**5.1控制层·控制层样例**

控制层即为业务处理的入口，所有的请求都需要经过它，基本的处理流程如下：

客户端发起请求->控制层拿到请求参数->控制层调用处理业务方法->控制层返回处理结果。为了规范化控制层的整个请求入参出参规范，我们需要制作一个控制层的基础样例，方便统一代码风格。这里以user\_controller.py为例。

# 修改controllers/\_\_init\_\_.py

新增通用返回结构封装方法

from flask import jsonify  
  
  
class R:  
 @staticmethod  
 def success(msg="成功"):  
 mDict = {'code': 0, 'msg': msg}  
 return jsonify(mDict)  
  
 @staticmethod  
 def fail(msg, code=9999):  
 mDict = {'code': code, 'msg': msg}  
 return jsonify(mDict)  
  
 @staticmethod  
 def data(data):  
 mDict = {'code': 0, 'msg': "成功", "data": data}  
 return jsonify(mDict)

# 修改controllers/user\_controller.py

from flask import Blueprint  
  
from controllers import R  
  
user = Blueprint('user', \_\_name\_\_, url\_prefix="/user")  
  
  
@user.route("/get", methods=['POST'])  
def user\_get():  
 """  
 通过id获取用户信息  
 :return:  
 """  
 return R.data({  
 "id": 1,  
 "userName": "admin",  
 "userType": 1  
 })  
  
  
@user.route("/list", methods=['POST'])  
def user\_list():  
 """  
 分页查询用户列表  
 :return:  
 """  
 return R.data({  
 "recordCount": 100,  
 "totalPage": 10,  
 "pageSize": 10,  
 "pageNum": 1,  
 "rows": [  
 {  
 "id": 1,  
 "userName": "admin1",  
 "userType": 1  
 },  
 {  
 "id": 2,  
 "userName": "admin2",  
 "userType": 1  
 }  
 ]  
 })  
  
  
@user.route("/save", methods=['POST'])  
def user\_save():  
 """  
 添加用户  
 :return:  
 """  
 return R.success("添加用户成功")  
  
  
@user.route("/update", methods=['POST'])  
def user\_update():  
 """  
 修改用户  
 :return:  
 """  
 return R.success("修改用户成功")  
  
  
@user.route("/delete", methods=['POST'])  
def user\_delete():  
 """  
 删除用户  
 :return:  
 """  
 return R.success("删除用户成功")

# 修改app.py

from flask import Flask, request  
  
from config import getConfig  
from controllers import R  
from controllers.role\_controller import role  
from controllers.user\_controller import user  
from models import db  
  
app = Flask(\_\_name\_\_)  
  
# 注册用户模块  
app.register\_blueprint(user)  
# 注册角色模块  
app.register\_blueprint(role)  
# 从配置对象中加载  
app.config.from\_ｏｂｊｅｃｔ(getConfig())  
  
# 初始化db  
db.init\_app(app)  
  
  
@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")  
  
  
@app.route("/ex/test")  
def ex\_test():  
 """  
 测试其他异常  
 :return:  
 """  
 a = 3 / 0  
 return a  
  
  
@app.route("/db/test")  
def db\_test():  
 """  
 数据库连接测试  
 :return:  
 """  
 cursor = db.session.execute('select \* from t\_user')  
 result = cursor.fetchall()  
 if len(result) > 0:  
 u = result[0]  
 return R.data({  
 "id": u.id,  
 "userName": u.user\_name  
 })  
 return R.fail("无记录")  
  
  
@app.route("/config/test")  
def config\_test():  
 """  
 配置测试  
 :return:  
 """  
 return R.data({  
 'APP\_AUTHOR': app.config['APP\_AUTHOR'],  
 "ENV": app.config['ENV']  
 })  
  
  
@app.errorhandler(404)  
def error\_404(e):  
 """  
 404异常处理  
 :param e:  
 :return:  
 """  
 return R.fail("请求地址不存在")  
  
  
@app.errorhandler(Exception)  
def error(e):  
 """  
 其他异常处理  
 :param e: 异常  
 :return:  
 """  
 return R.fail(str(e))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(host="0.0.0.0")

# 运行Flask服务

flask run

# 使用curl命令访问

## 访问/user/get

curl -X POST http://localhost:5000/user/get

## 访问/user/list

curl -X POST http://localhost:5000/user/list

## 访问/user/save

curl -X POST http://localhost:5000/user/save

## 访问/user/update

curl -X POST http://localhost:5000/user/update

## 访问/user/delete

curl -X POST http://localhost:5000/user/delete

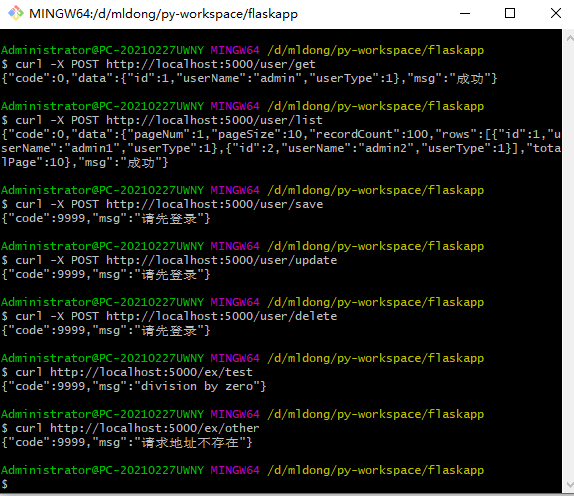
## 访问/ex/test

curl http://localhost:5000/ex/test

## 访问不存在的路由

curl http://localhost:5000/ex/other

访问结果如下：



/user/save、/user/update、/user/delete这个提示是权限拦截的问题