springboot整合shiro –快速入门(二)

使用springboot+mybatis+druid+thymleaf模板实现快速入门(关于整合mybatis和druid数据源,参考之前的博客,这里不做介绍)，博客部分内容参考以下博客：
https://blog.csdn.net/ityouknow/article/details/73836159
http://www.cnblogs.com/strinkbug/p/6139393.html
https://www.cnblogs.com/kibana/p/8953566.html

快速上手

配置信息

pom添加依赖

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-spring</artifactId>
    <version>1.4.0</version>
</dependency>
<!-- shiro-thymeleaf 2.0.0-->
<dependency>
    <groupId>com.github.theborakompanioni</groupId>
    <artifactId>thymeleaf-extras-shiro</artifactId>
    <version>2.0.0</version>
</dependency>
```

配置文件

```
#配置tomcat
server.port=9090
server.servlet-path=/

#关闭默认模板引擎缓存
spring.thymeleaf.cache=false

#配置日志文件
logging.config=classpath:config/logback-spring.xml

#配置jdbc数据源
jdbc.ds.driverClassName=com.mysql.jdbc.Driver
jdbc.ds.url=jdbc:mysql://127.0.0.1:3306/testshiro?useUnicode=true&characterEncoding=UTF-8
jdbc.ds.username=root
jdbc.ds.password=123456

#mybatis配置
mybatis.mapperLocations=classpath*:mapper/**/*.xml
```

数据库建表

数据库创建以下几个表(用户表，角色表，用户-角色表，权限表，角色-权限表),并创建对应的实体类

user_info.sql(用户表)

```
DROP TABLE IF EXISTS `user_info`;
CREATE TABLE `user_info` (
  `uid` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(50) DEFAULT '' COMMENT '用户名',
  `password` varchar(256) DEFAULT NULL COMMENT '登录密码',
  `name` varchar(256) DEFAULT NULL COMMENT '用户真实姓名',
  `id_card_num` varchar(256) DEFAULT NULL COMMENT '用户身份证号',
  `state` char(1) DEFAULT '0' COMMENT '用户状态: 0:正常状态,1: 用户被锁定',
  PRIMARY KEY (`uid`),
  UNIQUE KEY `username` (`username`) USING BTREE,
  UNIQUE KEY `id_card_num` (`id_card_num`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

sys_role.sql(角色表)

```
DROP TABLE IF EXISTS `sys_role`;
CREATE TABLE `sys_role` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `available` char(1) DEFAULT '0' COMMENT '是否可用0可用  1不可用',
  `role` varchar(20) DEFAULT NULL COMMENT '角色标识程序中判断使用,如"admin"',
  `description` varchar(100) DEFAULT NULL COMMENT '角色描述,UI界面显示使用',
```

```
  PRIMARY KEY (`id`),
  UNIQUE KEY `role` (`role`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

sys_user_role.sql(用户-角色表)

```
DROP TABLE IF EXISTS `sys_user_role`;
CREATE TABLE `sys_user_role` (
  `uid` int(11) DEFAULT NULL COMMENT '用户id',
  `role_id` int(11) DEFAULT NULL COMMENT '角色id',
  KEY `uid` (`uid`) USING BTREE,
  KEY `role_id` (`role_id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

sys_permission.sql(权限表)

```
DROP TABLE IF EXISTS `sys_permission`;
CREATE TABLE `sys_permission` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `parent_id` int(11) DEFAULT NULL COMMENT '父编号,本权限可能是该父编号权限的子权限',
  `parent_ids` varchar(20) DEFAULT NULL COMMENT '父编号列表',
  `permission` varchar(100) DEFAULT NULL COMMENT '权限字符串,menu例子：role:*，button例子：role:create,role:update,role:delete,role:view',
  `resource_type` varchar(20) DEFAULT NULL COMMENT '资源类型，[menu|button]',
  `url` varchar(200) DEFAULT NULL COMMENT '资源路径 如：/userinfo/list',
  `name` varchar(50) DEFAULT NULL COMMENT '权限名称',
  `available` char(1) DEFAULT '0' COMMENT '是否可用0可用  1不可用',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

sys_role_permission.sql(角色-权限表)

```
DROP TABLE IF EXISTS `sys_role_permission`;
CREATE TABLE `sys_role_permission` (
  `role_id` int(11) DEFAULT NULL COMMENT '角色id',
  `permission_id` int(11) DEFAULT NULL COMMENT '权限id',
  KEY `role_id` (`role_id`) USING BTREE,
  KEY `permission_id` (`permission_id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

