# 集成Spring Security进行权限控制

1：掌握Spring Security实现认证的过程

2：掌握Spring Security实现授权的过程

# ****第一章****. 在项目中使用Spring Security

前面我们已经学习了Spring Security框架的使用方法，本章节我们就需要将Spring Security框架应用到后台系统中进行权限控制，其本质就是认证和授权。

要进行认证和授权需要前面课程中提到的权限模型涉及的7张表支撑，因为用户信息、权限信息、菜单信息、角色信息、关联信息等都保存在这7张表中，也就是这些表中的数据是我们进行认证和授权的依据。所以在真正进行认证和授权之前需要对这些数据进行管理，即我们需要开发如下一些功能：

1、用户数据管理（增删改查、用户关联角色）

2、角色数据管理（增删改查、角色关联权限、角色关联菜单）

3、权限数据管理（增删改查）

4、菜单数据管理（增删改查）

鉴于时间关系，我们不再实现这些数据管理的代码开发。我们可以直接将数据导入到数据库中即可。

**导入用户、角色、权限、菜单 的初始数据**

**-- ----------------------------**

**-- Table structure for `t\_user`**

**-- ----------------------------**

**DROP TABLE IF EXISTS `t\_user`;**

**CREATE TABLE `t\_user` (**

**`id` int(11) NOT NULL AUTO\_INCREMENT,**

**`birthday` date DEFAULT NULL,**

**`gender` varchar(1) DEFAULT NULL,**

**`username` varchar(32) DEFAULT NULL,**

**`password` varchar(256) DEFAULT NULL,**

**`remark` varchar(32) DEFAULT NULL,**

**`station` varchar(1) DEFAULT NULL,**

**`telephone` varchar(11) DEFAULT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;**

**-- ----------------------------**

**-- Records of t\_user**

**-- ----------------------------**

**INSERT INTO `t\_user` VALUES ('1', null, null, 'admin', '$2a$10$u/BcsUUqZNWUxdmDhbnoeeobJy6IBsL1Gn/S0dMxI2RbSgnMKJ.4a', null, null, null);**

**INSERT INTO `t\_user` VALUES ('2', null, null, 'xiaoming', '$2a$10$3xW2nBjwBM3rx1LoYprVsemNri5bvxeOd/QfmO7UDFQhW2HRHLi.C', null, null, null);**

**INSERT INTO `t\_user` VALUES ('3', null, null, 'test', '$2a$10$zYJRscVUgHX1wqwu90WereuTmIg6h/JGirGG4SWBsZ60wVPCgtF8W', null, null, null);**

**-- ----------------------------**

**-- Table structure for `t\_role`**

**-- ----------------------------**

**DROP TABLE IF EXISTS `t\_role`;**

**CREATE TABLE `t\_role` (**

**`id` int(11) NOT NULL AUTO\_INCREMENT,**

**`name` varchar(32) DEFAULT NULL,**

**`keyword` varchar(64) DEFAULT NULL,**

**`description` varchar(128) DEFAULT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;**

**-- ----------------------------**

**-- Records of t\_role**

**-- ----------------------------**

**INSERT INTO `t\_role` VALUES ('1', '系统管理员', 'ROLE\_ADMIN', null);**

**INSERT INTO `t\_role` VALUES ('2', '导游', 'ROLE\_HEALTH\_MANAGER', null);**

**-- ----------------------------**

**-- Table structure for `t\_user\_role`**

**-- ----------------------------**

**DROP TABLE IF EXISTS `t\_user\_role`;**

**CREATE TABLE `t\_user\_role` (**

**`user\_id` int(11) NOT NULL,**

**`role\_id` int(11) NOT NULL,**

**PRIMARY KEY (`user\_id`,`role\_id`),**

**KEY `FK\_Reference\_8` (`role\_id`),**

**CONSTRAINT `FK\_Reference\_7` FOREIGN KEY (`user\_id`) REFERENCES `t\_user` (`id`),**

**CONSTRAINT `FK\_Reference\_8` FOREIGN KEY (`role\_id`) REFERENCES `t\_role` (`id`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8;**

