介绍

SpringSecurity

https://vip.tulingxueyuan.cn/detail/p_602e308de4b029faba185b3b/6

SpringSecurity是一个强大的可高度定制的认证和授权框架，对于Spring应用来说它是一套Web安全标准。SpringSecurity注重于为Java应用提供认证和授权功能，像所有的Spring项目一样，它对自定义需求具有强大的扩展性。

JWT

JWT是JSON WEB TOKEN的缩写，它是基于 RFC 7519 标准定义的一种可以安全传输的的JSON对象，由于使用了数字签名，所以是可信任和安全的。

JWT的组成

- JWT token的格式：header.payload.signature
- header中用于存放签名的生成算法

```
{"alg": "HS512"}
```

- payload中用于存放用户名、token的生成时间和过期时间

```
{"sub":"admin","created":1489079981393,"exp":1489684781}
```

- signature为以header和payload生成的签名，一旦header和payload被篡改，验证将失败

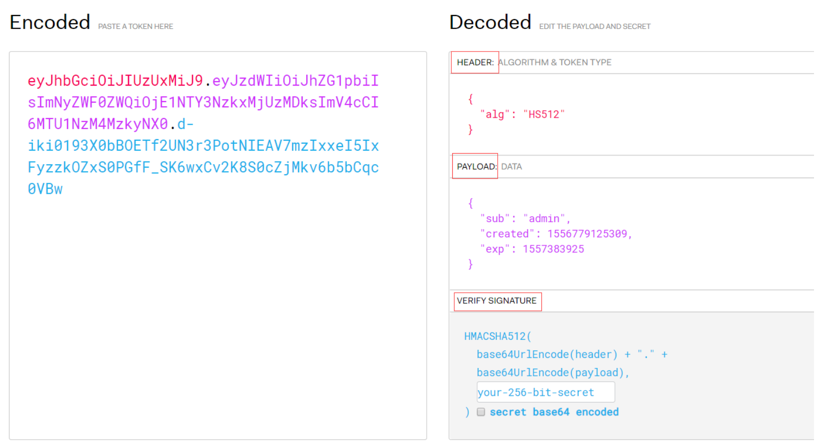
```
//secret为加密算法的密钥
String signature = HMACSHA512(base64UrlEncode(header) + "." +base64UrlEncode(payload),secret)
```

JWT实例

这是一个JWT的字符串

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImNyZWF0ZWQiOjE1NTY3NzkxMjUzMDksImV4cCI6MTU1NzM4MzkyNX0.d-iki0193X0bBOETf2UN3r3PotNIEAV7mzIxxeI5IxFyzzkOZxS0PGfF_SK6wxCv2K8S0cZjMkv6b5bCqc0VBw
```

可以在该网站上获得解析结果：https://jwt.io/


Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImNyZWF0ZWQiOjE1NTY3NzkxMjUzMDksImV4cCI6MTU1NzM4MzkyNX0.d-iki0193X0bBOETf2UN3r3PotNIEAV7mzIxxeI5IxFyzzkOZxS0PGfF_SK6wxCv2K8S0cZjMkv6b5bCqc0VBw
```

```
{
  "alg": "HS512"
}
```

```
{
  "sub": "admin",
  "created": 1556779125309,
  "exp": 1557383925
}
```

```
HMACSHA512(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

JWT实现认证和授权的原理

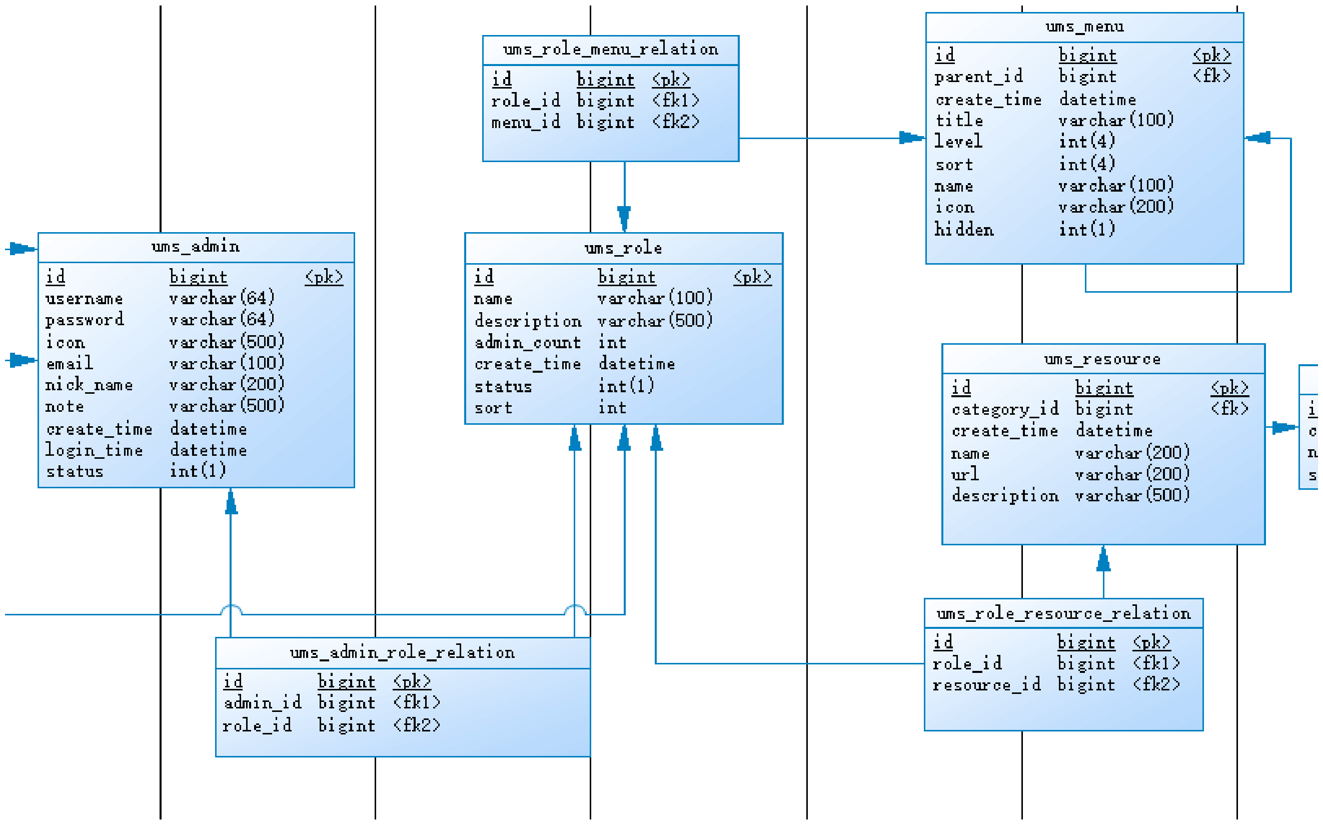
- 用户调用登录接口，登录成功后获取到JWT的token；
- 之后用户每次调用接口都在http的header中添加一个叫Authorization的头，值为JWT的token；
- 后台程序通过对Authorization头中信息的解码及数字签名校验来获取其中的用户信息，从而实现认证和授权。

Hutool

Hutool是一个丰富的Java开源工具包,它帮助我们简化每一行代码，减少每一个方法，mall项目采用了此工具包。

项目使用表说明

- `ums_admin`：后台用户表
- `ums_role`：后台用户角色表
- `ums_resource`：后台用户权限表
- `ums_admin_role_relation`：后台用户和角色关系表，用户与角色是多对多关系
- `ums_role_permission_relation`：后台用户角色和权限关系表，角色与权限是多对多关系
- `ums_admin_permission_relation`：后台用户和权限关系表(除角色中定义的权限以外的加减权限)，加权限是指用户比角色多出的权限，减权限是指用户比角色少的权限



ums_admin

后台用户表，定义了后台用户的一些基本信息。

