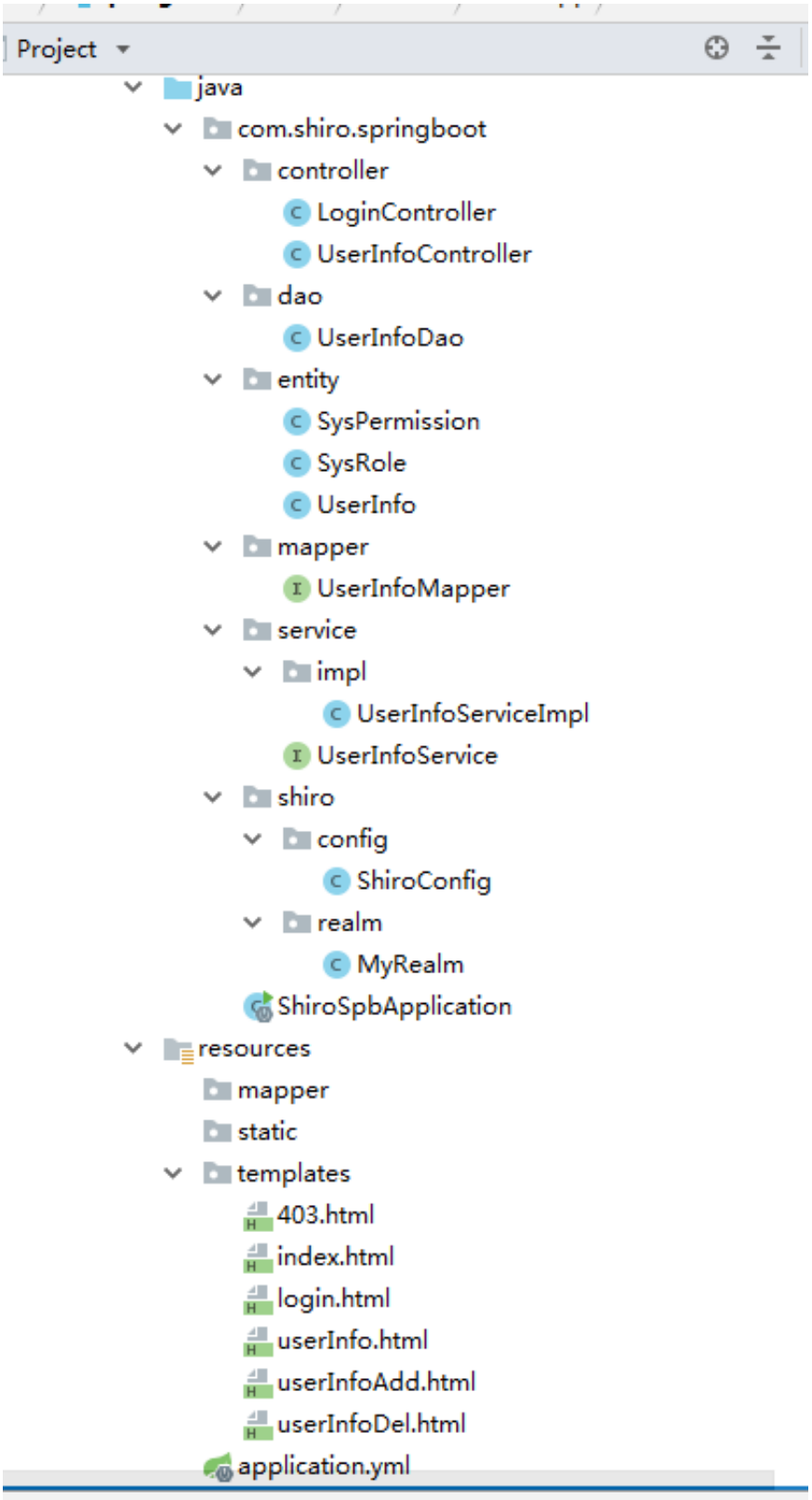
来源： https://mp.weixin.qq.com/s?__biz=MzI4NDY5Mjc1Mg==&mid=2247483909&idx=1&sn=4cbb467af285750d121d128804bc1a65&chksm=ebf6da7adc81536c992dfc7dc8332ed116d1c2e71d40fbcd720452f9a81f5f9374717d9c8771&scene=21&key=f40fb5f67815d18d3e59314c37c89b7dc03b1a0a22ff4aede667f1f5e4d1601dc19c4c2a9f40725973e1b066b1b20775e494425fb4f41352272bde2fdb803309805e0f97c08c789f971df0476d0834cb&ascene=7&uin=MjU2ODM4NDMwMw%3D%3D&devicetype=Windows+7&version=6206021b&lang=zh_CN&pass_ticket=9YDkaNuj7R3xeT4oebHktKlJamZYRD3qy9AHON2eS6N%2B9k3HhUuKMhh%2BIj%2B%2BpXfV&winzoom=1