**-- ----------------------------**

**-- Records of t\_user\_role**

**-- ----------------------------**

**INSERT INTO `t\_user\_role` VALUES ('1', '1');**

**INSERT INTO `t\_user\_role` VALUES ('2', '2');**

**-- ----------------------------**

**-- Table structure for `t\_permission`**

**-- ----------------------------**

**DROP TABLE IF EXISTS `t\_permission`;**

**CREATE TABLE `t\_permission` (**

**`id` int(11) NOT NULL AUTO\_INCREMENT,**

**`name` varchar(32) DEFAULT NULL,**

**`keyword` varchar(64) DEFAULT NULL,**

**`description` varchar(128) DEFAULT NULL,**

**PRIMARY KEY (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=27 DEFAULT CHARSET=utf8;**

**-- ----------------------------**

**-- Records of t\_permission**

**-- ----------------------------**

**INSERT INTO `t\_permission` VALUES ('1', '新增自由行', 'TRAVELITEM\_ADD', null);**

**INSERT INTO `t\_permission` VALUES ('2', '删除自由行', 'TRAVELITEM\_DELETE', null);**

**INSERT INTO `t\_permission` VALUES ('3', '编辑自由行', 'TRAVELITEM\_EDIT', null);**

**INSERT INTO `t\_permission` VALUES ('4', '查询自由行', 'TRAVELITEM\_QUERY', null);**

**INSERT INTO `t\_permission` VALUES ('5', '新增跟团游', 'TRAVELGROUP\_ADD', null);**

**INSERT INTO `t\_permission` VALUES ('6', '删除跟团游', 'TRAVELGROUP\_DELETE', null);**

**INSERT INTO `t\_permission` VALUES ('7', '编辑跟团游', 'TRAVELGROUP\_EDIT', null);**

**INSERT INTO `t\_permission` VALUES ('8', '查询跟团游', 'TRAVELGROUP\_QUERY', null);**

**INSERT INTO `t\_permission` VALUES ('9', '新增套餐', 'SETMEAL\_ADD', null);**

**INSERT INTO `t\_permission` VALUES ('10', '删除套餐', 'SETMEAL\_DELETE', null);**

**INSERT INTO `t\_permission` VALUES ('11', '编辑套餐', 'SETMEAL\_EDIT', null);**

**INSERT INTO `t\_permission` VALUES ('12', '查询套餐', 'SETMEAL\_QUERY', null);**

**INSERT INTO `t\_permission` VALUES ('13', '预约设置', 'ORDERSETTING', null);**

**INSERT INTO `t\_permission` VALUES ('14', '查看统计报表', 'REPORT\_VIEW', null);**

**INSERT INTO `t\_permission` VALUES ('15', '新增菜单', 'MENU\_ADD', null);**

**INSERT INTO `t\_permission` VALUES ('16', '删除菜单', 'MENU\_DELETE', null);**

**INSERT INTO `t\_permission` VALUES ('17', '编辑菜单', 'MENU\_EDIT', null);**

**INSERT INTO `t\_permission` VALUES ('18', '查询菜单', 'MENU\_QUERY', null);**

**INSERT INTO `t\_permission` VALUES ('19', '新增角色', 'ROLE\_ADD', null);**

**INSERT INTO `t\_permission` VALUES ('20', '删除角色', 'ROLE\_DELETE', null);**

**INSERT INTO `t\_permission` VALUES ('21', '编辑角色', 'ROLE\_EDIT', null);**

**INSERT INTO `t\_permission` VALUES ('22', '查询角色', 'ROLE\_QUERY', null);**

**INSERT INTO `t\_permission` VALUES ('23', '新增用户', 'USER\_ADD', null);**

**INSERT INTO `t\_permission` VALUES ('24', '删除用户', 'USER\_DELETE', null);**

**INSERT INTO `t\_permission` VALUES ('25', '编辑用户', 'USER\_EDIT', null);**

**INSERT INTO `t\_permission` VALUES ('26', '查询用户', 'USER\_QUERY', null);**

**-- ----------------------------**

**-- Table structure for `t\_role\_permission`**

**-- ----------------------------**

**DROP TABLE IF EXISTS `t\_role\_permission`;**

**CREATE TABLE `t\_role\_permission` (**

**`role\_id` int(11) NOT NULL,**

**`permission\_id` int(11) NOT NULL,**

**PRIMARY KEY (`role\_id`,`permission\_id`),**

**KEY `FK\_Reference\_12` (`permission\_id`),**

**CONSTRAINT `FK\_Reference\_11` FOREIGN KEY (`role\_id`) REFERENCES `t\_role` (`id`),**

**CONSTRAINT `FK\_Reference\_12` FOREIGN KEY (`permission\_id`) REFERENCES `t\_permission` (`id`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8;**

**-- ----------------------------**

**-- Records of t\_role\_permission**

**-- ----------------------------**

**INSERT INTO `t\_role\_permission` VALUES ('1', '1');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '1');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '2');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '2');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '3');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '3');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '4');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '4');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '5');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '5');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '6');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '6');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '7');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '7');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '8');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '8');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '9');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '9');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '10');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '10');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '11');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '11');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '12');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '12');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '13');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '13');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '14');**

**INSERT INTO `t\_role\_permission` VALUES ('2', '14');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '15');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '16');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '17');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '18');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '19');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '20');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '21');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '22');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '23');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '24');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '25');**