```sql
create table ums_admin
(
 id bigint not null auto_increment,
 username varchar(64) comment '用户名',
 password varchar(64) comment '密码',
 icon varchar(500) comment '头像',
 email varchar(100) comment '邮箱',
 nick_name varchar(200) comment '昵称',
 note varchar(500) comment '备注信息',
 create_time datetime comment '创建时间',
 login_time datetime comment '最后登录时间',
 status int(1) default 1 comment '帐号启用状态：0->禁用；1->启用',
 primary key (id)
);
```

ums_role

后台用户角色表，定义了后台用户角色的一些基本信息，通过给后台用户分配角色来实现菜单和资源的分配。

```sql
create table ums_role
(
 id bigint not null auto_increment,
 name varchar(100) comment '名称',
 description varchar(500) comment '描述',
 admin_count int comment '后台用户数量',
 create_time datetime comment '创建时间',
 status int(1) default 1 comment '启用状态：0->禁用；1->启用',
 sort int default 0,
 primary key (id)
);
```

ums_admin_role_relation

后台用户和角色关系表，多对多关系表，一个角色可以分配给多个用户。

```sql
create table ums_admin_role_relation
(
 id bigint not null auto_increment,
 admin_id bigint,
 role_id bigint,
 primary key (id)
);
```

ums_menu

后台菜单表，用于控制后台用户可以访问的菜单，支持隐藏、排序和更改名称、图标。

```sql
create table ums_menu
(
 id bigint not null auto_increment,
 parent_id bigint comment '父级ID',
 create_time datetime comment '创建时间',
 title varchar(100) comment '菜单名称',
 level int(4) comment '菜单级数',
 sort int(4) comment '菜单排序',
 name varchar(100) comment '前端名称',
 icon varchar(200) comment '前端图标',
 hidden int(1) comment '前端隐藏',
 primary key (id)
);
```

ums_resource

后台资源表，用于控制后台用户可以访问的接口，使用了Ant路径的匹配规则，可以使用通配符定义一系列接口的权限。

```sql
create table ums_resource
(
 id bigint not null auto_increment,
 category_id bigint comment '资源分类ID',
 create_time datetime comment '创建时间',
 name varchar(200) comment '资源名称',
 url varchar(200) comment '资源URL',
 description varchar(500) comment '描述',
 primary key (id)
);
```

ums_resource_category

后台资源分类表，在细粒度进行权限控制时，可能资源会比较多，所以设计了个资源分类的概念，便于给角色分配资源。

```sql
create table ums_resource_category
(
 id bigint not null auto_increment,
 create_time datetime comment '创建时间',
 name varchar(200) comment '分类名称',
 sort int(4) comment '排序',
 primary key (id)
);
```

ums_role_menu_relation

后台角色菜单关系表，多对多关系，可以给一个角色分配多个菜单。

```sql
create table ums_role_menu_relation
(
 id bigint not null auto_increment,
 role_id bigint comment '角色ID',
 menu_id bigint comment '菜单ID',
 primary key (id)
);
```

ums_role_resource_relation

后台角色资源关系表，多对多关系，可以给一个角色分配多个资源。

```sql
create table ums_role_resource_relation
(
 id bigint not null auto_increment,
 role_id bigint comment '角色ID',
 resource_id bigint comment '资源ID',
 primary key (id)
);

```

整合SpringSecurity及JWT

在pom.xml中添加项目依赖