这篇文章我们来学习如何使用Spring Boot集成Apache Shiro。安全应该是互联网公司的一道生命线，几乎任何的公司都会涉及到这方面的需求。在Java领域一般有Spring Security、Apache Shiro等安全框架，但是由于Spring Security过于庞大和复杂，大多数公司会选择Apache Shiro来使用，这篇文章会先介绍一下Apache Shiro，在结合Spring Boot给出使用案例。



1.pom.xml

这里面使用了1.5.4RELEASE版本的spring-boot，引入alibaba的druid数据库连接池，还有重点是shiro-spring包

```
<parent>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.4.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
        <shiro.version>1.3.2</shiro.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot </groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>net.sourceforge.nekohtml</groupId>
            <artifactId>nekohtml</artifactId>
            <version>1.9.22</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
```

```
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.4.0</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>6.0.6</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <optional>true</optional>
        </dependency>

        <!-- alibaba的druid数据库连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.1.0</version>
        </dependency>

    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <fork>true</fork>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

2.application.yml（springboot的配置文档）

```yaml
spring:
  datasource:
    url: jdbc:mysql://10.6.253.207:3306/test
    username: root
    password: 123
    #schema: database/import.sql
    #sql-script-encoding: utf-8
    driver-class-name: com.mysql.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource
    filters: stat
    maxActive: 20
    initialSize: 1
    maxWait: 60000
    minIdle: 1
    timeBetweenEvictionRunsMillis: 60000
    minEvictableIdleTimeMillis: 300000
    validationQuery: select 'x'
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    poolPreparedStatements: true
    maxOpenPreparedStatements: 20

  thymeleaf:
    cache: false
    mode: LEGACYHTML5

## 该配置节点为独立的节点，有很多同学容易将这个配置放在spring的节点下，导致配置无法被识别
## 因为我们使用的是mybatis 的注解，所以可以省略
#mybatis:
# mapper-locations: classpath:mapper/*.xml #注意：一定要对应mapper映射xml文件的所在路径
# type-aliases-package: com.shiro.springboot.entity # 注意：对应实体类的路径
logging:
  level: TRACE
debug: true
```

```
36
```

3.权限、角色、用户的实体类

用户类:

```
//@Entity
public class UserInfo implements Serializable {

// @Id
// @GeneratedValue
    private Integer uid;

    /**
     * 账号
     */
// @Column(unique = true)
    private String username;

    /**
     * 名称（昵称或者真实姓名，不同系统不同定义）
     */
    private String name;

    /**
     * 密码
     */
    private String password;

    /**
     * 加密的盐值
     */
    private String salt;

    /**
     * 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） 1:正常状态,2：用户被锁定.
     */
    private byte state;
```

```
33
34      /**
35       * 关联的角色（查询用户时立即获取关联的角色），一个用户多个角色
36       */
37  // @ManyToMany(fetch = FetchType.EAGER)
38  // @JoinTable(name = "SysUserRole",joinColumns =
    {@JoinColumn(name = "uid")},inverseJoinColumns =
    {@JoinColumn(name="roleId")})
39      private List<SysRole> roleList;
40
41
42      /**
43       * 密码盐. 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
44       * @return
45       */
46      public String getCredentialsSalt(){
47          return this.username+this.salt;
48      }
49  }
```

角色类：

