**# flood-Permission**

**## 简介**

基于jjwt实现的一套用于前后端分离项目的权限控制框架，用于实现基于token验证的Web权限框架。

<br>

**## 导入**

**\*\*SpringBoot项目导入：\*\***

```xml

<dependency>

<groupId>cn.flood</groupId>

<artifactId>flood-permission</artifactId>

<version>2.0.0</version>

</dependency>

```

**\*\*jar包方式导入：\*\***

<br>

**## 用法**

**### 在SpringBoot中使用**

在你的`Application`类上面加入`@EnableJwtPermission`注解，在`application.properties`有如下配置可选：

```text

**## 0是 redisTokenStore ，1是 jdbcTokenStore (未实现)，默认是0**

fpermission.store-type=0

**## 拦截路径，默认是/\*\***

fpermission.path=/\*\*

**## 排除拦截路径，默认无**

fpermission.exclude-path=/,/index,/login

**## 单个用户最大token数，默认-1不限制**

fpermission.max-token=2

**## 自定义视图解析器**

fpermission.viewResolverPrefix=/static/

fpermission.viewResolverSuffix=.html

<br>

**### 登录签发token**

```java

@RestController

public class LoginController {

@Autowired

private TokenStore tokenStore;

@PostMapping("/login")

public Result login(String account, String password, HttpServletRequest request) {

// 你的验证逻辑

// ......

// 签发token

UserToken token = tokenStore.createNewToken(userId, permissions, roles, info);

Map<String, Object> map = new HashMap<>();

map.put("\_token",token.getAccessToken())

return Result.ok(map);

}

}

```

token默认过期时间是一天，设置过期时间方法（单位秒）：

UserToken token = tokenStore.createNewToken(userId, permissions, roles, 60*\*60\**24\*30);

<br>

**### 使用注解或代码限制权限**

1.使用注解的方式：

```text

// 需要有system权限才能访问

@RequiresPermissions("system")

// 需要有system和front权限才能访问,logical可以不写,默认是AND

@RequiresPermissions(value={"system","front"}, logical=Logical.AND)

// 需要有system或者front权限才能访问

@RequiresPermissions(value={"system","front"}, logical=Logical.OR)

// 需要有admin或者user角色才能访问

@RequiresRoles(value={"admin","user"}, logical=Logical.OR)

```

> 注解只能加在Controller的方法上面。

<br>

2.使用代码的方式：

```text

//是否有system权限

SubjectUtil.hasPermission(request, "system");

//是否有system或者front权限

SubjectUtil.hasPermission(request, new String[]{"system","front"}, Logical.OR);

//是否有admin或者user角色

SubjectUtil.hasRole(request, new String[]{"admin","user"}, Logical.OR)

```

<br>

**### 前端传递token**

放在参数里面用`\_token`传递：

```javascript

$.get("/xxx", { \_token: token }, function(data) {

});

```

放在header里面用`Authorization`、`Bearer`传递：

```javascript

$.ajax({

url: "/xxx",

beforeSend: function(xhr) {

xhr.setRequestHeader("Authorization", 'Bearer '+ token);

},

success: function(data){ }

});

```

<br>

**## 注意事项**

**### 异常处理**

&emsp;flood-permistion在token验证失败和没有权限的时候抛出异常，框架定义了几个异常：

| 描述 | 错误信息 |

|:-------------|:----------------------------------|

| token不能为空 | 错误信息“token不能为空”，错误码400 |

| token验证失败 | 错误信息“身份验证失败”，错误码401 |

| token已经过期 | 错误信息“登录已过期”，错误码402 |

| 没有权限 | 错误信息“没有访问权限”，错误码403 |

| token不在系统中 | 错误信息“token不存在”，错误码404 |

<br>

**### 主动让token失效：**

```java

public class XX {

@Autowired

private TokenStore tokenStore;

public void xx(){

// 移除用户的某个token

tokenStore.removeToken(userId, access\_token);

// 移除用户的全部token

tokenStore.removeTokensByUserId(userId);

}

}

```

<br>

**### 更新角色和权限列表**

&emsp;修改了用户的角色和权限需要同步更新框架中的角色和权限：

```java

public class XX {

@Autowired

private TokenStore tokenStore;

public void xx(){

// 更新用户的角色列表

tokenStore.updateRolesByUserId(userId, roles);

// 更新用户的权限列表

tokenStore.updatePermissionsByUserId(userId, permissions);

}

}

```

<br>

**### 获取当前的用户信息**

```text

UserToken token = SubjectUtil.getToken(request);

```

<br>