```
<!--SpringSecurity依赖配置-->
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!--Hutool Java工具包-->
<dependency>
 <groupId>cn.hutool</groupId>
 <artifactId>hutool-all</artifactId>
  <version>4.5.7</version>
 </dependency>
 <!--JWT(Json Web Token)登录支持-->
 <dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.0</version>
 </dependency>
```

添加JWT token的工具类

用于生成和解析JWT token的工具类

相关方法说明:

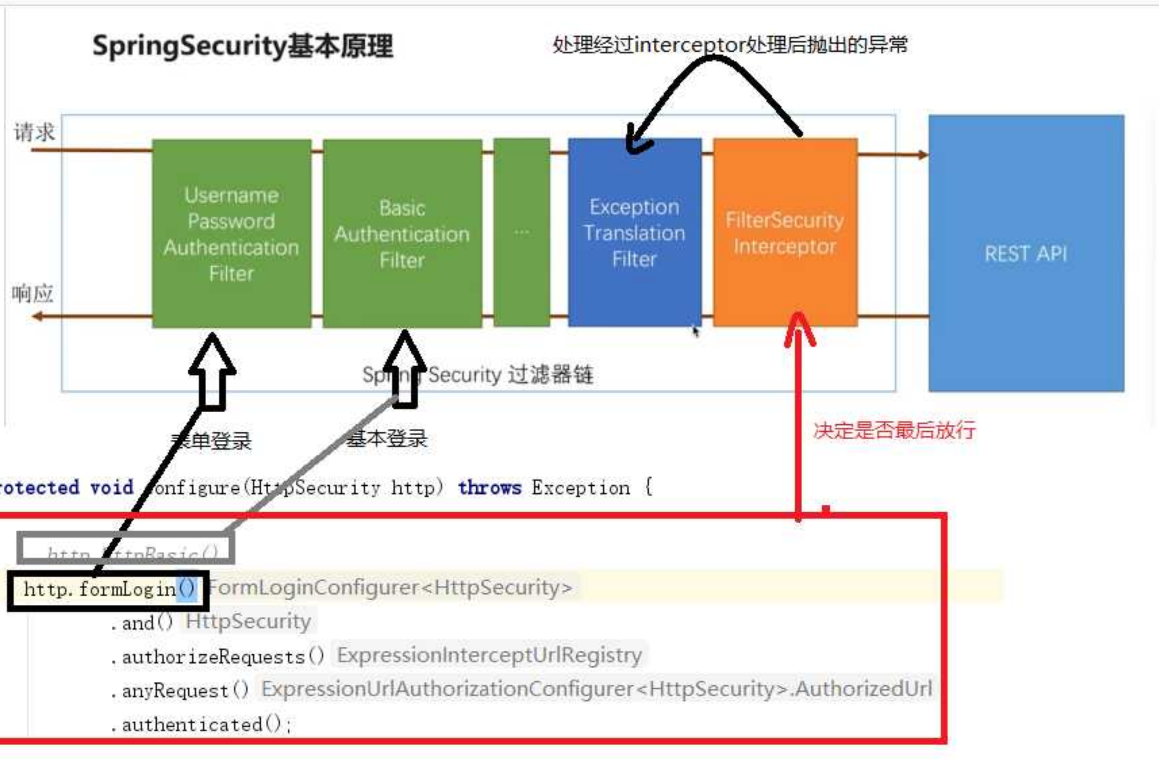
- generateToken(UserDetails userDetails) :用于根据登录用户信息生成token
- getUserNameFromToken(String token)：从token中获取登录用户的信息
- validateToken(String token, UserDetails userDetails)：判断token是否还有效

```
package com.tulingxueyuan.mall.common.utils;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
 import org.springframework.stereotype.Component;

 import java.util.Date;
 import java.util.HashMap;
 import java.util.Map;

 /**
  * JwtToken生成的工具类
  * Created by xushu on 2018/4/26.
  */
 @Component
 public class JwtTokenUtil {
  private static final Logger LOGGER = LoggerFactory.getLogger(JwtTokenUtil.class);
  private static final String CLAIM_KEY_USERNAME = "sub";
  private static final String CLAIM_KEY_CREATED = "created";
  @Value("${jwt.secret}")
  private String secret;
  @Value("${jwt.expiration}")
```

```
private Long expiration;

/**
* 根据负责生成JWT的token
*/
private String generateToken(Map<String, Object> claims) {
return Jwts.builder()
.setClaims(claims)
.setExpiration(generateExpirationDate())
.signWith(SignatureAlgorithm.HS512, secret)
.compact();
}

/**
* 从token中获取JWT中的负载
*/
private Claims getClaimsFromToken(String token) {
Claims claims = null;
try {
claims = Jwts.parser()
.setSigningKey(secret)
.parseClaimsJws(token)
.getBody();
} catch (Exception e) {
LOGGER.info("JWT格式验证失败:{}",token);
}
return claims;
}

/**
* 生成token的过期时间
*/
private Date generateExpirationDate() {
return new Date(System.currentTimeMillis() + expiration * 1000);
}

/**
* 从token中获取登录用户名
*/
public String getUserNameFromToken(String token) {
String username;
try {
Claims claims = getClaimsFromToken(token);
username = claims.getSubject();
} catch (Exception e) {
username = null;
}
return username;
}

/**
* 验证token是否还有效
*
* @param token 客户端传入的token
* @param userDetails 从数据库中查询出来的用户信息
*/
public boolean validateToken(String token, UserDetails userDetails) {
String username = getUserNameFromToken(token);
return username.equals(userDetails.getUsername()) && !isTokenExpired(token);
}

```

```
/**
* 判断token是否已经失效
*/
private boolean isTokenExpired(String token) {
Date expiredDate = getExpiredDateFromToken(token);
return expiredDate.before(new Date());
}