```
1   //@Entity
2   public class SysRole {
3       // 编号
4   // @Id
5   // @GeneratedValue
6       private Integer id;
7
8       /**
9        * 角色标识程序中判断使用,如"admin",这个是唯一的:
10       */
11      private String role;
12
13      /**
14       * 角色描述,UI界面显示使用
15       */
16      private String description;
17
```

```
    /**
     * 是否可用,如果不可用将不会添加给用户
     */
    private Boolean available = Boolean.FALSE;

    /**
     * 角色 -- 权限关系：多对多关系;
     */

// @ManyToMany(fetch= FetchType.EAGER)
// @JoinTable(name="SysRolePermission",joinColumns=
{@JoinColumn(name="roleId")},inverseJoinColumns=
{@JoinColumn(name="permissionId")})
    private List<SysPermission> permissions;

    /**
     * 用户 - 角色关系定义：一个角色对应多个用户
      */
// @ManyToMany
// @JoinTable(name="SysUserRole",joinColumns=
{@JoinColumn(name="roleId")},inverseJoinColumns=
{@JoinColumn(name="uid")})
    private List<UserInfo> userInfos;
}
```

权限类：

```
public class SysPermission {

    /**
     * 主键.
     */
// @Id
// @GeneratedValue
    private Integer id;

    /**
     * 名称.
     */
    private String name;
```

```
14
15      /**
16       * 资源类型，[menu|button]
17       */
18  // @Column(columnDefinition = "enum('menu','button')")
19      private String resourceType;
20
21      /**
22       * 资源路径.
23       */
24      private String url;
25
26      /**
27       * 权限字符串,
28       * menu例子：role:*
29       * button例子：role:create,role:update,role:delete,role:view
30       */
31      private String permission;
32
33      /**
34       * 父编号
35       */
36      private Long parentId;
37
38      /**
39       * 父编号列表
40       */
41      private String parentIds;
42
43      private Boolean available = Boolean.FALSE;
44
45  // @ManyToMany
46  // @JoinTable(name = "SysRolePermission",
47  // joinColumns ={@JoinColumn(name = "permissionId")},
48  // inverseJoinColumns ={@JoinColumn(name = "roleId")})
49      private List<SysRole> roles;
50  }
```

4. mybatis的mapper层

使用注解声明成mapper，通过注解@Select设置执行的SQL

```
@Mapper
public interface UserInfoMapper /*extends
CrudRepository<UserInfo,Long>*/ {
    /**通过username查找用户信息;*/
    @Select("select * from user_info where username=#{username}")
    public UserInfo findByUsername(String username);
}
```

5.mybatis的dao层

通过@Autowired自动注入UserInfoMapper对象，并将Dao通过@Repository声明为持久层的Bean

```
@Repository
public class UserInfoDao {

    @Autowired
    public UserInfoMapper userInfoMapper;

    public UserInfo findByUsername(String username){
       return userInfoMapper.findByUsername(username);
    }

}
```

6. service层

dao层定义好了，service就容易了，自动注入userInfoDao即可，然后将service声明为bean。

```
@Service
public class UserInfoServiceImpl implements UserInfoService {
```

```
3
4       @Autowired
5       private UserInfoDao userInfoDao;
6       @Override
7       public UserInfo findByUsername(String username) {
8
 System.out.println("UserInfoServiceImpl.findByUsername()");
9           return userInfoDao.findByUsername(username);
10      }
11  }
```

7.自定义实现shiro的Realm

Realm包含了身份验证和授权的方法，必须要实现。