在之前的表中添加数据

```
#插入用户信息表
INSERT INTO user_info(uid,username,`password`,`name`,id_card_num) VALUES (null,'admin','123456','超哥','133333333333333333');
INSERT INTO user_info(uid,username,`password`,`name`,id_card_num) VALUES (null,'test','123456','孙悟空','155555555555555555');
#插入用户角色表
INSERT INTO `sys_role` (`id`,`available`,`description`,`role`) VALUES (null,0,'管理员','admin');
INSERT INTO `sys_role` (`id`,`available`,`description`,`role`) VALUES (null,0,'VIP会员','vip');
INSERT INTO `sys_role` (`id`,`available`,`description`,`role`) VALUES (null,1,'测试','test');
#插入用户_角色关联表
INSERT INTO `sys_user_role` (`role_id`,`uid`) VALUES (1,1);
INSERT INTO `sys_user_role` (`role_id`,`uid`) VALUES (2,2);
#插入权限表
INSERT INTO `sys_permission` (`id`,`available`,`name`,`parent_id`,`parent_ids`,`permission`,`resource_type`,`url`) VALUES (null,0,'用户
INSERT INTO `sys_permission` (`id`,`available`,`name`,`parent_id`,`parent_ids`,`permission`,`resource_type`,`url`) VALUES (null,0,'用户
INSERT INTO `sys_permission` (`id`,`available`,`name`,`parent_id`,`parent_ids`,`permission`,`resource_type`,`url`) VALUES (null,0,'用户
#插入角色_权限表
INSERT INTO `sys_role_permission` (`permission_id`,`role_id`) VALUES (1,1);
INSERT INTO `sys_role_permission` (`permission_id`,`role_id`) VALUES (2,1);
INSERT INTO `sys_role_permission` (`permission_id`,`role_id`) VALUES (3,2);
```

根据上面的sql创建实体类

用户信息

```
public class User {
    private Integer uid;
    private String username;
    private String password;
    private String name;
    private String id_card_num;
    private String state;
    private Set<Role> roles = new HashSet<>();
}
```

角色信息

```
public class Role {
    private Integer id;
    private String role;
    private String description;
    private String available;
    private Set<User> users = new HashSet<>();
    private Set<Permission> permissions = new HashSet<>();
}
```

权限信息

```
public class Permission {
    private Integer id;
    private Integer parent_id;
    private String parent_ids;
    private String permission;
    private String resource_type;
    private String url;
    private String name;
    private String available;
    private Set<Role> roles = new HashSet<>();
}
```

编写mapper

UserMapper.java

```
@Mapper
public interface UserMapper {
    User findByUserName(String userName);
    int insert(User user);
    int del(@Param("username") String username);
}
```

RoleMapper.java

```
@Mapper
public interface RoleMapper {
    Set<Role> findRolesByUserId(@Param("uid") Integer uid);
}
```

PermissionMapper.java

```
@Mapper
public interface PermissionMapper {
    Set<Permission> findPermissionsByRoleId(@Param("roles") Set<Role> roles);
}
```

编写Mapper.xml

UserMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.springboot.test.shiro.modules.user.dao.UserMapper">

    <!-- 查询用户信息 -->
    <select id="findByUserName" resultType="com.springboot.test.shiro.modules.user.dao.entity.User">
        SELECT * FROM user_info WHERE username = #{userName}
    </select>

    <!-- 添加用户 -->
    <!-- 创建用户 -->
    <insert id="insert" parameterType="com.springboot.test.shiro.modules.user.dao.entity.User">
        <selectKey resultType="java.lang.Integer" keyProperty="uid" order="AFTER">
            SELECT
            LAST_INSERT_ID()
        </selectKey>
        insert into user_info
        <trim prefix="(" suffix=")" suffixOverrides="," >
            <if test="uid != null" >
                uid,
            </if>
            <if test="username != null and username != ''" >
                username,
            </if>
```

```
            <if test="password != null and password != ''" >
                password,
            </if>
            <if test="name != null and name != ''" >
                `name`,
            </if>
            <if test="id_card_num != null and id_card_num != ''" >
                id_card_num,
            </if>
            <if test="state != null and state != ''" >
                state,
            </if>
        </trim>
        <trim prefix="values (" suffix=")" suffixOverrides="," >
            <if test="uid != null" >
                #{uid},
            </if>
            <if test="username != null and username != ''" >
                #{username},
            </if>
            <if test="password != null and password != ''" >
                #{password},
            </if>
            <if test="name != null and name != ''" >
                #{name},
            </if>
            <if test="id_card_num != null and id_card_num != ''" >
                #{id_card_num},
            </if>
            <if test="state != null and state != ''" >
                #{state},
            </if>
        </trim>
    </insert>