**INSERT INTO `t\_role\_permission` VALUES ('1', '26');**

**-- ----------------------------**

**-- Table structure for `t\_menu`**

**-- ----------------------------**

**DROP TABLE IF EXISTS `t\_menu`;**

**CREATE TABLE `t\_menu` (**

**`id` int(11) NOT NULL AUTO\_INCREMENT,**

**`name` varchar(128) DEFAULT NULL,**

**`linkUrl` varchar(128) DEFAULT NULL,**

**`path` varchar(128) DEFAULT NULL,**

**`priority` int(11) DEFAULT NULL,**

**`icon` varchar(64) DEFAULT NULL,**

**`description` varchar(128) DEFAULT NULL,**

**`parentMenuId` int(11) DEFAULT NULL,**

**`level` int(11) DEFAULT NULL,**

**PRIMARY KEY (`id`),**

**KEY `FK\_Reference\_13` (`parentMenuId`),**

**CONSTRAINT `FK\_Reference\_13` FOREIGN KEY (`parentMenuId`) REFERENCES `t\_menu` (`id`)**

**) ENGINE=InnoDB AUTO\_INCREMENT=22 DEFAULT CHARSET=utf8;**

**-- ----------------------------**

**-- Records of t\_menu**

**-- ----------------------------**

**INSERT INTO `t\_menu` VALUES ('1', '会员管理', null, '2', '1', 'fa-user-md', null, null, '1');**

**INSERT INTO `t\_menu` VALUES ('2', '会员档案', 'member.html', '/2-1', '1', null, null, '1', '2');**

**INSERT INTO `t\_menu` VALUES ('3', '会员信息上传', null, '/2-2', '2', null, null, '1', '2');**

**INSERT INTO `t\_menu` VALUES ('4', '会员统计', null, '/2-3', '3', null, null, '1', '2');**

**INSERT INTO `t\_menu` VALUES ('5', '预约管理', null, '3', '2', 'fa-tty', null, null, '1');**

**INSERT INTO `t\_menu` VALUES ('6', '预约列表', 'ordersettinglist.html', '/3-1', '1', null, null, '5', '2');**

**INSERT INTO `t\_menu` VALUES ('7', '预约设置', 'ordersetting.html', '/3-2', '2', null, null, '5', '2');**

**INSERT INTO `t\_menu` VALUES ('8', '套餐管理', 'setmeal.html', '/3-3', '3', null, null, '5', '2');**

**INSERT INTO `t\_menu` VALUES ('9', '跟团游管理', 'checkgroup.html', '/3-4', '4', null, null, '5', '2');**

**INSERT INTO `t\_menu` VALUES ('10', '自由行管理', 'checkitem.html', '/3-5', '5', null, null, '5', '2');**

**INSERT INTO `t\_menu` VALUES ('11', '旅游评估', null, '4', '3', 'fa-stethoscope', null, null, '1');**

**INSERT INTO `t\_menu` VALUES ('12', '大家一起游', null, '/4-1', '1', null, null, '11', '2');**

**INSERT INTO `t\_menu` VALUES ('13', '统计分析', null, '5', '4', 'fa-heartbeat', null, null, '1');**

**INSERT INTO `t\_menu` VALUES ('14', '会员数量', 'report\_member.html', '/5-1', '1', null, null, '13', '2');**

**INSERT INTO `t\_menu` VALUES ('15', '系统设置', null, '6', '5', 'fa-users', null, null, '1');**

**INSERT INTO `t\_menu` VALUES ('16', '菜单管理', 'menu.html', '/6-1', '1', null, null, '15', '2');**

**INSERT INTO `t\_menu` VALUES ('17', '权限管理', 'permission.html', '/6-2', '2', null, null, '15', '2');**

**INSERT INTO `t\_menu` VALUES ('18', '角色管理', 'role.html', '/6-3', '3', null, null, '15', '2');**

**INSERT INTO `t\_menu` VALUES ('19', '用户管理', 'user.html', '/6-4', '4', null, null, '15', '2');**

**INSERT INTO `t\_menu` VALUES ('20', '套餐占比', 'report\_setmeal.html', '/5-2', '2', null, null, '13', '2');**

**INSERT INTO `t\_menu` VALUES ('21', '运营数据', 'report\_business.html', '/5-3', '3', null, null, '13', '2');**

**-- ----------------------------**

**-- Table structure for `t\_role\_menu`**

**-- ----------------------------**

**DROP TABLE IF EXISTS `t\_role\_menu`;**

**CREATE TABLE `t\_role\_menu` (**

**`role\_id` int(11) NOT NULL,**

**`menu\_id` int(11) NOT NULL,**

**PRIMARY KEY (`role\_id`,`menu\_id`),**

**KEY `FK\_Reference\_10` (`menu\_id`),**

**CONSTRAINT `FK\_Reference\_10` FOREIGN KEY (`menu\_id`) REFERENCES `t\_menu` (`id`),**

**CONSTRAINT `FK\_Reference\_9` FOREIGN KEY (`role\_id`) REFERENCES `t\_role` (`id`)**

**) ENGINE=InnoDB DEFAULT CHARSET=utf8;**

**-- ----------------------------**

**-- Records of t\_role\_menu**

**-- ----------------------------**

**INSERT INTO `t\_role\_menu` VALUES ('1', '1');**

**INSERT INTO `t\_role\_menu` VALUES ('2', '1');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '2');**

**INSERT INTO `t\_role\_menu` VALUES ('2', '2');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '3');**

**INSERT INTO `t\_role\_menu` VALUES ('2', '3');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '4');**

**INSERT INTO `t\_role\_menu` VALUES ('2', '4');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '5');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '6');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '7');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '8');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '9');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '10');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '11');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '12');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '13');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '14');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '15');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '16');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '17');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '18');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '19');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '20');**

**INSERT INTO `t\_role\_menu` VALUES ('1', '21');**

### 【目标】

在美年旅游的项目中使用SpringSecurity完成认证和授权

### 【路径】

1：导入SpringSecurity环境

（1）pom.xml中添加依赖

（2）web.xml添加代理过滤器

2：实现认证和授权

（1）认证：SpringSecurityUserService.java

（2）创建Service类、Dao接口类、Mapper映射文件

（3）springmvc.xml（dubbo注解扫描范围扩大）

（4）spring-security.xml

（5）springmvc.xml（导入spring-security.xml）

（6）TravelItemController类（@PreAuthorize(“hasAuthority(‘CHECKITEM\_ADD’)”)：完成权限）

（7）travelitem.html（如果没有权限，可以提示错误信息）

（8）导入login.html测试登录

3：显示用户名

4：用户退出

### 【讲解】

## 1.1. ****导入Spring Security环境****

【路径】

1：pom.xml导入坐标

2：web.xml添加代理过滤器

### 1.1.1. 第一步：pom.xml导入坐标

在meinian\_parent父工程的pom.xml中导入Spring Security的maven坐标（已经引入）

<**dependency**>

 <**groupId**>org.springframework.security</**groupId**>

 <**artifactId**>spring-security-web</**artifactId**>

 <**version**>${spring.security.version}</**version**>

</**dependency**>

<**dependency**>

 <**groupId**>org.springframework.security</**groupId**>

 <**artifactId**>spring-security-config</**artifactId**>

 <**version**>${spring.security.version}</**version**>

</**dependency**>

### 1.1.2. 第二步：web.xml添加代理过滤器

在 meinian\_web工程的web.xml文件中配置用于整合Spring Security框架的过滤器DelegatingFilterProxy

<**filter**>

<!--

DelegatingFilterProxy用于整合第三方框架（代理过滤器，非真正的过滤器，真正的过滤器需要在spring的配置文件）

整合Spring Security时过滤器的名称必须为springSecurityFilterChain，

否则会抛出NoSuchBeanDefinitionException异常

-->

<**filter-name**>springSecurityFilterChain</**filter-name**>

<**filter-class**>org.springframework.web.filter.DelegatingFilterProxy</**filter-class**>