```
1   public class MyRealm extends AuthorizingRealm {
2
3       @Autowired
4       public UserInfoService userInfoService;
5
6       /**
7        * 授权
8        * @param principalCollection
9        * @return
10       */
11      @Override
12      protected AuthorizationInfo
    doGetAuthorizationInfo(PrincipalCollection principalCollection) {
13
14          System.out.println("权限配置--
    >MyShiroRealm.doGetAuthorizationInfo()");
15          SimpleAuthorizationInfo authorizationInfo = new
    SimpleAuthorizationInfo();
16          UserInfo userInfo = (UserInfo)
    principalCollection.getPrimaryPrincipal();
17          for (SysRole role : userInfo.getRoleList()){
18              authorizationInfo.addRole(role.getRole());
19
20              for (SysPermission p : role.getPermissions()){
```

```
21
 authorizationInfo.addStringPermission(p.getPermission());
22              }
23          }
24
25          return authorizationInfo;
26      }
27
28      /**
29       * 身份验证
30       * @param authenticationToken 包含用户名和密码
31       * @return
32       * @throws AuthenticationException
33       */
34      @Override
35      protected AuthenticationInfo
doGetAuthenticationInfo(AuthenticationToken authenticationToken)
throws AuthenticationException {
36
37
 System.out.println("MyShiroRealm.doGetAuthenticationInfo()");
38          //获取用户的输入的账号.
39          String username = (String)
authenticationToken.getPrincipal();
40          System.out.println(authenticationToken.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
46          if (userInfo == null){
47              return null;
48          }
49          //交给SimpleAuthenticationInfo进行用户密码的校验
50          SimpleAuthenticationInfo authenticationInfo = new
SimpleAuthenticationInfo(
51                  //用户名
52                  userInfo,
53                  //密码
54                  userInfo.getPassword(),
```

```
                //salt=username+salt

 ByteSource.Util.bytes(userInfo.getCredentialsSalt()),
                //realm name
                getName()
        );

        return authenticationInfo;
    }
}
```

登录认证实现

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. 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。

链接权限的实现

shiro的权限授权是通过继承 AuthorizingRealm抽象类，重载 doGetAuthorizationInfo();当访问到页面的时候，链接配置了相应的权限或者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。

当然也可以添加set集合：roles是从数据库查询的当前用户的角色，stringPermissions

是从数据库查询的当前用户对应的权限
```` authorizationInfo.setRoles(roles);
authorizationInfo.setStringPermissions(stringPermissions); ```
就是说如果在shiro配置文件中添加了 filterChainDefinitionMap.put(“/add”,“perms[权限添加]”);就说明访问/add这个链接必须要有“权限添加”这个权限才可以访问，如果在shiro配置文件中添加了 filterChainDefinitionMap.put(“/add”,“roles[100002]，perms[权限添加]”);就说明访问 /add这个链接必须要有“权限添加”这个权限和具有“100002”这个角色才可以访问。

## 8.shiro的配置类

这个类很重要，指定了Realm、SecurityManager、Filter、加密算法等。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。

```
1  package com.shiro.springboot.shiro.config;
2
3  import com.shiro.springboot.shiro.realm.MyRealm;
4  import
   org.apache.shiro.authc.credential.HashedCredentialsMatcher;
5  import
   org.apache.shiro.spring.security.interceptor.AuthorizationAttrib
   uteSourceAdvisor;
6  import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
7  import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
8  import org.springframework.context.annotation.Bean;
9  import org.springframework.context.annotation.Configuration;
10 import org.apache.shiro.mgt.SecurityManager;
11 import
   org.springframework.web.servlet.handler.SimpleMappingExceptionRe
   solver;
12
13 import java.util.LinkedHashMap;
14 import java.util.Map;
15 import java.util.Properties;
16
17 /**
18  * ShiroConfig
```
```

```
 *
 * @author linjie
 * @date 2018/8/26
 */
@Configuration
public class ShiroConfig {

    @Bean
    public ShiroFilterFactoryBean shirFilter(org.apache.shiro.mgt.SecurityManager securityManager)
    {
        System.out.println("ShiroConfiguration.shirFilter()");
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();

 shiroFilterFactoryBean.setSecurityManager(securityManager);