/**
* 从token中获取过期时间
*/
private Date getExpiredDateFromToken(String token) {
Claims claims = getClaimsFromToken(token);
return claims.getExpiration();
}

/**
* 根据用户信息生成token
*/
public String generateToken(UserDetails userDetails) {
Map<String, Object> claims = new HashMap<>();
claims.put(CLAIM_KEY_USERNAME, userDetails.getUsername());
claims.put(CLAIM_KEY_CREATED, new Date());
return generateToken(claims);
}

/**
* 判断token是否可以被刷新
*/
public boolean canRefresh(String token) {
return !isTokenExpired(token);
}

/**
* 刷新token
*/
public String refreshToken(String token) {
Claims claims = getClaimsFromToken(token);
claims.put(CLAIM_KEY_CREATED, new Date());
return generateToken(claims);
}
}
```

添加SpringSecurity的配置类


SpringSecurity基本原理
处理经过interceptor处理后抛出的异常
请求
响应
Username Password Authentication Filter
Basic Authentication Filter
...
Exception Translation Filter
FilterSecurity Interceptor
REST API
Spring Security 过滤器链
表单登录
基本登录
决定是否最后放行

```java
protected void configure(HttpSecurity http) throws Exception {
    http.httpBasic()
    http.formLogin() FormLoginConfigurer<HttpSecurity>
            .and() HttpSecurity
            .authorizeRequests() ExpressionInterceptUrlRegistry
            .anyRequest() ExpressionUrlAuthorizationConfigurer<HttpSecurity>.AuthorizedUrl
            .authenticated();
}
```

```java
package com.tulingxueyuan.mall.config;

import com.macro.mall.tiny.component.JwtAuthenticationTokenFilter;
import com.macro.mall.tiny.component.RestAuthenticationEntryPoint;
import com.macro.mall.tiny.component.RestfulAccessDeniedHandler;
import com.macro.mall.tiny.dto.AdminUserDetails;
import com.macro.mall.tiny.mbg.model.UmsAdmin;
import com.macro.mall.tiny.mbg.model.UmsPermission;
import com.macro.mall.tiny.service.UmsAdminService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import java.util.List;


/**
 * SpringSecurity的配置
 * Created by xushu 2018/4/26.
 */
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

```java
@Autowired
private UmsAdminService adminService;
@Autowired
private RestfulAccessDeniedHandler restfulAccessDeniedHandler;
@Autowired
private RestAuthenticationEntryPoint restAuthenticationEntryPoint;

@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
httpSecurity.csrf()// 由于使用的是JWT，我们这里不需要csrf
.disable()
.sessionManagement()// 基于token，所以不需要session
.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
.and()
.authorizeRequests()
.antMatchers(HttpMethod.GET, // 允许对于网站静态资源的无授权访问
"/",
"/*.html",
"/favicon.ico",
"/**/*.html",
"/**/*.css",
"/**/*.js",
"/swagger-resources/**",
"/v2/api-docs/**"
)
.permitAll()
.antMatchers("/admin/login", "/admin/register")// 对登录注册要允许匿名访问
.permitAll()
.antMatchers(HttpMethod.OPTIONS)//跨域请求会先进行一次options请求
.permitAll()
// .antMatchers("/**")//测试时全部运行访问
// .permitAll()
.anyRequest()// 除上面外的所有请求全部需要鉴权认证
.authenticated();
// 禁用缓存
httpSecurity.headers().cacheControl();
// 添加JWT filter
httpSecurity.addFilterBefore(jwtAuthenticationTokenFilter(), UsernamePasswordAuthenticationFilter.class);
//添加自定义未授权和未登录结果返回
httpSecurity.exceptionHandling()
.accessDeniedHandler(restfulAccessDeniedHandler)
.authenticationEntryPoint(restAuthenticationEntryPoint);
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
auth.userDetailsService(userDetailsService())
.passwordEncoder(passwordEncoder());
}

@Bean
public PasswordEncoder passwordEncoder() {
return new BCryptPasswordEncoder();
}

@Bean
public UserDetailsService userDetailsService() {
//获取登录用户信息
return username -> {
UmsAdmin admin = adminService.getAdminByUsername(username);
if (admin != null) {
List<UmsPermission> permissionList = adminService.getPermissionList(admin.getId());
 return new AdminUserDetails(admin,permissionList);
 }
 throw new UsernameNotFoundException("用户名或密码错误");
 };
 }

 @Bean
 public JwtAuthenticationTokenFilter jwtAuthenticationTokenFilter(){
 return new JwtAuthenticationTokenFilter();
 }

 @Bean
 @Override
 public AuthenticationManager authenticationManagerBean() throws Exception {
 return super.authenticationManagerBean();
 }

}
```

相关依赖及方法说明

- configure(HttpSecurity httpSecurity)：用于配置需要拦截的url路径、jwt过滤器及出异常后的处理器；
- configure(AuthenticationManagerBuilder auth)：用于配置UserDetailsService及PasswordEncoder；
- RestfulAccessDeniedHandler：当用户没有访问权限时的处理器，用于返回JSON格式的处理结果；
- RestAuthenticationEntryPoint：当未登录或token失效时，返回JSON格式的结果；
- UserDetailsService:SpringSecurity定义的核心接口，用于根据用户名获取用户信息，需要自行实现；
- UserDetails：SpringSecurity定义用于封装用户信息的类（主要是用户信息和权限），需要自行实现；
- PasswordEncoder：SpringSecurity定义的用于对密码进行编码及比对的接口，目前使用的是 BCryptPasswordEncoder；
- JwtAuthenticationTokenFilter：在用户名和密码校验前添加的过滤器，如果有jwt的token，会自行根据token信息进行登录。