</**filter**>

<**filter-mapping**>

<**filter-name**>springSecurityFilterChain</**filter-name**>

<**url-pattern**>/\*</**url-pattern**>

</**filter-mapping**>

### 1.1.3 第三步 ：sql语句

# 1 使用登录名查询用户信息

**SELECT** \* **FROM** t\_user **WHERE** username = 'admin'

# 2 传递用户id查询角色集合

**SELECT** r.\* **FROM** t\_role r,t\_user\_role ur **WHERE** ur.role\_id = r.id **AND** ur.user\_id = 1

# 3 传递角色id查询权限集合

**SELECT** p.\* **FROM** t\_permission p,t\_role\_permission rp **WHERE** p.id = rp.permission\_id **AND** rp.role\_id = 1

## 1.2. 实现认证和授权

### 1.2.1. 第一步：SpringSecurityUserService.java

在meinian\_web工程中按照Spring Security框架要求提供SpringSecurityUserService，并且实现UserDetailsService接口

**package** com.atguigu.security;

**import** com.alibaba.dubbo.config.annotation.Reference;

**import** com.atguigu.pojo.Permission;

**import** com.atguigu.pojo.Role;

**import** com.atguigu.service.UserService;

**import** org.springframework.security.core.GrantedAuthority;

**import** org.springframework.security.core.authority.SimpleGrantedAuthority;

**import** org.springframework.security.core.userdetails.User;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.core.userdetails.UserDetailsService;

**import** org.springframework.security.core.userdetails.UsernameNotFoundException;

**import** org.springframework.stereotype.Component;

**import** java.util.ArrayList;

**import** java.util.List;

@Component

**public** **class** **SpringSecurityUserService** **implements** **UserDetailsService** {

@Reference //注意：此处要通过dubbo远程调用用户服务

**private** UserService userService;

//根据用户名查询用户信息

**public** UserDetails **loadUserByUsername**(String username) **throws** UsernameNotFoundException {

//远程调用用户服务，根据用户名查询用户信息

com.atguigu.pojo.User user = userService.findUserByUsername(username);

**if**(user == **null**){

//用户名不存在，抛出异常UsernameNotFoundException

**return** **null**;

}

List<GrantedAuthority> list = **new** ArrayList<GrantedAuthority>();

Set<Role> roles = user.getRoles();

**for**(Role role : roles){

Set<Permission> permissions = role.getPermissions();

**for**(Permission permission : permissions){

//授权

list.add(**new** SimpleGrantedAuthority(permission.getKeyword()));

}

}

/\*\*

\* User()

\* 1：指定用户名

\* 2：指定密码（SpringSecurity会自动对密码进行校验）

\* 3：传递授予的角色和权限

\*/

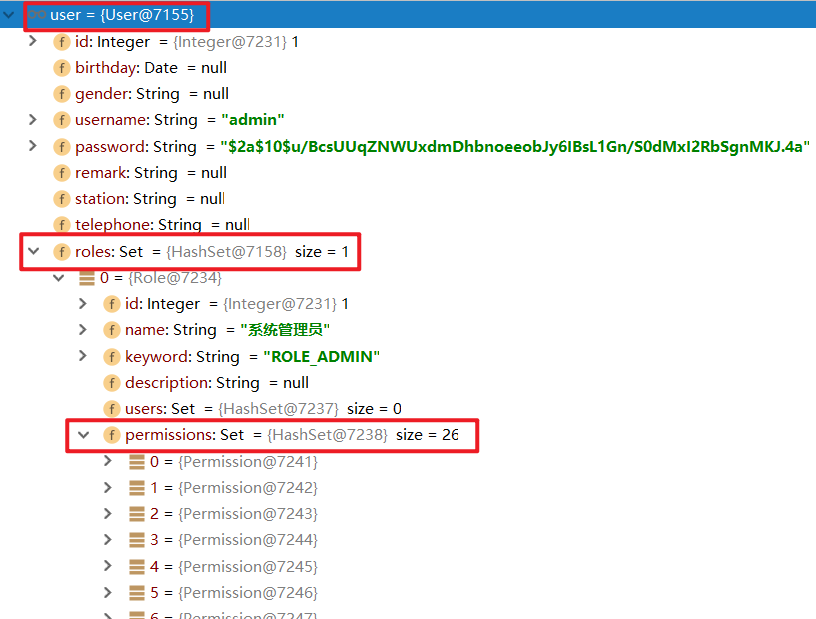
UserDetails userDetails = **new** User(username,user.getPassword(),list);

**return** userDetails;

}

}

使用debug跟踪调试，查看user。



### 1.2.2. 第二步：Service、Dao接口、Mapper映射文件

创建UserService服务接口、服务实现类、Dao接口、Mapper映射文件等

【路径】

1：UserService.java接口

2：UserServiceImpl.java类

3：UserDao.java（使用用户id查询用户）

4：RoleDao.java（使用用户id查询角色集合）

5：PermissionDao.java（使用角色id查询权限集合）

6：UserDao.xml（使用用户id查询用户）

7：RoleDao.xml（使用用户id查询角色集合）

8：PermissionDao.xml（使用角色id查询权限集合）

【讲解】

1：服务接口

**package** com.atguigu.service;

**import** com.atguigu.pojo.User;

/\*\*

\* 用户服务接口

\*/

**public** **interface** **UserService** {

User **findUserByUsername**(String username);

}

2：服务实现类

**package** com.atguigu.service;

**import** com.alibaba.dubbo.config.annotation.Service;

**import** com.atguigu.dao.UserDao;

**import** com.atguigu.pojo.User;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.transaction.annotation.Transactional;

@Service(interfaceClass = UserService.class)

@Transactional

**public** **class** **UserServiceImpl** **implements** **UserService** {

@Autowired

**private** UserDao userDao;

**public** User **findUserByUsername**(String username) {

User user = userDao.findUserByUsername(username);

**return** user;

}

}

3：Dao接口

（1）UserDao

**package** com.atguigu.dao;

**import** com.atguigu.pojo.User;

**import** org.springframework.stereotype.Repository;

/\*\*

\* 持久层Dao接口

\*/

@Repository

**public** **interface** **UserDao** {

User **findUserByUsername**(String username);

