public interface UserInfoService {
7. public UserInfo findByUsername(String username);
9. }

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.service.impl;
3. import javax.annotation.Resource;
5. import org.springframework.stereotype.Service;
6. import org.springframework.transaction.annotation.Transactional;
8. import com.example.domain.UserInfo;
9. import com.example.repository.UserInfoRepository;
10. import com.example.service.UserInfoService;
12. @Service
13. public class UserInfoServiceImpl implements UserInfoService{
14. @Resource
15. private UserInfoRepository userInfoRepository;
17. @Transactional(readOnly=true)
18. @Override
19. public UserInfo findByUsername(String username) {
20. System.out.println("UserInfoServiceImpl.findByUsername()");
21. return userInfoRepository.findByUsername(username);
22. }
24. }

基本工作准备好之后，剩下的才是重点，shiro的认证最终是交给了Realm进行执行了，所以我们需要自己重新实现一个Realm，此Realm继承AuthorizingRealm。

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.config.shiro;
3. import javax.annotation.Resource;
5. import org.apache.shiro.authc.AuthenticationException;
6. import org.apache.shiro.authc.AuthenticationInfo;
7. import org.apache.shiro.authc.AuthenticationToken;
8. import org.apache.shiro.authc.SimpleAuthenticationInfo;
9. import org.apache.shiro.authz.AuthorizationInfo;
10. import org.apache.shiro.authz.SimpleAuthorizationInfo;
11. import org.apache.shiro.realm.AuthorizingRealm;
12. import org.apache.shiro.subject.PrincipalCollection;
13. import org.apache.shiro.util.ByteSource;
14. import org.springframework.context.annotation.Bean;
16. import com.example.domain.SysPermission;
17. import com.example.domain.SysRole;
18. import com.example.domain.UserInfo;
19. import com.example.service.UserInfoService;
21. /\*\*
22. \* 身份校验核心类
23. \*
24. \* @author Administrator
25. \*
26. \*/
27. public class MyShiroRealm extends AuthorizingRealm {
29. @Resource
30. private UserInfoService userInfoService;
32. /\*\*
33. \* 认证信息(身份验证) Authentication 是用来验证用户身份
34. \*/
35. @Override
36. protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
37. System.out.println("MyShiroRealm.doGetAuthenticationInfo()");
38. // 获取用户的输入帐号
39. String username = (String) token.getPrincipal();
40. System.out.println(token.getCredentials());
41. // 通过username从数据库中查找 User对象，如果找到，没找到.
42. // 实际项目中，这里可以根据实际情况做缓存，如果不做，Shiro自己也是有时间间隔机制，2分钟内不会重复执行该方法
43. UserInfo userInfo = userInfoService.findByUsername(username);
44. System.out.println("-----**>>**userInfo=" + userInfo);
45. if (userInfo == null) {
46. return null;
47. }
49. SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(userInfo, // 用户名
50. userInfo.getPassword(), // 密码
51. ByteSource.Util.bytes(userInfo.getCredentialsSalt()), // salt=username+salt
52. getName() // realm name
53. );
54. return authenticationInfo;
55. }
57. @Override
58. protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
59. // TODO Auto-generated method stub
60. System.out.println("权限配置--**>**MyShiroRealm.doGetAuthorizationInfo()");
62. SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
63. UserInfo userInfo = (UserInfo) principals.getPrimaryPrincipal();
65. for(SysRole role:userInfo.getRoleList()){
67. authorizationInfo.addRole(role.getRole());
68. System.out.println(role.getPermissions());
69. for(SysPermission p:role.getPermissions()){
70. System.out.println(p);
71. authorizationInfo.addStringPermission(p.getPermission());
72. }
73. }
74. return authorizationInfo;
75. }

78. }

继承AuthorizingRealm主要需要实现两个方法：  
doGetAuthenticationInfo();  
doGetAuthorizationInfo();  
其中doGetAuthenticationInfo主要是用来进行身份认证的，也就是说验证用户输入的账号和密码是否正确。