添加RestfulAccessDeniedHandler

```
package com.tulingxueyuan.mall.component;

import cn.hutool.json.JSONUtil;
import com.macro.mall.tiny.common.api.CommonResult;
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;

 /**
  * 当访问接口没有权限时，自定义的返回结果
  * Created by xushu on 2018/4/26.
  */
 @Component
 public class RestfulAccessDeniedHandler implements AccessDeniedHandler{
  @Override
  public void handle(HttpServletRequest request,
  HttpServletResponse response,
  AccessDeniedException e) throws IOException, ServletException {
  response.setCharacterEncoding("UTF-8");
  response.setContentType("application/json");
  response.getWriter().println(JSONUtil.parse(CommonResult.forbidden(e.getMessage())));
response.getWriter().flush();
 }
}
```

添加RestAuthenticationEntryPoint

```
package com.tulingxueyuan.mall.component;

import cn.hutool.json.JSONUtil;
import com.macro.mall.tiny.common.api.CommonResult;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 当未登录或者token失效访问接口时，自定义的返回结果
 * Created by xushu 2018/5/14.
 */
@Component
public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint {
 @Override
 public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {
 response.setCharacterEncoding("UTF-8");
 response.setContentType("application/json");
 response.getWriter().println(JSONUtil.parse(CommonResult.unauthorized(authException.getMessage())));
 response.getWriter().flush();
 }
}
```

添加AdminUserDetails

```
package com.tulingxueyuan.mall.dto;

import com.macro.mall.tiny.mbg.model.UmsAdmin;
import com.macro.mall.tiny.mbg.model.UmsPermission;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

/**
 * SpringSecurity需要的用户详情
 * Created by xushu on 2018/4/26.
 */
public class AdminUserDetails implements UserDetails {
 private UmsAdmin umsAdmin;
 private List<UmsPermission> permissionList;
 public AdminUserDetails(UmsAdmin umsAdmin, List<UmsPermission> permissionList) {
 this.umsAdmin = umsAdmin;
 this.permissionList = permissionList;
 }

 @Override
 public Collection<? extends GrantedAuthority> getAuthorities() {
 //返回当前用户的权限
```

```java
return permissionList.stream()
.filter(permission -> permission.getValue()!=null)
.map(permission ->new SimpleGrantedAuthority(permission.getValue()))
.collect(Collectors.toList());
}

@Override
public String getPassword() {
return umsAdmin.getPassword();
}

@Override
public String getUsername() {
return umsAdmin.getUsername();
}

@Override
public boolean isAccountNonExpired() {
return true;
}

@Override
public boolean isAccountNonLocked() {
return true;
}

@Override
public boolean isCredentialsNonExpired() {
return true;
}

@Override
public boolean isEnabled() {
return umsAdmin.getStatus().equals(1);
}
}
```

添加JwtAuthenticationTokenFilter

在用户名和密码校验前添加的过滤器，如果请求中有jwt的token且有效，会取出token中的用户名，然后调用SpringSecurity的API进行登录操作。

由于 security 默认的登录方式不支持这种方式，需要重写过滤器，修改为支持从 header 中获取 token 值，根据 token 设置当前登录对象。

思路大概为：App 端登录成功，后端返回 token 值，App 保存在本地后在后面的每次请求中都在 header 中带着 token 进行请求。后端重写过滤器，在过滤器中解析 token 值并将 token 中带的对象放到登录用户中，让 security 认为请求已经是在登录状态下进行。

```java
package com.tulingxueyuan.mall.component;

import com.macro.mall.tiny.common.utils.JwtTokenUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * JWT登录授权过滤器
 * Created by xushu on 2018/4/26.
 */
public class JwtAuthenticationTokenFilter extends OncePerRequestFilter {
 private static final Logger LOGGER = LoggerFactory.getLogger(JwtAuthenticationTokenFilter.class);
 @Autowired
 private UserDetailsService userDetailsService;
 @Autowired
 private JwtTokenUtil jwtTokenUtil;
 @Value("${jwt.tokenHeader}")
 private String tokenHeader;
 @Value("${jwt.tokenHead}")
 private String tokenHead;

 @Override
 protected void doFilterInternal(HttpServletRequest request,
 HttpServletResponse response,
 FilterChain chain) throws ServletException, IOException {
 String authHeader = request.getHeader(this.tokenHeader);
 if (authHeader != null && authHeader.startsWith(this.tokenHead)) {
 String authToken = authHeader.substring(this.tokenHead.length());// The part after "Bearer "
 String username = jwtTokenUtil.getUserNameFromToken(authToken);
 LOGGER.info("checking username:{}", username);
 if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
 UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);
 if (jwtTokenUtil.validateToken(authToken, userDetails)) {
 UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
 LOGGER.info("authenticated user:{}", username);
 SecurityContextHolder.getContext().setAuthentication(authentication);
 }
 }
 }
 chain.doFilter(request, response);
 }
}
```