    <!-- 删除用户 -->
    <delete id="del">
        DELETE FROM user_info WHERE username = #{username}
    </delete>

</mapper>
```

RoleMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.springboot.test.shiro.modules.user.dao.RoleMapper">

    <!-- 查询用户信息 -->
    <select id="findRolesByUserId" resultType="com.springboot.test.shiro.modules.user.dao.entity.Role">
        SELECT r.* from sys_role r LEFT JOIN sys_user_role ur on r.id = ur.role_id where ur.uid  = #{uid}
    </select>

</mapper>
```

PermissionMapper.java

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.springboot.test.shiro.modules.user.dao.PermissionMapper">

    <!-- 查询用户权限信息 -->
    <select id="findPermissionsByRoleId" resultType="com.springboot.test.shiro.modules.user.dao.entity.Permission">
        SELECT p.* from sys_permission p LEFT JOIN sys_role_permission rp on p.id = rp.permission_id WHERE rp.role_id IN
        <foreach collection="roles" index="index" item="item" open="(" close=")" separator=",">
            #{item.id}
        </foreach>
    </select>

</mapper>
```

Shiro 配置

创建ShiroConfig.java配置类

我们需要定义一系列关于URL的规则和访问权限。

```java
package com.springboot.test.shiro.config;

import at.pollux.thymeleaf.shiro.dialect.ShiroDialect;
import com.springboot.test.shiro.config.shiro.ShiroRealm;
import org.apache.shiro.codec.Base64;
import org.apache.shiro.mgt.RememberMeManager;
import org.apache.shiro.spring.LifecycleBeanPostProcessor;
import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.web.mgt.CookieRememberMeManager;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.handler.SimpleMappingExceptionResolver;

import javax.servlet.Filter;
import java.util.LinkedHashMap;
import java.util.Properties;

/**
 * @author: wangsaichao
 * @date: 2018/5/10
 * @description: Shiro配置
 */
@Configuration
public class ShiroConfig {


    /**
     * ShiroFilterFactoryBean 处理拦截资源文件问题。
     * 注意：初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager
     * Web应用中,Shiro可控制的Web请求必须经过Shiro主过滤器的拦截
     * @param securityManager
     * @return
     */
    @Bean(name = "shirFilter")
    public ShiroFilterFactoryBean shiroFilter(@Qualifier("securityManager") SecurityManager securityManager) {

        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();

        //必须设置 SecurityManager,Shiro的核心安全接口
        shiroFilterFactoryBean.setSecurityManager(securityManager);
        //这里的/login是后台的接口名,非页面，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("/login");
        //这里的/index是后台的接口名,非页面,登录成功后要跳转的链接
        shiroFilterFactoryBean.setSuccessUrl("/index");
        //未授权界面,该配置无效，并不会进行页面跳转
        shiroFilterFactoryBean.setUnauthorizedUrl("/unauthorized");

        //自定义拦截器限制并发人数,参考博客：
        //LinkedHashMap<String, Filter> filtersMap = new LinkedHashMap<>();
        //限制同一帐号同时在线的个数
        //filtersMap.put("kickout", kickoutSessionControlFilter());
        //shiroFilterFactoryBean.setFilters(filtersMap);

        // 配置访问权限 必须是LinkedHashMap，因为它必须保证有序
        // 过滤链定义，从上向下顺序执行，一般将 /**放在最为下边 一定要注意顺序,否则就不好使了
        LinkedHashMap<String, String> filterChainDefinitionMap = new LinkedHashMap<>();
        //配置不登录可以访问的资源，anon 表示资源都可以匿名访问
        filterChainDefinitionMap.put("/login", "anon");
        filterChainDefinitionMap.put("/", "anon");
        filterChainDefinitionMap.put("/css/**", "anon");
        filterChainDefinitionMap.put("/js/**", "anon");
        filterChainDefinitionMap.put("/img/**", "anon");
        filterChainDefinitionMap.put("/druid/**", "anon");
        //logout是shiro提供的过滤器
        filterChainDefinitionMap.put("/logout", "logout");
        //此时访问/userInfo/del需要del权限,在自定义Realm中为用户授权。
        //filterChainDefinitionMap.put("/userInfo/del", "perms[\"userInfo:del\"]");

        //其他资源都需要认证  authc 表示需要认证才能进行访问
        filterChainDefinitionMap.put("/**", "authc");