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. SimpleAuthenticationInfoauthenticationInfo =
2. new SimpleAuthenticationInfo(
3. userInfo, //用户名
4. userInfo.getPassword(), //密码
5. ByteSource.Util.bytes(userInfo.getCredentialsSalt()),//salt=username+salt
6. getName()  //realm name
7. );

交给AuthenticatingRealm使用CredentialsMatcher进行密码匹配，如果觉得人家的不好可以自定义实现  
如果你是进行明文进行编码的话，那么使用使用如下方式：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(
2. userInfo, //用户名
3. userInfo.getPassword(), //密码
4. getName()  //realm name
5. );

至于doGetAuthorizationInfo()是权限控制，当访问到页面的时候，使用了相应的注解或者shiro标签才会执行此方法否则不会执行，所以如果只是简单的身份认证没有权限的控制的话，那么这个方法可以不进行实现，直接返回null即可。  
在这个方法中主要是使用类：SimpleAuthorizationInfo  
进行角色的添加和权限的添加。  
authorizationInfo.addRole(role.getRole());  
authorizationInfo.addStringPermission(p.getPermission());  
当然也可以添加集合：  
authorizationInfo.setRoles(roles);  
authorizationInfo.setStringPermissions(stringPermissions);

到这里我们还需要有一个步骤很重要就是将我们自定义的Realm注入到SecurityManager中。

在ShiroConfiguration.java中添加方法

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. /\*\*
2. \* 身份认证realm;
3. \*
4. \*/
5. @Bean
6. public MyShiroRealm myShiroRealm(){
7. MyShiroRealm myShiroRealm = new MyShiroRealm();
8. return myShiroRealm;
9. }

将myShiroRealm注入到securityManager中：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. @Bean
2. public SecurityManager securityManager(){
3. DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
4. //设置realm.
5. securityManager.setRealm(myShiroRealm());
6. return securityManager;
7. }

到这里的话身份认证权限控制基本是完成了，最后我们在编写一个登录的时候，登录的处理：  
在HomeController中添加login post处理：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.controller;
3. import java.util.Map;
5. import javax.servlet.http.HttpServletRequest;
7. import org.apache.shiro.authc.IncorrectCredentialsException;
8. import org.apache.shiro.authc.UnknownAccountException;
9. import org.springframework.stereotype.Controller;
10. import org.springframework.web.bind.annotation.RequestMapping;
11. import org.springframework.web.bind.annotation.RequestMethod;
13. @Controller
14. public class HomeController {
15. @RequestMapping({ "/", "index" })
16. public String index() {
17. return "/index";
18. }
20. @RequestMapping(value = "/login", method = RequestMethod.GET)
21. public String login() {
22. return "/login";
23. }
25. @RequestMapping(value = "/login", method = RequestMethod.POST)
26. public String login(HttpServletRequest request, Map**<String**, Object**>** map) {
27. System.out.println("HomeController.login");
28. // 登录失败从request中获取shiro处理的异常信息
29. // shiroLoginFailure:就是shiro异常类的全类名
30. String exception = (String) request.getAttribute("shiroLoginFailure");
31. String msg = "";
32. if (exception != null) {
33. if (UnknownAccountException.class.getName().equals(exception)) {
34. System.out.println("UnknownAccountException --**>**帐号不存在：");
35. msg = "UnknownAccountException -->帐号不存在：";
36. } else if (IncorrectCredentialsException.class.getName().equals(exception)) {
37. System.out.println("IncorrectCredentialsException -- **>** 密码不正确：");
38. msg = "IncorrectCredentialsException -- > 密码不正确：";
39. } else if ("kaptchaValidateFailed".equals(exception)) {
40. System.out.println("kaptchaValidateFailed -- **>** 验证码错误");
41. msg = "kaptchaValidateFailed -- > 验证码错误";
42. } else {
43. msg = "else >> " + exception;
44. System.out.println("else -- **>**" + exception);
45. }
46. }
47. map.put("msg", msg);
48. // 此方法不处理登录成功,由shiro进行处理.
49. return "/login";
50. }
51. }