登录注册功能实现

添加UmsAdminController类

实现了后台用户登录、注册及获取权限的接口

```java
package com.macro.mall.tiny.controller;

import com.macro.mall.tiny.common.api.CommonResult;
import com.macro.mall.tiny.dto.UmsAdminLoginParam;
import com.macro.mall.tiny.mbg.model.UmsAdmin;
import com.macro.mall.tiny.mbg.model.UmsPermission;
import com.macro.mall.tiny.service.UmsAdminService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 后台用户管理
 * Created by xushu on 2018/4/26.
 */
@Controller
@Api(tags = "UmsAdminController", description = "后台用户管理")
@RequestMapping("/admin")
public class UmsAdminController {
 @Autowired
 private UmsAdminService adminService;
 @Value("${jwt.tokenHeader}")
 private String tokenHeader;
 @Value("${jwt.tokenHead}")
 private String tokenHead;

 @ApiOperation(value = "用户注册")
 @RequestMapping(value = "/register", method = RequestMethod.POST)
 @ResponseBody
 public CommonResult<UmsAdmin> register(@RequestBody UmsAdmin umsAdminParam, BindingResult result) {
 UmsAdmin umsAdmin = adminService.register(umsAdminParam);
 if (umsAdmin == null) {
 CommonResult.failed();
 }
 return CommonResult.success(umsAdmin);
 }

 @ApiOperation(value = "登录以后返回token")
 @RequestMapping(value = "/login", method = RequestMethod.POST)
 @ResponseBody
 public CommonResult login(@RequestBody UmsAdminLoginParam umsAdminLoginParam, BindingResult result) {
 String token = adminService.login(umsAdminLoginParam.getUsername(), umsAdminLoginParam.getPassword());
 if (token == null) {
 return CommonResult.validateFailed("用户名或密码错误");
 }
 Map<String, String> tokenMap = new HashMap<>();
 tokenMap.put("token", token);
 tokenMap.put("tokenHead", tokenHead);
 return CommonResult.success(tokenMap);
 }

 @ApiOperation("获取用户所有权限（包括+-权限）")
 @RequestMapping(value = "/permission/{adminId}", method = RequestMethod.GET)
 @ResponseBody
 public CommonResult<List<UmsPermission>> getPermissionList(@PathVariable Long adminId) {
 List<UmsPermission> permissionList = adminService.getPermissionList(adminId);
 return CommonResult.success(permissionList);
 }
}
```

添加UmsAdminService接口

```java
package com.macro.mall.tiny.service;

import com.macro.mall.tiny.mbg.model.UmsAdmin;
import com.macro.mall.tiny.mbg.model.UmsPermission;

import java.util.List;

/**
 * 后台管理员Service
  * Created by xushu on 2018/4/26.
  */
 public interface UmsAdminService {
  /**
  * 根据用户名获取后台管理员
  */
  UmsAdmin getAdminByUsername(String username);

  /**
  * 注册功能
  */
  UmsAdmin register(UmsAdmin umsAdminParam);

  /**
  * 登录功能
  * @param username 用户名
  * @param password 密码
  * @return 生成的JWT的token
  */
  String login(String username, String password);

  /**
  * 获取用户所有权限（包括角色权限和+-权限）
  */
  List<UmsPermission> getPermissionList(Long adminId);
 }
```

添加UmsAdminServiceImpl类

```java
package com.macro.mall.tiny.service.impl;

import com.macro.mall.tiny.common.utils.JwtTokenUtil;
import com.macro.mall.tiny.dao.UmsAdminRoleRelationDao;
import com.macro.mall.tiny.dto.UmsAdminLoginParam;
import com.macro.mall.tiny.mbg.mapper.UmsAdminMapper;
import com.macro.mall.tiny.mbg.model.UmsAdmin;
import com.macro.mall.tiny.mbg.model.UmsAdminExample;
import com.macro.mall.tiny.mbg.model.UmsPermission;
 import com.macro.mall.tiny.service.UmsAdminService;
 import org.slf4j.Logger;
 import org.slf4j.LoggerFactory;
 import org.springframework.beans.BeanUtils;
 import org.springframework.beans.factory.annotation.Autowired;
 import org.springframework.beans.factory.annotation.Value;
 import org.springframework.security.authentication.AuthenticationManager;
 import org.springframework.security.authentication.BadCredentialsException;
 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
 import org.springframework.security.core.AuthenticationException;
 import org.springframework.security.core.context.SecurityContextHolder;
 import org.springframework.security.core.userdetails.UserDetails;
 import org.springframework.security.core.userdetails.UserDetailsService;
 import org.springframework.security.crypto.password.PasswordEncoder;
 import org.springframework.stereotype.Service;

 import java.util.Date;
 import java.util.List;

```

```java
/**
 * UmsAdminService实现类
 * Created by xushu on 2018/4/26.
 */
@Service
public class UmsAdminServiceImpl implements UmsAdminService {
 private static final Logger LOGGER = LoggerFactory.getLogger(UmsAdminServiceImpl.class);
 @Autowired
 private UserDetailsService userDetailsService;
 @Autowired
 private JwtTokenUtil jwtTokenUtil;
 @Autowired
 private PasswordEncoder passwordEncoder;
 @Value("${jwt.tokenHead}")
 private String tokenHead;
 @Autowired
 private UmsAdminMapper adminMapper;
 @Autowired
 private UmsAdminRoleRelationDao adminRoleRelationDao;

 @Override
 public UmsAdmin getAdminByUsername(String username) {
 UmsAdminExample example = new UmsAdminExample();
 example.createCriteria().andUsernameEqualTo(username);
 List<UmsAdmin> adminList = adminMapper.selectByExample(example);
 if (adminList != null && adminList.size() > 0) {
 return adminList.get(0);
 }
 return null;
 }

 @Override
 public UmsAdmin register(UmsAdmin umsAdminParam) {
 UmsAdmin umsAdmin = new UmsAdmin();
 BeanUtils.copyProperties(umsAdminParam, umsAdmin);
 umsAdmin.setCreateTime(new Date());
 umsAdmin.setStatus(1);
 //查询是否有相同用户名的用户
 UmsAdminExample example = new UmsAdminExample();
 example.createCriteria().andUsernameEqualTo(umsAdmin.getUsername());
 List<UmsAdmin> umsAdminList = adminMapper.selectByExample(example);
 if (umsAdminList.size() > 0) {
 return null;
 }
 //将密码进行加密操作
 String encodePassword = passwordEncoder.encode(umsAdmin.getPassword());
 umsAdmin.setPassword(encodePassword);
 adminMapper.insert(umsAdmin);
 return umsAdmin;
 }

 @Override
 public String login(String username, String password) {
 String token = null;
 try {
 UserDetails userDetails = userDetailsService.loadUserByUsername(username);
 if (!passwordEncoder.matches(password, userDetails.getPassword())) {
 throw new BadCredentialsException("密码不正确");
 }
 UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
 SecurityContextHolder.getContext().setAuthentication(authentication);
 token = jwtTokenUtil.generateToken(userDetails);
 } catch (AuthenticationException e) {
 LOGGER.warn("登录异常:{}", e.getMessage());
 }
 return token;
 }


 @Override
 public List<UmsPermission> getPermissionList(Long adminId) {
  return adminRoleRelationDao.getPermissionList(adminId);
  }
 }
```