        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);

        return shiroFilterFactoryBean;
    }
```

```
80
81      /**
82       * 配置核心安全事务管理器
83       * @param shiroRealm
84       * @return
85       */
86      @Bean(name="securityManager")
87      public SecurityManager securityManager(@Qualifier("shiroRealm") ShiroRealm shiroRealm) {
88          DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
89          //设置自定义realm.
90          securityManager.setRealm(shiroRealm);
91          //配置记住我 参考博客:
92          //securityManager.setRememberMeManager(rememberMeManager());
93
94          //配置 redis缓存管理器 参考博客:
95          //securityManager.setCacheManager(getEhCacheManager());
96
97          //配置自定义session管理，使用redis 参考博客:
98          //securityManager.setSessionManager(sessionManager());
99
100         return securityManager;
101     }
102
103     /**
104      * 配置Shiro生命周期处理器
105      * @return
106      */
107     @Bean(name = "lifecycleBeanPostProcessor")
108     public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {
109         return new LifecycleBeanPostProcessor();
110     }
111
112     /**
113      *  身份认证realm；(这个需要自己写，账号密码校验；权限等)
114      * @return
115      */
116     @Bean
117     public ShiroRealm shiroRealm(){
118         ShiroRealm shiroRealm = new ShiroRealm();
119         return shiroRealm;
120     }
121
122     /**
123      * 必须（thymeleaf页面使用shiro标签控制按钮是否显示）
124      * 未引入thymeleaf包，Caused by: java.lang.ClassNotFoundException: org.thymeleaf.dialect.AbstractProcessorDialect
125      * @return
126      */
127     @Bean
128     public ShiroDialect shiroDialect() {
129         return new ShiroDialect();
130     }
131
132
133 }
```

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。

ShiroRealm.java

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.

```
package com.springboot.test.shiro.config.shiro;

import com.springboot.test.shiro.modules.user.dao.PermissionMapper;
import com.springboot.test.shiro.modules.user.dao.RoleMapper;
import com.springboot.test.shiro.modules.user.dao.entity.Permission;
import com.springboot.test.shiro.modules.user.dao.entity.Role;
import com.springboot.test.shiro.modules.user.dao.UserMapper;
import com.springboot.test.shiro.modules.user.dao.entity.User;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.*;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.Set;

/**
 * @author: wangsaichao
 * @date: 2018/5/10
 * @description: 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
 * 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.
 */
public class ShiroRealm extends AuthorizingRealm {

    @Autowired
    private UserMapper userMapper;

    @Autowired
    private RoleMapper roleMapper;

    @Autowired
    private PermissionMapper permissionMapper;

    /**
     * 验证用户身份
     * @param authenticationToken
     * @return
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

        //获取用户名密码 第一种方式
        //String username = (String) authenticationToken.getPrincipal();
        //String password = new String((char[]) authenticationToken.getCredentials());

        //获取用户名 密码 第二种方式
        UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) authenticationToken;
        String username = usernamePasswordToken.getUsername();
        String password = new String(usernamePasswordToken.getPassword());

        //从数据库查询用户信息
        User user = this.userMapper.findByUserName(username);

        //可以在这里直接对用户名校验,或者调用 CredentialsMatcher 校验
        if (user == null) {
            throw new UnknownAccountException("用户名或密码错误！");
        }
        if (!password.equals(user.getPassword())) {
            throw new IncorrectCredentialsException("用户名或密码错误！");
        }
        if ("1".equals(user.getState())) {
            throw new LockedAccountException("账号已被锁定,请联系管理员！");
        }

        //调用 CredentialsMatcher 校验 还需要创建一个类 继承CredentialsMatcher  如果在上面校验了,这个就不需要了
        //配置自定义权限登录器 参考博客:

        SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user,user.getPassword(), getName());
```

```
        return info;
    }

    /**
     * 授权用户权限
     * 授权的方法是在碰到<shiro:hasPermission name=''></shiro:hasPermission>标签的时候调用的
     * 它会去检测shiro框架中的权限(这里的permissions)是否包含有该标签的name值,如果有,里面的内容显示
     * 如果没有,里面的内容不予显示(这就完成了对于权限的认证.)
     *
     * 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。
     * authorizationInfo.addRole(role.getRole()); authorizationInfo.addStringPermission(p.getPermission());
     *
     * 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
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

        //获取用户
        User user = (User) SecurityUtils.getSubject().getPrincipal();