        //拦截器.
        Map<String, String> filterChainDefinitionMap = new LinkedHashMap<String, String>();

        // 配置不会被拦截的链接 顺序判断
        filterChainDefinitionMap.put("/static/**", "anon");

        //配置退出 过滤器,其中的具体的退出代码Shiro已经替我们实现了
        filterChainDefinitionMap.put("/logout", "logout");

        //<!-- 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;

        //<!-- 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-->
        filterChainDefinitionMap.put("/**", "authc");
        filterChainDefinitionMap.put("/", "anon");
        filterChainDefinitionMap.put("/login", "anon");
        // 默认的登录验证地址，如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面
        shiroFilterFactoryBean.setLoginUrl("/userlogin");

        // 登录成功后要跳转的链接
        shiroFilterFactoryBean.setSuccessUrl("/index");

```

```
54          //未授权界面;
55          shiroFilterFactoryBean.setUnauthorizedUrl("/403");
56
 shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainD
efinitionMap);
57
58          return shiroFilterFactoryBean;
59      }
60
61      /**
62       * 凭证匹配器
63       * （我们的密码校验交给Shiro的SimpleAuthenticationInfo进行处理
64       * ）
65       * @return
66       */
67      @Bean
68      public HashedCredentialsMatcher hashedCredentialsMatcher(){
69          HashedCredentialsMatcher hashedCredentialsMatcher = new
HashedCredentialsMatcher();
70          //散列算法:这里使用MD5算法;
71          hashedCredentialsMatcher.setHashAlgorithmName("md5");
72          //散列的次数，比如散列两次，相当于 md5(md5(""));
73          hashedCredentialsMatcher.setHashIterations(2);
74          return hashedCredentialsMatcher;
75      }
76
77      @Bean
78      public MyRealm myShiroRealm(){
79          MyRealm myShiroRealm = new MyRealm();
80          return myShiroRealm;
81      }
82
83      @Bean
84      public org.apache.shiro.mgt.SecurityManager
securityManager(){
85          DefaultWebSecurityManager securityManager = new
DefaultWebSecurityManager();
86          securityManager.setRealm(myShiroRealm());
87          return securityManager;
88      }
89
90      /**
```

```
     * 开启shiro aop注解支持.
     * 使用代理方式;所以需要开启代码支持;
     * @param securityManager
     * @return
     */
    @Bean
    public AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor(SecurityManager
securityManager){
        AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor = new
AuthorizationAttributeSourceAdvisor();
```

Filter Chain定义说明：

1. 一个URL可以配置多个Filter，使用逗号分隔
2. 当设置多个过滤器时，全部验证通过，才视为通过
3. 部分过滤器可指定参数，如perms，roles

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

# 9.login页面

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
<body>
错误信息：<h4 th:text="${msg}"></h4>
<form action="/userlogin" method="post">
    <p>账号：<input type="text" name="username" value="admin"/>
</p>
    <p>密码：<input type="text" name="password" value="123456"/>
</p>
    <p><input type="submit" value="登录"/></p>
```

```
</form>
</body>
</html>
```

## 测试

1、编写好后就可以启动程序，访问index页面，由于没有登录就会跳转到login页面。登录之后就会跳转到index页面，登录后，有直接在浏览器中输入index页面访问，又会跳转到login页面。上面这些操作时候触发 `MyShiroRealm.doGetAuthenticationInfo()` 这个方法，也就是登录认证的方法。

2、登录admin账户，访问： `http://127.0.0.1:8080/userInfo/userAdd` 显示 `用户添加界面` ，访问 `http://127.0.0.1:8080/userInfo/userDel` 显示 `403 没有权限` 。上面这些操作时候触发 `MyShiroRealm.doGetAuthorizationInfo()` 这个方面，也就是权限校验的方法。

3、修改admin不同的权限进行测试

shiro很强大，这仅仅是完成了登录认证和权限管理这两个功能，更多内容以后有时间再做探讨。

示例代码：https://github.com/ityouknow/spring-boot-examples