这时候我们启动应用程序，访问http://127.0.0.1:8080/index  
会自动跳转到http://127.0.0.1:8080/login 界面，然后输入账号和密码：admin/123456,这时候会提示：IncorrectCredentialsException -- > 密码不正确。  
这主要是因为我们在上面进行了密文的方式，那么怎么加密方式，我们并没有告诉Shiro，所以认证失败了

在这里我们需要编写一个加密**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)**类，当然Shiro也已经有了具体的实现HashedCredentialsMatcher  
我们只需要进行注入使用即可：  
在ShiroConfiguration中加入方法：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. /\*\*
2. \* 凭证匹配器
3. \* （由于我们的密码校验交给Shiro的SimpleAuthenticationInfo进行处理了
4. \*  所以我们需要修改下doGetAuthenticationInfo中的代码;
5. \* ）
6. \* @return
7. \*/
8. @Bean
9. public HashedCredentialsMatcher hashedCredentialsMatcher(){
10. HashedCredentialsMatcher hashedCredentialsMatcher = new HashedCredentialsMatcher();
12. hashedCredentialsMatcher.setHashAlgorithmName("md5");//散列算法:这里使用MD5算法;
13. hashedCredentialsMatcher.setHashIterations(2);//散列的次数，比如散列两次，相当于 md5(md5(""));
15. return hashedCredentialsMatcher;
16. }

在myShiroRealm()方法中注入凭证匹配器：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. @Bean
2. public MyShiroRealm myShiroRealm(){
3. MyShiroRealm myShiroRealm = new MyShiroRealm();
4. myShiroRealm.setCredentialsMatcher(hashedCredentialsMatcher());;
5. return myShiroRealm;
6. }

这时候在访问/login进行登录就可以登陆到/index界面了。  
  
(d) 权限控制  
在我们新建一个UserInfoController

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.controller;
3. import org.apache.shiro.authz.annotation.RequiresPermissions;
4. import org.springframework.stereotype.Controller;
5. import org.springframework.web.bind.annotation.RequestMapping;
7. @Controller
8. @RequestMapping("userInfo")
9. public class UserInfoController {
10. /\*\*
11. \* 用户查询.
12. \* @return
13. \*/
14. @RequestMapping("/userList")
15. public String userInfo(){
16. return "userInfo";
17. }
19. /\*\*
20. \* 用户添加;
21. \* @return
22. \*/
23. @RequestMapping("/userAdd")
24. public String userInfoAdd(){
25. return "userInfoAdd";
26. }
27. /\*\*
28. \* 用户删除;
29. \* @return
30. \*/
31. @RequestMapping("/userDel")
32. @RequiresPermissions("userInfo:del")//权限管理;
33. public String userDel(){
34. return "userInfoDel";
35. }
36. }

然后运行登录进行访问：http://127.0.0.1:8080/userInfo/userAdd  
并没有执行doGetAuthorizationInfo()打印信息，所以我们会发现我们的身份认证是好使了，但是权限控制好像没有什么作用哦。

我们少了几部分代码，

第一就是开启shiro aop注解支持，这个只需要在ShiroConfiguration加入如下方法进行开启即可：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. /\*\*
2. \*  开启shiro aop注解支持.
3. \*  使用代理方式;所以需要开启代码支持;
4. \* @param securityManager
5. \* @return
6. \*/
7. @Bean
8. public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){
9. AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new AuthorizationAttributeSourceAdvisor();
10. authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
11. return authorizationAttributeSourceAdvisor;
12. }