        //获取用户角色
        Set<Role> roles =this.roleMapper.findRolesByUserId(user.getUid());
        //添加角色
        SimpleAuthorizationInfo authorizationInfo =  new SimpleAuthorizationInfo();
        for (Role role : roles) {
            authorizationInfo.addRole(role.getRole());
        }

        //获取用户权限
        Set<Permission> permissions = this.permissionMapper.findPermissionsByRoleId(roles);
        //添加权限
        for (Permission permission:permissions) {
            authorizationInfo.addStringPermission(permission.getPermission());
        }

        return authorizationInfo;
    }

}
```

编写登录Controller

LoginController.java

```
package com.springboot.test.shiro.modules.login;

import com.springboot.test.shiro.modules.user.dao.entity.User;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.subject.Subject;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

/**
 * @author: wangsaichao
 * @date: 2018/5/11
 * @description:
 */
@Controller
public class LoginController {
```

```
22
23      /**
24       * 访问项目根路径
25       * @return
26       */
27      @RequestMapping(value = "/",method = RequestMethod.GET)
28      public String root(Model model) {
29          Subject subject = SecurityUtils.getSubject();
30          User user=(User) subject.getPrincipal();
31          if (user == null){
32              return "redirect:/login";
33          }else{
34              return "redirect:/index";
35          }
36
37      }
38
39
40      /**
41       * 跳转到login页面
42       * @return
43       */
44      @RequestMapping(value = "/login",method = RequestMethod.GET)
45      public String login(Model model) {
46          Subject subject = SecurityUtils.getSubject();
47          User user=(User) subject.getPrincipal();
48          if (user == null){
49              return "login";
50          }else{
51              return "redirect:index";
52          }
53
54      }
55
56      /**
57       * 用户登录
58       * @param request
59       * @param username
60       * @param password
61       * @param model
62       * @param session
63       * @return
64       */
65      @RequestMapping(value = "/login",method = RequestMethod.POST)
66      public String loginUser(HttpServletRequest request, String username, String password, Model model, HttpSession session) {
67
68          //对密码进行加密
69          //password=new SimpleHash("md5", password, ByteSource.Util.bytes(username.toLowerCase() + "shiro"),2).toHex();
70          //如果有点击  记住我
71          //UsernamePasswordToken usernamePasswordToken=new UsernamePasswordToken(username,password,remeberMe);
72          UsernamePasswordToken usernamePasswordToken = new UsernamePasswordToken(username,password);
73          Subject subject = SecurityUtils.getSubject();
74          try {
75              //登录操作
76              subject.login(usernamePasswordToken);
77              User user=(User) subject.getPrincipal();
78              //更新用户登录时间，也可以在ShiroRealm里面做
79              session.setAttribute("user", user);
80              model.addAttribute("user",user);
81              return "index";
82          } catch(Exception e) {
83              //登录失败从request中获取shiro处理的异常信息 shiroLoginFailure:就是shiro异常类的全类名
84              String exception = (String) request.getAttribute("shiroLoginFailure");
85              model.addAttribute("msg",e.getMessage());
86              //返回登录页面
87              return "login";
88          }
89      }
90
91      @RequestMapping("/index")
92      public String index(HttpSession session, Model model) {
93          Subject subject = SecurityUtils.getSubject();
94          User user=(User) subject.getPrincipal();
95          if (user == null){
96              return "login";
97          }else{
98              model.addAttribute("user",user);
99              return "index";
100         }
```

```
    }

    /**
     * 登出  这个方法没用到,用的是shiro默认的logout
     * @param session
     * @param model
     * @return
     */
    @RequestMapping("/logout")
    public String logout(HttpSession session, Model model) {
        Subject subject = SecurityUtils.getSubject();
        subject.logout();
        model.addAttribute("msg","安全退出！");
        return "login";
    }

    /**
     * 跳转到无权限页面
     * @param session
     * @param model
     * @return
     */
    @RequestMapping("/unauthorized")
    public String unauthorized(HttpSession session, Model model) {
        return "unauthorized";
    }


}
```

UserController.java

```
package com.springboot.test.shiro.modules.login;

import com.springboot.test.shiro.modules.user.dao.entity.User;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.subject.Subject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

/**
 * @author: wangsaichao
 * @date: 2018/5/12
 * @description:
 */
@RestController
@RequestMapping("userInfo")
public class UserController {

    @Autowired
    private UserService userService;

    /**
     * 创建固定写死的用户
     * @param model
     * @return
     */
    @RequestMapping(value = "/add",method = RequestMethod.GET)
    @ResponseBody
    public String login(Model model) {