}

（2）RoleDao

**package** com.atguigu.dao;

**import** com.atguigu.pojo.Role;

**import** org.springframework.stereotype.Repository;

**import** java.util.Set;

/\*\*

\* 持久层Dao接口

\*/

@Repository

**public** **interface** **RoleDao** {

Set<Role> **findRolesByUserId**(Integer userId);

}

（3）PermissionDao

**package** com.atguigu.dao;

**import** com.atguigu.pojo.Permission;

**import** org.springframework.stereotype.Repository;

**import** java.util.Set;

/\*\*

\* 持久层Dao接口

\*/

@Repository

**public** **interface** **PermissionDao** {

Set<Permission> **findPermissionsByRoleId**(Integer roleId);

}

4：Mapper映射文件

（1）UserDao.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.atguigu.dao.UserDao">

<**resultMap** **id**="userMap" **type**="com.atguigu.pojo.User">

<**id** **column**="id" **property**="id"></**id**>

<**result** **column**="birthday" **property**="birthday"></**result**>

<**result** **column**="gender" **property**="gender"></**result**>

<**result** **column**="username" **property**="username"></**result**>

<**result** **column**="password" **property**="password"></**result**>

<**result** **column**="remark" **property**="remark"></**result**>

<**result** **column**="station" **property**="station"></**result**>

<**result** **column**="telephone" **property**="telephone"></**result**>

<**collection** **property**="roles" **column**="id" **select**="com.atguigu.dao.RoleDao.findRolesByUserId"></**collection**>

</**resultMap**>

<**select** **id**="findUserByUsername" **parameterType**="string" **resultMap**="userMap">

select u.\* from t\_user u where username = #{username}

</**select**>

</**mapper**>

（2）RoleDao.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.atguigu.dao.RoleDao">

<**resultMap** **id**="roleMap" **type**="role">

<**id** **column**="id" **property**="id"></**id**>

<**result** **column**="name" **property**="name"></**result**>

<**result** **column**="keyword" **property**="keyword"></**result**>

<**result** **column**="description" **property**="description"></**result**>

<**collection** **property**="permissions" **column**="id" **select**="com.atguigu.dao.PermissionDao.findPermissionsByRoleId"></**collection**>

</**resultMap**>

<**select** **id**="findRolesByUserId" **parameterType**="int" **resultMap**="roleMap">

SELECT r.\* FROM t\_role r,t\_user\_role ur WHERE r.id = ur.role\_id AND ur.user\_id = #{userId}

</**select**>

</**mapper**>

（3）PermissionDao.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.atguigu.dao.PermissionDao">

<**select** **id**="findPermissionsByRoleId" **parameterType**="int" **resultType**="com.atguigu.pojo.Permission">

SELECT p.\* FROM t\_permission p,t\_role\_permission rp WHERE p.id = rp.permission\_id AND rp.role\_id = #{roleId}

</**select**>

</**mapper**>

### 1.2.3. 第三步：springmvc.xml

修改meinian\_web工程中的springmvc.xml文件，修改dubbo批量扫描的包路径

之前的包扫描

<!--批量扫描-->

<**dubbo:annotation** **package**="com.atguigu.controller" />

**现在的包扫描**

<!--批量扫描-->

<**dubbo:annotation** **package**="com.atguigu" />

**注意：**此处原来扫描的包为com.atguigu.controller，现在改为com.atguigu包的目的是需要将我们上面定义的SpringSecurityUserService也扫描到，因为在SpringSecurityUserService的loadUserByUsername方法中需要通过dubbo远程调用名称为UserService的服务。

### 1.2.4. 第四步：spring-security.xml

【路径】

1：定义哪些链接可以放行

2：定义哪些链接不可以放行，即需要有角色、权限才可以放行

3：认证管理，定义登录账号名和密码，并授予访问的角色、权限

4：设置在页面可以通过iframe访问受保护的页面，默认为不允许访问，需要添加security:frame-options policy=“SAMEORIGIN”

【讲解】

在meinian\_web工程中提供spring-security.xml配置文件

<?xml version="1.0" encoding="UTF-8"?>

<**beans** **xmlns**="http://www.springframework.org/schema/beans"

**xmlns:xsi**="http://www.w3.org/2001/XMLSchema-instance"

**xmlns:context**="http://www.springframework.org/schema/context"

**xmlns:dubbo**="http://code.alibabatech.com/schema/dubbo"

**xmlns:mvc**="http://www.springframework.org/schema/mvc"

**xmlns:security**="http://www.springframework.org/schema/security"

**xsi:schemaLocation**="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://code.alibabatech.com/schema/dubbo

http://code.alibabatech.com/schema/dubbo/dubbo.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security.xsd">

<!--一：定义哪些链接可以放行-->

<!--

http：用于定义相关权限控制

指定哪些资源不需要进行权限校验，可以使用通配符

-->

<**security:http** **security**="none" **pattern**="/js/\*\*" />

<**security:http** **security**="none" **pattern**="/css/\*\*" />

<**security:http** **security**="none" **pattern**="/img/\*\*" />

<**security:http** **security**="none" **pattern**="/plugins/\*\*" />

<**security:http** **security**="none" **pattern**="/login.html" />

<!--开启注解方式权限控制-->

<**security:global-method-security** **pre-post-annotations**="enabled" />

<!--

二：定义哪些链接不可以放行，即需要有角色、权限才可以放行

http：用于定义相关权限控制

auto-config：是否自动配置

设置为true时框架会提供默认的一些配置，例如提供默认的登录页面、登出处理等

设置为false时需要显示提供登录表单配置，否则会报错

use-expressions：用于指定intercept-url中的access属性是否使用表达式

-->

<**security:http** **auto-config**="true" **use-expressions**="true">

<!--只要认证通过就可以访问-->

<!--

intercept-url：定义一个拦截规则

pattern：对哪些url进行权限控制

access：在请求对应的URL时需要什么权限，默认配置时它应该是一个以逗号分隔的角色列表，

请求的用户只需拥有其中的一个角色就能成功访问对应的URL

isAuthenticated()：需要经过认证后才能访问（不是匿名用户）

-->

<**security:intercept-url** **pattern**="/pages/\*\*" **access**="isAuthenticated()" />