修改Swagger的配置

通过修改配置实现调用接口自带Authorization头，这样就可以访问需要登录的接口了。

```
package com.macro.mall.tiny.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.ApiKey;
import springfox.documentation.service.AuthorizationScope;
import springfox.documentation.service.SecurityReference;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spi.service.contexts.SecurityContext;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;
import java.util.List;

/**
 * Swagger2API文档的配置
 */
@Configuration
@EnableSwagger2
public class Swagger2Config {
 @Bean
 public Docket createRestApi(){
 return new Docket(DocumentationType.SWAGGER_2)
 .apiInfo(apiInfo())
 .select()
 //为当前包下controller生成API文档
 .apis(RequestHandlerSelectors.basePackage("com.macro.mall.tiny.controller"))
 .paths(PathSelectors.any())
 .build()
 //添加登录认证
 .securitySchemes(securitySchemes())
 .securityContexts(securityContexts());
 }

 private ApiInfo apiInfo() {
 return new ApiInfoBuilder()
 .title("SwaggerUI演示")
 .description("mall-tiny")
 .contact("macro")
```

```java
.version("1.0")
.build();
}

private List<ApiKey> securitySchemes() {
//设置请求头信息
List<ApiKey> result = new ArrayList<>();
ApiKey apiKey = new ApiKey("Authorization", "Authorization", "header");
result.add(apiKey);
return result;
}

private List<SecurityContext> securityContexts() {
//设置需要登录认证的路径
List<SecurityContext> result = new ArrayList<>();
result.add(getContextByPath("/brand/.*"));
return result;
}

private SecurityContext getContextByPath(String pathRegex){
return SecurityContext.builder()
.securityReferences(defaultAuth())
.forPaths(PathSelectors.regex(pathRegex))
.build();
}

private List<SecurityReference> defaultAuth() {
List<SecurityReference> result = new ArrayList<>();
AuthorizationScope authorizationScope = new AuthorizationScope("global", "accessEverything");
AuthorizationScope[] authorizationScopes = new AuthorizationScope[1];
authorizationScopes[0] = authorizationScope;
result.add(new SecurityReference("Authorization", authorizationScopes));
return result;
}
}
```

给PmsBrandController接口中的方法添加访问权限

- 给查询接口添加`pms:brand:read`权限
- 给修改接口添加`pms:brand:update`权限
- 给删除接口添加`pms:brand:delete`权限
- 给添加接口添加`pms:brand:create`权限

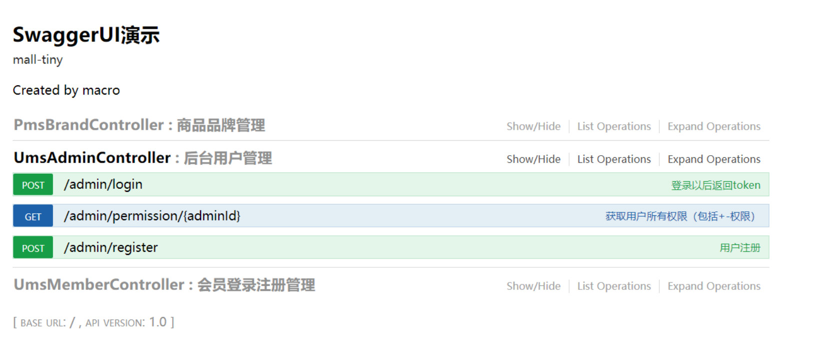
例子:

```java
@PreAuthorize("hasAuthority('pms:brand:read')")
public CommonResult<List<PmsBrand>> getBrandList() {
 return CommonResult.success(brandService.listAllBrand());
}
```

认证与授权流程演示

运行项目，访问API

Swagger api地址：http://localhost:8080/swagger-ui.html


SwaggerUI演示
mall-tiny

Created by macro

PmsBrandController : 商品品牌管理 — Show/Hide | List Operations | Expand Operations

UmsAdminController : 后台用户管理 — Show/Hide | List Operations | Expand Operations

POST /admin/login — 登录以后返回token

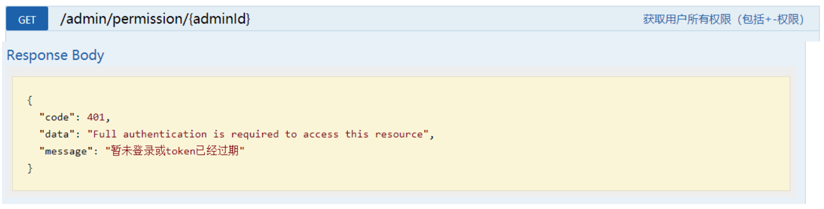
GET /admin/permission/{adminId} — 获取用户所有权限（包括+-权限）

POST /admin/register — 用户注册

UmsMemberController : 会员登录注册管理 — Show/Hide | List Operations | Expand Operations

[BASE URL: / , API VERSION: 1.0]