        User user = new User();
        user.setName("王赛超");
        user.setId_card_num("17777777777777777");
        user.setUsername("wangsaichao");

        userService.insert(user);

        return "创建用户成功";

    }

    /**
     * 删除固定写死的用户
```

```
     * @param model
     * @return
     */
    @RequestMapping(value = "/del",method = RequestMethod.GET)
    @ResponseBody
    public String del(Model model) {

        userService.del("wangsaichao");

        return "删除用户名为wangsaichao用户成功";

    }

    @RequestMapping(value = "/view",method = RequestMethod.GET)
    @ResponseBody
    public String view(Model model) {

        return "这是用户列表页";

    }


}
```

新建页面

login.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
      xmlns:shiro="http://www.pollix.at/thymeleaf/shiro">
<head>
    <meta charset="UTF-8" />
    <title>Insert title here</title>
</head>
<body>
<h1>欢迎登录</h1>
<h1 th:if="${msg != null }" th:text="${msg}" style="color: red"></h1>
<form action="/login" method="post">
    用户名: <input type="text" name="username"/><br/>
    密码: <input type="password" name="password"/><br/>
    <input type="submit" value="提交"/>
</form>
</body>
</html>
```

index.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
      xmlns:shiro="http://www.pollix.at/thymeleaf/shiro">
<head>
    <meta charset="UTF-8" />
    <title>Insert title here</title>
</head>
<body>
<h1 th:text="'欢迎' + ${user.username } + '光临!请选择你的操作'"></h1><br/>
<ul>
    <h1 th:if="${msg != null }" th:text="${msg}" style="color: red"></h1>

    <shiro:hasPermission name="userInfo:add"><a href="/userInfo/add">点击添加固定用户信息(后台写死,方便测试)</a></shiro:hasPermission><br/>
    <shiro:hasPermission name="userInfo:del"><a href="/userInfo/del">点击删除固定用户信息(后台写死,方便测试)</a></shiro:hasPermission><br/>
    <shiro:hasPermission name="userInfo:view"><a href="/userInfo/view">显示此内容表示拥有查看用户列表的权限</a></shiro:hasPermission><br/>



    <!-- 用户没有身份验证时显示相应信息，即游客访问信息 -->
    <shiro:guest>游客显示的信息</shiro:guest><br/>
    <!-- 用户已经身份验证/记住我登录后显示相应的信息 -->
    <shiro:user>用户已经登录过了</shiro:user><br/>
    <!-- 用户已经身份验证通过，即Subject.login登录成功，不是记住我登录的 -->
    <shiro:authenticated>不是记住我登录</shiro:authenticated><br/>
    <!-- 显示用户身份信息，通常为登录帐号信息，默认调用Subject.getPrincipal()获取，即Primary Principal -->
    <shiro:principal></shiro:principal><br/>
    <!--用户已经身份验证通过，即没有调用Subject.login进行登录，包括记住我自动登录的也属于未进行身份验证，与guest标签的区别是，该标签包含已记住用户 -->
    <shiro:notAuthenticated>已记住用户</shiro:notAuthenticated><br/>
```

```
    <!-- 相当于Subject.getPrincipals().oneByType(String.class) -->
    <shiro:principal type="java.lang.String"/><br/>
    <!-- 相当于((User)Subject.getPrincipals()).getUsername() -->
    <shiro:principal property="username"/><br/>
    <!-- 如果当前Subject有角色将显示body体内容 name="角色名" -->
    <shiro:hasRole name="admin">这是admin角色</shiro:hasRole><br/>
    <!-- 如果当前Subject有任意一个角色（或的关系）将显示body体内容。 name="角色名1,角色名2..." -->
    <shiro:hasAnyRoles name="admin,vip">用户拥有admin角色 或者 vip角色</shiro:hasAnyRoles><br/>
    <!-- 如果当前Subject没有角色将显示body体内容 -->
    <shiro:lacksRole name="admin">如果不是admin角色,显示内容</shiro:lacksRole><br/>
    <!-- 如果当前Subject有权限将显示body体内容 name="权限名" -->
    <shiro:hasPermission name="userInfo:add">用户拥有添加权限</shiro:hasPermission><br/>
    <!-- 用户同时拥有以下两种权限,显示内容 -->
    <shiro:hasAllPermissions name="userInfo:add,userInfo:view">用户同时拥有列表权限和添加权限</shiro:hasAllPermissions><br/>
    <!-- 用户拥有以下权限任意一种 -->
    <shiro:hasAnyPermissions name="userInfo:view,userInfo:del">用户拥有列表权限或者删除权限</shiro:hasAnyPermissions><br/>
    <!-- 如果当前Subject没有权限将显示body体内容 name="权限名" -->
    <shiro:lacksPermission name="userInfo:add">如果用户没有添加权限，显示的内容</shiro:lacksPermission><br/>
</ul>
<a href="/logout">点我注销</a>
</body>
</html>
```

unauthorized.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
      xmlns:shiro="http://www.pollix.at/thymeleaf/shiro">
<head>
    <meta charset="UTF-8" />
    <title>Insert title here</title>
</head>
<body>
<h1>对不起，您没有权限</h1>
</body>
</html>
```