<!--

form-login：定义表单登录信息

-->

<**security:form-login** **login-page**="/login.html"

**username-parameter**="username"

**password-parameter**="password"

**login-processing-url**="/login.do"

**default-target-url**="/pages/main.html"

**authentication-failure-url**="/login.html"

**always-use-default-target**="true"

/>

<!--

csrf：对应CsrfFilter过滤器

disabled：是否启用CsrfFilter过滤器，如果使用自定义登录页面需要关闭此项，否则登录操作会被禁用（403）

-->

<**security:csrf** **disabled**="true"></**security:csrf**>

<!--

logout：退出登录

logout-url：退出登录操作对应的请求路径

logout-success-url：退出登录后的跳转页面

-->

<**security:logout** **logout-url**="/logout.do" **logout-success-url**="/login.html" **invalidate-session**="true"/>

</**security:http**>

<!--配置密码加密对象-->

<**bean** **id**="passwordEncoder" **class**="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />

<!--

三：认证管理，定义登录账号名和密码，并授予访问的角色、权限

authentication-manager：认证管理器，用于处理认证操作

-->

<**security:authentication-manager**>

<!--

authentication-provider：认证提供者，执行具体的认证逻辑

-->

<**security:authentication-provider** **user-service-ref**="springSecurityUserService">

<!--指定密码加密策略-->

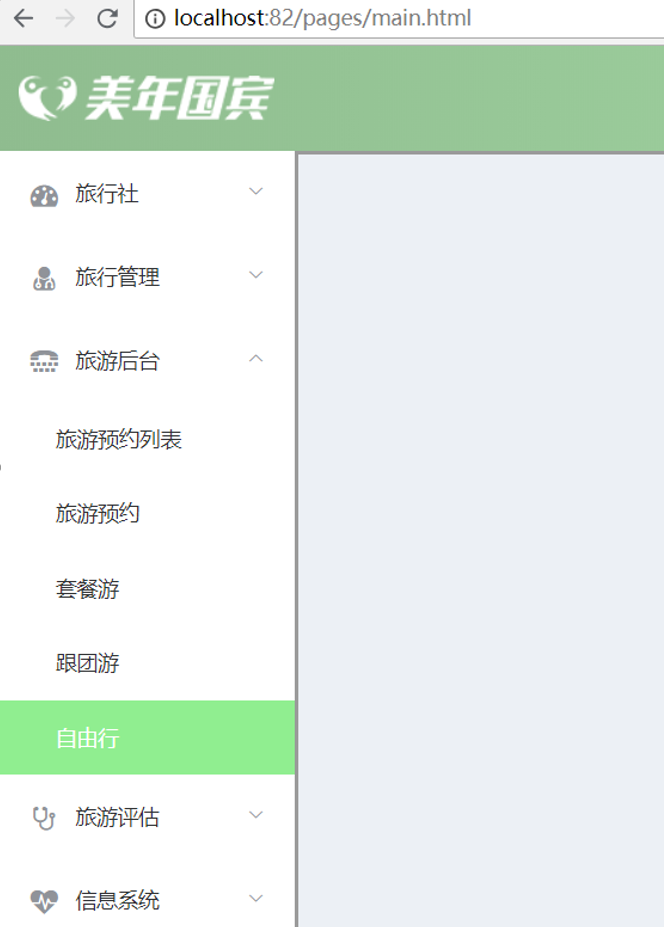
<**security:password-encoder** **ref**="passwordEncoder"></**security:password-encoder**>

</**security:authentication-provider**>

</**security:authentication-manager**>

</**beans**>

这里注意：如果出现以下问题



使用下面的配置，在spring-security.xml中添加。

放置到<security:http auto-config=“true” use-expressions=“true”>里面

<**security:headers**>

<!--设置在页面可以通过iframe访问受保护的页面，默认为不允许访问-->

<**security:frame-options** **policy**="SAMEORIGIN"></**security:frame-options**>

</**security:headers**>

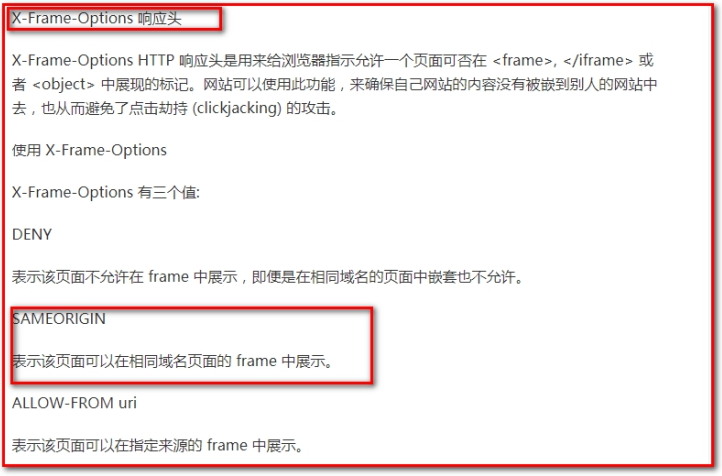
是因为我们在main.html中定义：如果不配置springSecurity会认为iframe访问的html页面是受保护的页面，不允许访问。

<**el-container**>

<**iframe** **name**="right" **class**="el-main" **src**="travelitem.html" **width**="100%" **height**="580px" **frameborder**="0"></**iframe**>

