**8.2权限拦截·使用装饰器实现权限拦截**

为了配合后续的权限管理，本工程化使用自定义装饰器的方式来实现权限拦截。

# 新增目录mauth

mkdir mauth

# 新增mauth/\_\_init\_\_.py文件

这里为了演示，暂时只对添加用户接口进行权限拦截。

from functools import wraps  
from flask import abort  
  
  
class HasPerm(ｏｂｊｅｃｔ):  
 """  
 权限装饰器类  
 """  
  
 def \_\_init\_\_(self, access=None, name=None):  
 self.access = access  
 self.name = name  
  
 def \_\_call\_\_(self, func):  
 @wraps(func)  
 def wrapped\_function(\*args, \*\*kwargs):  
 # 这里处理权限拦截的逻辑  
 print(f"access:{self.access}")  
 if self.access in ['user:save']:  
 abort(403)  
 else:  
 return func(\*args, \*\*kwargs)  
  
 return wrapped\_function

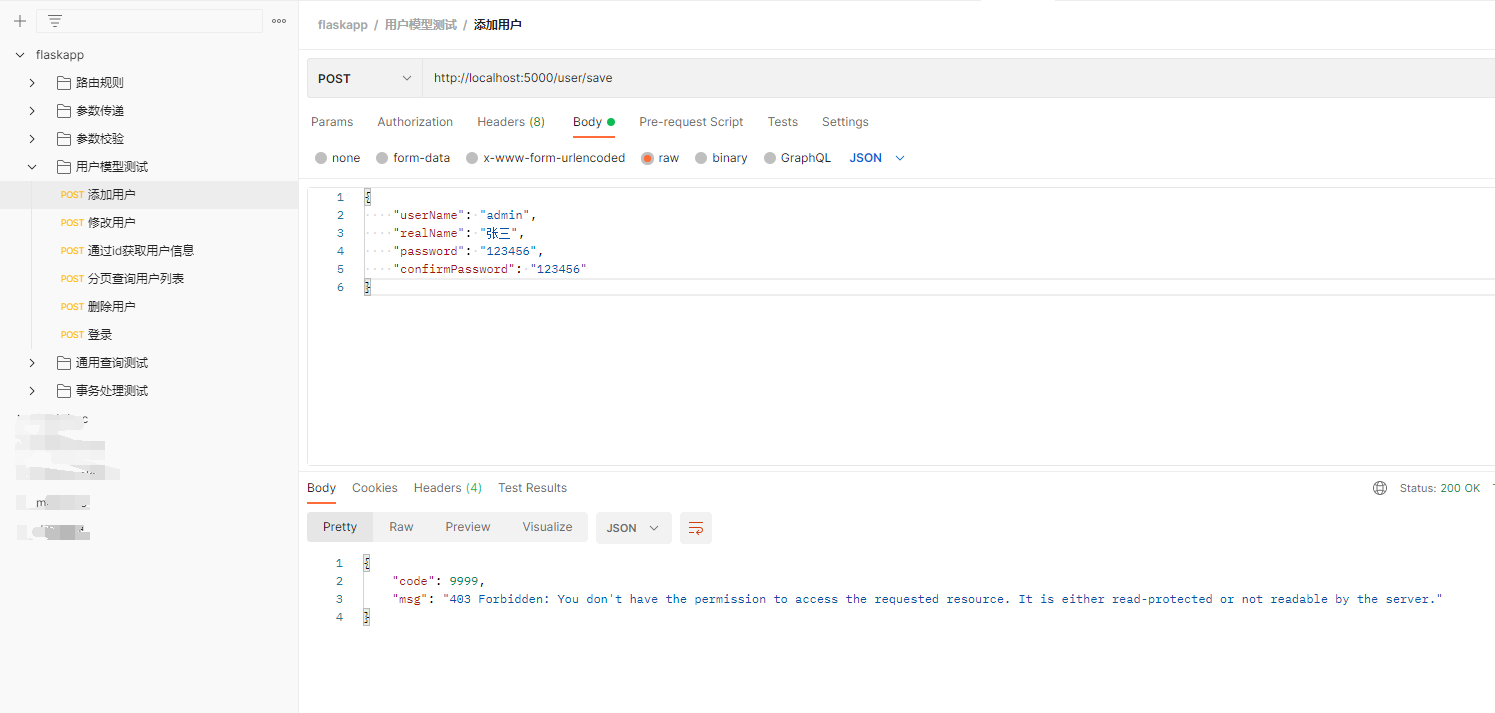
# 修改controllers/user\_controller.py

from flask import Blueprint, request  
  
from controllers import R  
from mauth import HasPerm  
from models.user import User  
from services.user\_service import UserService  
from validators import BasePageForm  
from validators.id\_validator import IdForm, IdsForm  
from validators.user\_validator import UserForm, LoginForm  
  
# 声明一个蓝图  
user = Blueprint('user', \_\_name\_\_, url\_prefix="/user")  
  
# 声明一个用户业务服务  
user\_service = UserService(model=User)  
  
  
@user.route("/get", methods=['POST'])  
@HasPerm(access="user:get", name="通过id获取用户信息")  
def user\_get():  
 """  
 通过id获取用户信息  
 :return:  
 """  
 form = IdForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # 或者直接拿id值 id=form.id.data  
 # u = User.query.filter\_by(id=form.id.data).first()  
 # 通过主键查询  
 u = user\_service.get(form)  
 if u is not None:  
 return R.data(u.to\_dict(camel=True))  
 else:  
 return R.fail("该记录不存在")  
  
  
@user.route("/list", methods=['POST'])  
@HasPerm(access="user:list", name="分页查询用户列表")  
def user\_list():  
 """  
 分页查询用户列表  
 :return:  
 """  