进行身份验证测试

第一步: 访问http://localhost:9090/userInfo/add 发现自动跳转到登录页
第二步： 使用admin登录

欢迎admin光临!请选择你的操作

点击添加固定用户信息(后台写死,方便测试)

显示此内容表示拥有查看用户列表的权限

用户已经登录过了
不是记住我登录
User{uid=1, username='admin', password='123456', name='超哥', id_card_num='133333333333333333', state='0', roles=[]}

admin
这是admin角色
用户拥有admin角色 或者 vip角色

用户拥有添加权限
用户同时拥有列表权限和添加权限
用户拥有列表权限或者删除权限

点我注销

第三步： 注销之后使用test登录

欢迎test光临!请选择你的操作

点击删除固定用户信息(后台写死,方便测试)

用户已经登录过了
不是记住我登录
User{uid=2, username='test', password='123456', name='孙悟空', id_card_num='155555555555555555', state='0', roles=[]}

test

用户拥有admin角色 或者 vip角色
如果不是admin角色,显示内容

用户拥有列表权限或者删除权限
如果用户没有添加权限，显示的内容

点我注销

不同的用户登录,显示不同的功能,点击之后也可以调用后台服务,证明身份验证成功。

权限功能校验

经过上面的过程,已经可以对用户的身份进行校验,但是这个时候， 但是权限控制好像没有什么作用， 因为我们使用admin用户登录之后,在浏览器上访问地址 /userInfo/del发现也是可以使用的,其实我们还少了以下步骤，也就是开启注解支持

第一：在ShiroConfig中配置以下bean

```
/**
 * 开启shiro 注解模式
 * 可以在controller中的方法前加上注解
 * 如 @RequiresPermissions("userInfo:add")
 * @param securityManager
 * @return
 */
@Bean
public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(@Qualifier("securityManager") SecurityManager securityM
    AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new AuthorizationAttributeSourceAdvisor();
    authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
    return authorizationAttributeSourceAdvisor;
}
```