第二就是在controller方法中加入相应的注解：

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.controller;
3. import org.apache.shiro.authz.annotation.RequiresPermissions;
4. import org.springframework.stereotype.Controller;
5. import org.springframework.web.bind.annotation.RequestMapping;
7. @Controller
8. @RequestMapping("userInfo")
9. public class UserInfoController {
10. /\*\*
11. \* 用户查询.
12. \* @return
13. \*/
14. @RequestMapping("/userList")
15. @RequiresPermissions("userInfo:view")//权限管理;
16. public String userInfo(){
17. return "userInfo";
18. }
20. /\*\*
21. \* 用户添加;
22. \* @return
23. \*/
24. @RequestMapping("/userAdd")
25. @RequiresPermissions("userInfo:add")//权限管理;
26. public String userInfoAdd(){
27. return "userInfoAdd";
28. }
29. /\*\*
30. \* 用户删除;
31. \* @return
32. \*/
33. @RequestMapping("/userDel")
34. @RequiresPermissions("userInfo:del")//权限管理;
35. public String userDel(){
36. return "userInfoDel";
37. }
38. }

这时候在访问http://127.0.0.1:8080/userInfo/userAdd 会看到控制台打印信息

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. 权限配置--**>**MyShiroRealm.doGetAuthorizationInfo()

如果访问：http://127.0.0.1:8080/userInfo/userDel会看到控制台打印信息

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. org.apache.shiro.authz.AuthorizationException: Not authorized to invoke method

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. 页面上看到

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. Whitelabel Error Page
3. This application has no explicit mapping for /error, so you are seeing this as a fallback.
5. Sun Aug 28 21:36:31 CST 2016
6. There was an unexpected error (type=Internal Server Error, status=500).
7. Subject does not have permission [userInfo:del]

----------------------------------------------

pom.xml

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
4. **<groupId>**com.example**</groupId>**
5. **<artifactId>**spring-boot-shiro1**</artifactId>**
6. **<version>**0.0.1-SNAPSHOT**</version>**
7. **<packaging>**jar**</packaging>**
9. **<properties>**
10. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
11. **</properties>**
13. <!-- Inherit defaults from Spring Boot -->
14. **<parent>**
15. **<groupId>**org.springframework.boot**</groupId>**
16. **<artifactId>**spring-boot-starter-parent**</artifactId>**
17. **<version>**1.4.0.RELEASE**</version>**
18. **</parent>**
20. **<dependencies>**
21. <!-- spring boot web支持：mvc,aop... -->
22. **<dependency>**
23. **<groupId>**org.springframework.boot**</groupId>**
24. **<artifactId>**spring-boot-starter-web**</artifactId>**
25. **</dependency>**
26. <!-- thmleaf模板依赖. -->
27. **<dependency>**
28. **<groupId>**org.springframework.boot**</groupId>**
29. **<artifactId>**spring-boot-starter-thymeleaf**</artifactId>**
30. **</dependency>**
31. <!-- shiro spring. -->
32. **<dependency>**
33. **<groupId>**org.apache.shiro**</groupId>**
34. **<artifactId>**shiro-spring**</artifactId>**
35. **<version>**1.2.3**</version>**
36. **</dependency>**
37. <!-- Spirng data JPA依赖; -->
38. **<dependency>**
39. **<groupId>**org.springframework.boot**</groupId>**
40. **<artifactId>**spring-boot-starter-data-jpa**</artifactId>**
41. **</dependency>**
43. <!-- 热部署 -->
44. **<dependency>**
45. **<groupId>**org.springframework.boot**</groupId>**
46. **<artifactId>**spring-boot-devtools**</artifactId>**
47. **<optional>**true**</optional>**
48. **</dependency>**
50. <!-- mysql驱动; -->
51. **<dependency>**
52. **<groupId>**mysql**</groupId>**
53. **<artifactId>**mysql-connector-java**</artifactId>**
54. **</dependency>**
55. **</dependencies>**
56. **</project>**