</**el-container**>

备注：



### 1.2.5. 第五步：springmvc.xml

在springmvc.xml文件中引入spring-security.xml文件

<**import** **resource**="classpath:spring-security.xml"></**import**>

### 1.2.6. 第六步：TravelItemController类

在Controller的方法上加入权限控制注解，此处以 TravelItemController 为例

**package** com.atguigu.controller;

**import** com.alibaba.dubbo.config.annotation.Reference;

**import** com.atguigu.constant.MessageConstant;

**import** com.atguigu.entity.PageResult;

**import** com.atguigu.entity.QueryPageBean;

**import** com.atguigu.entity.Result;

**import** com.atguigu.pojo.TravelItem;

**import** com.atguigu.service.TravelItemService;

**import** com.github.pagehelper.Page;

**import** org.springframework.security.access.prepost.PreAuthorize;

**import** org.springframework.ui.freemarker.FreeMarkerTemplateUtils;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** java.util.List;

@RestController

@RequestMapping("/travelItem")

**public** **class** **TravelItemController** {

@Reference

**private** TravelItemService travelItemService;

@RequestMapping("/findAll")

**public** Result **findAll**(){

List<TravelItem> lists = travelItemService.findAll();

**return** **new** **Result**(**true**,MessageConstant.QUERY\_TRAVELITEM\_SUCCESS,lists);

}

@RequestMapping("/edit")

@PreAuthorize("hasAuthority('TRAVELITEM\_EDIT')")//权限校验

**public** Result **edit**(@RequestBody TravelItem travelItem){

travelItemService.edit(travelItem);

**return** **new** **Result**(**true**,MessageConstant.EDIT\_TRAVELITEM\_SUCCESS);

}

@RequestMapping("/findById")

**public** Result **findById**(Integer id){

TravelItem travelItem = travelItemService.findById(id);

**return** **new** **Result**(**true**,MessageConstant.QUERY\_TRAVELITEM\_SUCCESS,travelItem);

}

@RequestMapping("/delete")

@PreAuthorize("hasAuthority('TRAVELITEM\_DELETE')")//权限校验，使用TRAVELITEM\_DELETE123测试

**public** Result **delete**(Integer id){

**try** {

travelItemService.delete(id);

**return** **new** **Result**(**true**,MessageConstant.DELETE\_TRAVELITEM\_SUCCESS);

}**catch** (RuntimeException e){

**return** **new** **Result**(**false**,MessageConstant.DELETE\_TRAVELITEM\_FAIL);

}**catch** (Exception e) {

e.printStackTrace();

}

**return** **new** **Result**(**false**,MessageConstant.DELETE\_TRAVELITEM\_FAIL);

}

@RequestMapping("/findPage")

@PreAuthorize("hasAuthority('TRAVELITEM\_QUERY')")//权限校验

**public** PageResult **findPage**(@RequestBody QueryPageBean queryPageBean){

PageResult pageResult = travelItemService.findPage(queryPageBean.getCurrentPage(),

queryPageBean.getPageSize(),queryPageBean.getQueryString());

**return** pageResult;

}

//@RequestBody:表示对象和json数据进行互转

@RequestMapping("/add")

@PreAuthorize("hasAuthority('TRAVELITEM\_ADD')")//权限校验

**public** Result **add**(@RequestBody TravelItem travelItem){

**try** {

// 增加，更新，删除 ，就不需要往data里面添加数据

// 查询：list 对象

travelItemService.add(travelItem);

Result result = **new** Result(**true**, MessageConstant.ADD\_TRAVELITEM\_SUCCESS);

**return** result;

} **catch** (Exception e) {

e.printStackTrace();

}

**return** **new** **Result**(**false**, MessageConstant.ADD\_TRAVELITEM\_SUCCESS);

}

}

### 1.2.7. 第七步：CustomAccessDeniedHandler类

添加页面，没有权限时提示信息设置。

1. 在<security:http>标签中增加<security:access-denied-handler>

<!--自定义异常处理-->  
<security:access-denied-handler ref="**customAccessDeniedHandler**"/>

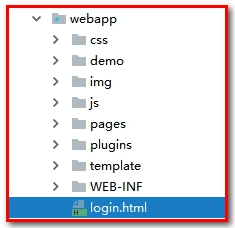
2.增加自定义处理类

package com.atguigu.security;  
  
import com.alibaba.fastjson.JSON;  
import com.atguigu.entity.Result;  
import org.springframework.security.access.AccessDeniedException;  
import org.springframework.security.web.access.AccessDeniedHandler;  
import org.springframework.stereotype.Component;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
/\*\*  
 \* 无权访问处理类  
 \*/  
@Component  
public class **CustomAccessDeniedHandler** implements **AccessDeniedHandler** {  
  
 @Override  
 public void **handle**(HttpServletRequest request,HttpServletResponse response,  
 AccessDeniedException accessDeniedException) throws IOException,ServletException {  
 if (isAjaxRequest(request)) {// AJAX请求,使用response发送403  
 Result result = new Result(false, "无权访问","403");  
 String json = JSON.toJSONString(result);  
 response.getWriter().print(json);  
 } else{// 同步请求处理  
 request.getRequestDispatcher("/pages/error/403.html").forward(request,response);  
 }  
 }  
  
 /\*\*  
 \* 判断是否为ajax请求  
 \*/  
 public static boolean **isAjaxRequest**(HttpServletRequest request) {  
 if (request.getHeader("accept").indexOf("application/json") > -1  
 || (request.getHeader("X-Requested-With") != null && request.getHeader("X-Requested-With").equalsIgnoreCase("XMLHttpRequest"))) {  
 return true;  
 }  
 return false;  
 }  
}

3.增加/pages/error/403.html页面

### 1.2.8. 第八步：导入login.html页面





此时login.html是可放行的页面，而pages下的页面必须认证之后才能访问的页面

## 1.3. 显示用户名

【路径】

1：引入js

2：定义username属性

3：使用钩子函数，调用ajax，查询登录用户（从SpringSecurity中获取），赋值username属性

4：修改页面，使用{{username}}显示用户信息

【讲解】

前面我们已经完成了认证和授权操作，如果用户认证成功后需要在页面展示当前用户的用户名。Spring Security在认证成功后会将用户信息保存到框架提供的上下文对象中，所以此处我们就可以调用Spring Security框架提供的API获取当前用户的username并展示到页面上。