第二：在UserController的方法中,添加对应权限,如下：

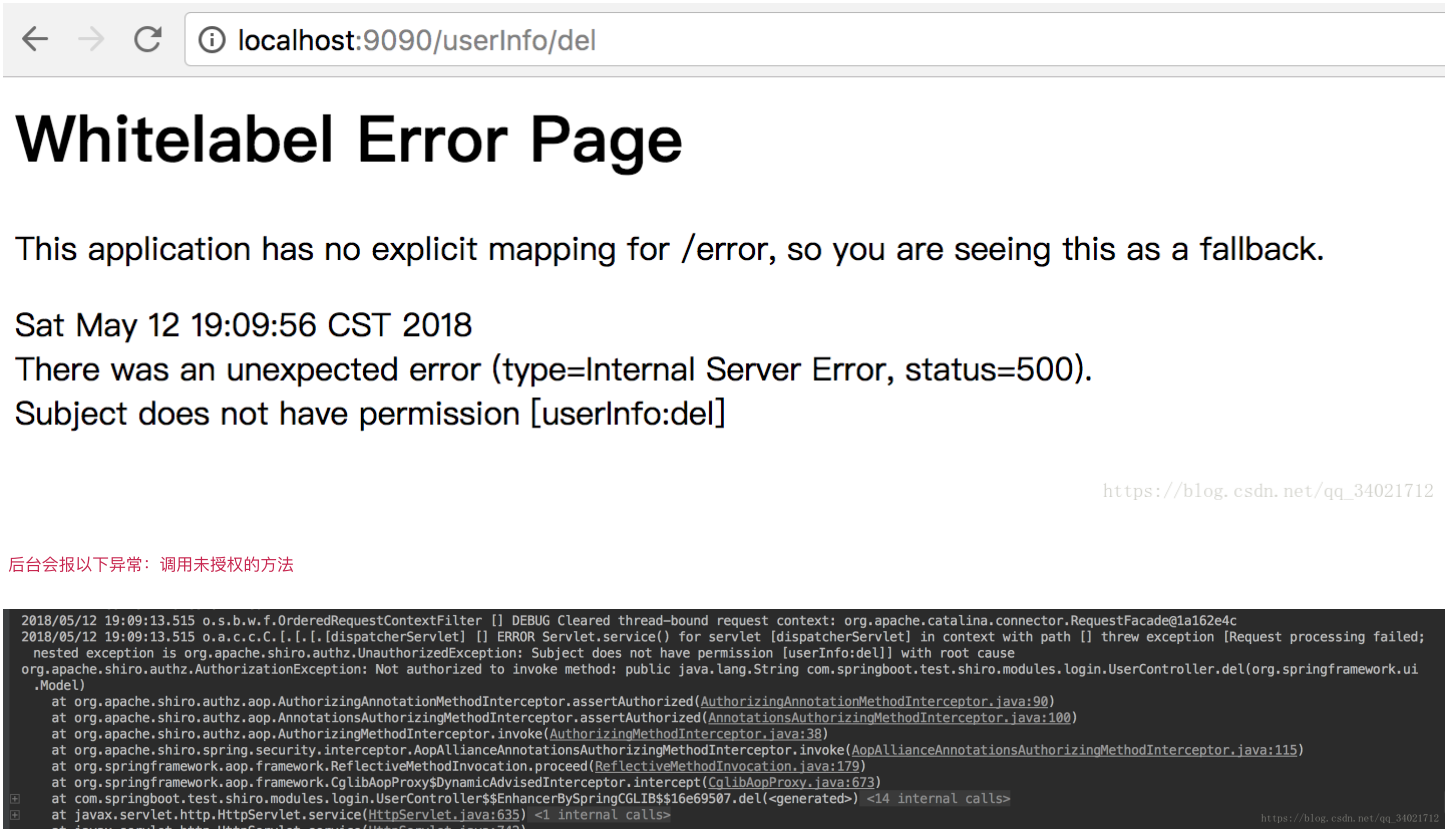
```
@RequiresPermissions("userInfo:del")
@RequestMapping(value = "/del",method = RequestMethod.GET)
@ResponseBody
public String del(Model model) {

    userService.del("wangsaichao");

    return "删除用户名为wangsaichao用户成功";

}
```

添加 `@RequiresPermissions("userInfo:del")` 然后重启项目,再次使用amdin登录之后,在浏览器上调用http://localhost:9090/userInfo/del就会跳转到以下错误页。证明权限校验成功。



后台会报以下异常：调用未授权的方法

```
2018/05/12 19:09:13.515 o.s.b.w.f.OrderedRequestContextFilter [] DEBUG Cleared thread-bound request context: org.apache.catalina.connector.RequestFacade@1a162e4c
2018/05/12 19:09:13.515 o.a.c.c.C.[.[.[.[dispatcherServlet] [] ERROR Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed;
  nested exception is org.apache.shiro.authz.UnauthorizedException: Subject does not have permission [userInfo:del]] with root cause
org.apache.shiro.authz.AuthorizationException: Not authorized to invoke method: public java.lang.String com.springboot.test.shiro.modules.login.UserController.del(org.springframework.ui
  .Model)
    at org.apache.shiro.authz.aop.AuthorizingAnnotationMethodInterceptor.assertAuthorized(AuthorizingAnnotationMethodInterceptor.java:90)
    at org.apache.shiro.authz.aop.AnnotationsAuthorizingMethodInterceptor.assertAuthorized(AnnotationsAuthorizingMethodInterceptor.java:100)
    at org.apache.shiro.authz.aop.AuthorizingMethodInterceptor.invoke(AuthorizingMethodInterceptor.java:38)
    at org.apache.shiro.spring.security.interceptor.AopAllianceAnnotationsAuthorizingMethodInterceptor.invoke(AopAllianceAnnotationsAuthorizingMethodInterceptor.java:115)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:179)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:673)
    at com.springboot.test.shiro.modules.login.UserController$$EnhancerBySpringCGLIB$$16e69507.del(<generated>) <14 internal calls>
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:635) <1 internal calls>
```

到此,shiro入门完了,我相信很多人对shiro 已经可以说了解怎么用了,其实还有很多问题:

1.首先是错误页显示,没有权限理论应该跳转到我们配置的无权限的页面,但是并没有

2.我们不断的访问http://localhost:9090/userInfo/view 发现每次都会去数据库查询权限,但是实际中我们的权限信息是不怎么会改变的，所以我们希望是第一次访问，然后进行缓存处理等等,这些会在后面的文章中。