未登录前访问接口

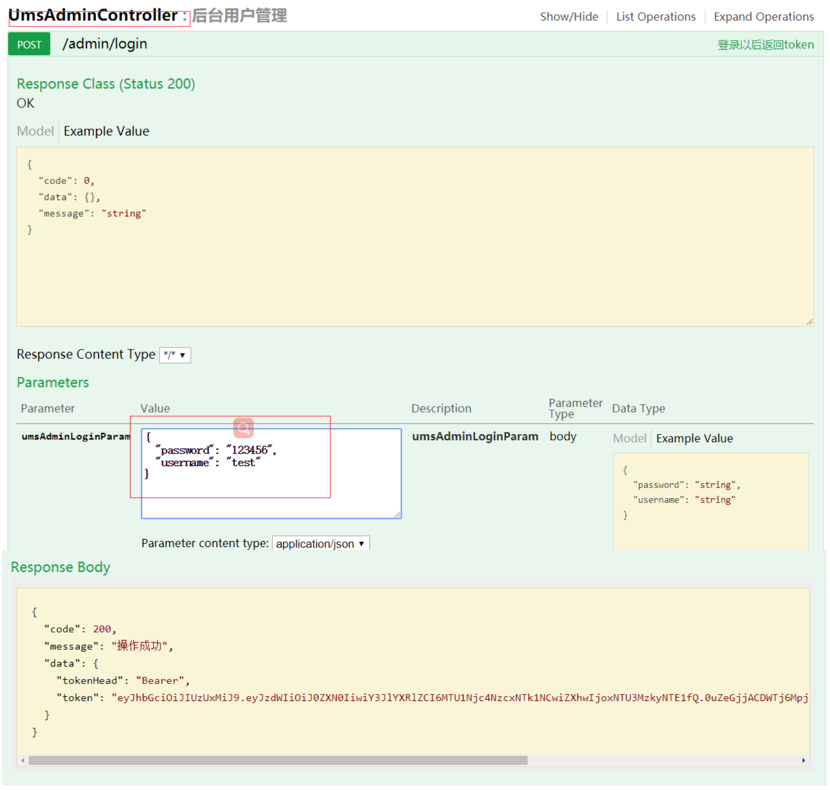

GET /admin/permission/{adminId} — 获取用户所有权限（包括+-权限）

Response Body

```
{
  "code": 401,
  "data": "Full authentication is required to access this resource",
  "message": "暂未登录或token已经过期"
}
```

登录后访问接口

- 进行登录操作：登录帐号test 123456


UmsAdminController : 后台用户管理 — Show/Hide | List Operations | Expand Operations

POST /admin/login — 登录以后返回token

Response Class (Status 200)
OK

Model | Example Value

```
{
  "code": 0,
  "data": {},
  "message": "string"
}
```

Response Content Type */*

Parameters

Parameter	Value	Description	Parameter Type	Data Type

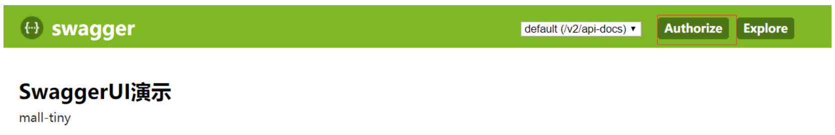
umsAdminLoginParam	{
	"password": "123456",
	"username": "test"
	}
		umsAdminLoginParam	body	Model | Example Value
				{
				"password": "string",
				"username": "string"
				}

Parameter content type: application/json

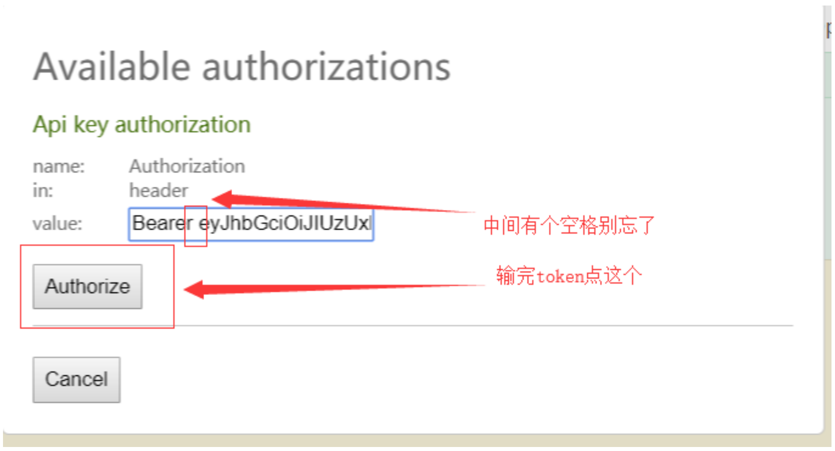
Response Body

```
{
  "code": 200,
  "message": "操作成功",
  "data": {
    "tokenHead": "Bearer",
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXN0IiwiY3JlYXRlZCI6MTU1Njc4NzcxNTk1NCwiZXhwIjoxNTU3MzkyNTE1fQ.0uZeGjjACDWTj6Mpj
  }
}
```

- 点击Authorize按钮，在弹框中输入登录接口中获取到的token信息



SwaggerUI演示
mall-tiny

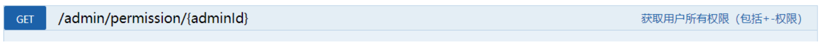


Available authorizations

Api key authorization

name: Authorization
in: header
value: Bearer eyJhbGciOiJIUzUx

- 登录后访问获取权限列表接口，发现已经可以正常访问



GET /admin/permission/{adminId} 获取用户所有权限（包括+-权限）