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. **<pre** name="code" class="html" style="font-size: 14px; line-height: 28px;"**>**ShiroConfiguration.java

****[html]**** [view plain](http://blog.csdn.net/u014695188/article/details/52347372" \o "view plain" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank) [copy](http://blog.csdn.net/u014695188/article/details/52347372" \o "copy" \t "http://blog.csdn.net/liuchuanhong1/article/details/_blank)

1. package com.example.config.shiro;
3. import java.util.LinkedHashMap;
4. import java.util.Map;
6. import org.apache.shiro.authc.credential.HashedCredentialsMatcher;
7. import org.apache.shiro.mgt.SecurityManager;
8. import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
9. import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
10. import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
11. import org.springframework.context.annotation.Bean;
12. import org.springframework.context.annotation.Configuration;
14. @Configuration
15. public class ShiroConfiguration {
17. @Bean
18. public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {
19. System.out.println("ShiroConfiguration.shiroFilter()");
20. ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
22. // 必须设置SecuritManager
23. shiroFilterFactoryBean.setSecurityManager(securityManager);
25. // 拦截器
26. Map**<String**, String**>** filterChainDefinitionMap = new LinkedHashMap**<String**, String**>**();
28. // 配置退出过滤器,其中的具体代码Shiro已经替我们实现了
29. filterChainDefinitionMap.put("/logout", "logout");
31. // <!-- 过滤链定义，从上向下顺序执行，一般将 /\*\*放在最为下边 -->:这是一个坑呢，一不小心代码就不好使了;
32. // <!-- authc:所有url都必须认证通过才可以访问; anon:所有url都都可以匿名访问-->
34. filterChainDefinitionMap.put("/\*\*", "authc");
36. // 如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面
37. shiroFilterFactoryBean.setLoginUrl("/login");
38. // 登录成功后要跳转的链接
39. shiroFilterFactoryBean.setSuccessUrl("/index");
40. // 未授权界面;
41. shiroFilterFactoryBean.setUnauthorizedUrl("/403");
43. shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
44. return shiroFilterFactoryBean;
46. }
48. @Bean
49. public SecurityManager securityManager() {
50. DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
51. // 设置realm.
52. securityManager.setRealm(myShiroRealm());
53. return securityManager;
54. }
56. /\*\*
57. \* 身份认证realm; (这个需要自己写，账号密码校验；权限等)
58. \*
59. \* @return
60. \*/
61. @Bean
62. public MyShiroRealm myShiroRealm() {
63. MyShiroRealm myShiroRealm = new MyShiroRealm();
64. myShiroRealm.setCredentialsMatcher(hashedCredentialsMatcher());
65. return myShiroRealm;
66. }
67. /\*\*
68. \* 凭证匹配器
69. \* （由于我们的密码校验交给Shiro的SimpleAuthenticationInfo进行处理了
70. \*  所以我们需要修改下doGetAuthenticationInfo中的代码;
71. \* ）
72. \* @return
73. \*/
74. @Bean
75. public HashedCredentialsMatcher hashedCredentialsMatcher(){
76. HashedCredentialsMatcher hashedCredentialsMatcher = new HashedCredentialsMatcher();
78. hashedCredentialsMatcher.setHashAlgorithmName("md5");//散列算法:这里使用MD5算法;
79. hashedCredentialsMatcher.setHashIterations(2);//散列的次数，比如散列两次，相当于 md5(md5(""));
81. return hashedCredentialsMatcher;
82. }
84. /\*\*
85. \*  开启shiro aop注解支持.
86. \*  使用代理方式;所以需要开启代码支持;
87. \* @param securityManager
88. \* @return
89. \*/
90. @Bean
91. public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){
92. AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new AuthorizationAttributeSourceAdvisor();
93. authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
94. return authorizationAttributeSourceAdvisor;
95. }
97. }