实现步骤：

第一步：在main.html页面中修改，定义username模型数据基于VUE的数据绑定展示用户名，发送ajax请求获取username

（1）：引入js

<**script** **src**="../js/axios-0.18.0.js"></**script**>

（2）：定义username属性

（3）：使用钩子函数，调用ajax

<script>

**new** Vue({

el: '#app',

data:{

menuList:[],

username:null // 显示用户名

},

//发送请求获取当前登录用户的用户名

created:**function** () {

//发送请求获取当前登录用户的用户名

// 返回Result(flag,message,data)，data放置User对象

axios.get("/user/getUsername.do").then((response)=>{

**if**(response.data.flag){

**this**.username = response.data.data.username;

}

}).catch((error)=>{

**this**.$message.error("获取用户名失败");

})

}

});

$(**function**() {

**var** wd = 200;

$(".el-main").css('width', $('body').width() - wd + 'px');

});

</**script**>

（4）：修改页面

<**div** **class**="avatar-wrapper">

<**img** **src**="../img/user2-160x160.jpg" **class**="user-avatar">

<!--显示用户名-->

{{username}}

</**div**>

第二步：创建UserController并提供getUsername方法

Spring Security使用一个Authentication对象来描述当前用户的相关信息。这个Authentication对象不需要我们自己去创建，在与系统交互的过程中，Spring Security会自动为我们创建相应的Authentication对象，然后赋值给当前的SecurityContext。通过Authentication.getPrincipal()可以获取到代表当前用户的信息。

**package** com.atguigu.controller;

**import** com.alibaba.dubbo.config.annotation.Reference;

**import** com.atguigu.constant.MessageConstant;

**import** com.atguigu.entity.PageResult;

**import** com.atguigu.entity.QueryPageBean;

**import** com.atguigu.entity.Result;

**import** com.atguigu.service.UserService;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.User;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** java.util.List;

@RestController

@RequestMapping("/user")

**public** **class** **UserController** {

@Reference

**private** UserService userService;

//获取当前登录用户的用户名

@RequestMapping("/getUsername")

**public** Result **getUsername**()**throws** Exception{

**try**{

User user = (User)SecurityContextHolder.getContext().getAuthentication().getPrincipal();

**return** **new** **Result**(**true**, MessageConstant.GET\_USERNAME\_SUCCESS,user);

}**catch** (Exception e){

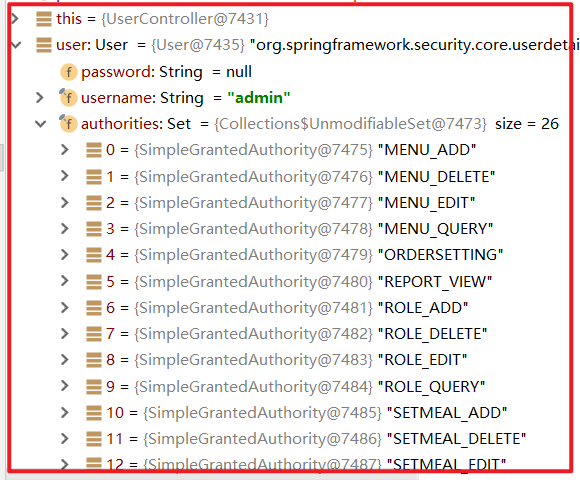
**return** **new** **Result**(**false**, MessageConstant.GET\_USERNAME\_FAIL);

}

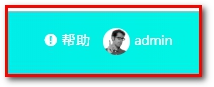
}

}

通过debug调试可以看到Spring Security框架在其上下文中保存的用户相关信息：



显示当前登录人：



## 1.4. 用户退出

【路径】

1：在main.html中提供的退出菜单上加入超链接

2：在spring-security.xml文件中配置

【讲解】

第一步：在main.html中提供的退出菜单上加入超链接

<**el-dropdown-item** **divided**>

<**span** **style**="display:block;"><**a** **href**="/logout.do">退出</**a**></**span**>

</**el-dropdown-item**>

第二步：在spring-security.xml文件中配置

<!--

logout：退出登录

logout-url：退出登录操作对应的请求路径

logout-success-url：退出登录后的跳转页面

invalidate-session="true" 销毁session

-->

<**security:logout** **logout-url**="/logout.do" **logout-success-url**="/login.html" **invalidate-session**="true"/>

### 【小结】

1：导入SpringSecurity环境

（1）pom.xml中添加依赖

（2）web.xml添加代理过滤器

2：实现认证和授权

（1）认证：SpringSecurityUserService.java，实现UserDetailsService接口

（2）创建Service类、Dao接口类、Mapper映射文件（使用用户名查询当前用户信息，包括角色集合和权限集合）

（3）springmvc.xml（dubbo注解扫描范围扩大）

（4）spring-security.xml（重点）

（5）springmvc.xml（导入spring-security.xml）

（6）CheckItemController类（@PreAuthorize(“hasAuthority(‘CHECKITEM\_ADD’)”)：对类中的方法完成权限控制）

（7）CustomAccessDeniedHandler.java（如果没有权限，返回“无权访问”提示信息）

（8）导入login.html测试登录

3：显示用户名

从SecurityContextHolder对象中获取认证的用户信息

@RequestMapping(value = "/getUsername")

**public** Result **getUsername**(){

**try** {

// 从SpringSecurity中获取认证用户的信息

User user = (User)SecurityContextHolder.getContext().getAuthentication().getPrincipal();

**return** **new** **Result**(**true**, MessageConstant.GET\_USERNAME\_SUCCESS,user);

} **catch** (Exception e) {

e.printStackTrace();

**return** **new** **Result**(**false**, MessageConstant.GET\_USERNAME\_FAIL);

}

}

4：用户退出