**8.1权限拦截·Flask实现权限拦截的两种方式**

任何业务系统，都可能涉及到权限问题，如何优雅的去实现权限，这里提供两种常用的方式，大家可以根据具体情况来选择。

# 使用Flask自带的@app.before\_request装饰器

@app.before\_request  
def auth():  
 """  
 权限拦截  
 :return:  
 """  
 # 简单处理一下，非白名单路由，提示登录  
 if request.path not in app.config['WHITE\_LIST']:  
 return R.fail("请先登录")

如上，在前面讲Flask的基础知识的时候，Flask有个请求前装饰器，所有的请求在进入具体的视图方法前，都先调用该方法，这样我们就可以在其上增加权限拦截的逻辑，从而实现权限拦截处理。

# 自定义装饰器

在我们做事务装饰器的时候，也有类似的方式，执行业务方法前，会先执行装饰器方法打开事务，这样我们就可以在里面自定义权限拦截的逻辑。这里先简单的贴代码。

### 定义装饰器

from functools import wraps  
  
  
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  
 pass  
  
 return wrapped\_function

## 使用装饰器

权限装饰器要放在最下面，要不然不生效。

@user.route("/get", methods=['POST'])  
@HasPerm(access="user:get",name="通过id获取用户信息")  
def user\_get():  
 pass  
  
@user.route("/list", methods=['POST'])  
@HasPerm(access="user:list",name="分页查询用户列表")  
def user\_list():  
 pass  
  
@user.route("/save", methods=['POST'])  
@HasPerm(access="user:save",name="添加用户")  
def user\_save():  
 pass  
  
@user.route("/update", methods=['POST'])  
@HasPerm(access="user:update",name="修改用户")  
def user\_update():  
 pass  
  
@user.route("/delete", methods=['POST'])  
@HasPerm(access="user:delete",name="删除用户")  
def user\_delete():  
 pass

通过该方式，我们可以给我们的路由增加一些描述，方便后续解析入库。