 form = BasePageForm()  
 form.validate\_for\_api()  
 return R.data(user\_service.list(form))  
  
  
@user.route("/save", methods=['POST'])  
@HasPerm(access="user:save", name="添加用户")  
def user\_save():  
 """  
 添加用户  
 :return:  
 """  
 form = UserForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 user\_service.save(form)  
 return R.success("添加用户成功")  
  
  
@user.route("/update", methods=['POST'])  
@HasPerm(access="user:update", name="修改用户")  
def user\_update():  
 """  
 修改用户  
 :return:  
 """  
 form = UserForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 user\_service.update(form)  
 return R.success("修改用户成功")  
  
  
@user.route("/delete", methods=['POST'])  
@HasPerm(access="user:delete", name="删除用户")  
def user\_delete():  
 """  
 删除用户  
 :return:  
 """  
 form = IdsForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 user\_service.delete(form)  
 return R.success("删除用户成功")  
  
  
@user.route("/login", methods=['POST'])  
def user\_login():  
 """  
 登录  
 :return:  
 """  
 form = LoginForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 res = user\_service.login(form.userName.data, form.password.data)  
 return R.data(res)  
  
  
@user.route("/saveBatch", methods=['POST'])  
def user\_save\_batch():  
 """  
 批量插入用户-开启事务  
 :return:  
 """  
 # 复杂的表单校验-wtforms支持不是很好，这里先不校验  
 user\_service.save\_batch(request.get\_json())  
 return R.success("添加成功")  
  
  
@user.route("/saveBatchNoTrans", methods=['POST'])  
def user\_save\_batch\_no\_trans():  
 """  
 批量插入用户-未开启事务  
 :return:  
 """  
 # 复杂的表单校验-wtforms支持不是很好，这里先不校验  
 user\_service.save\_batch\_no\_trans(request.get\_json())  
 return R.success("添加成功")

# 使用Postman接口测试工具访问

## 访问/user/save

权限拦截未放行，抛出403异常。



## 访问/user/get

权限拦截里面已经放行，访问正常

