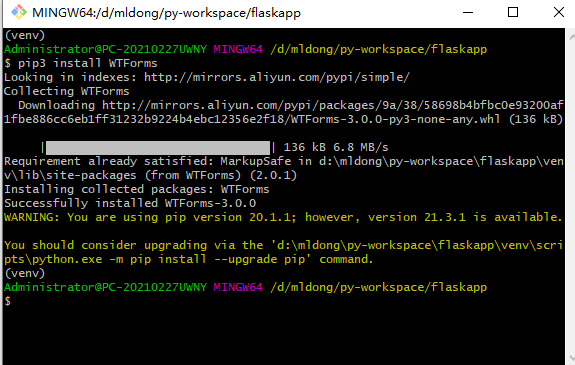
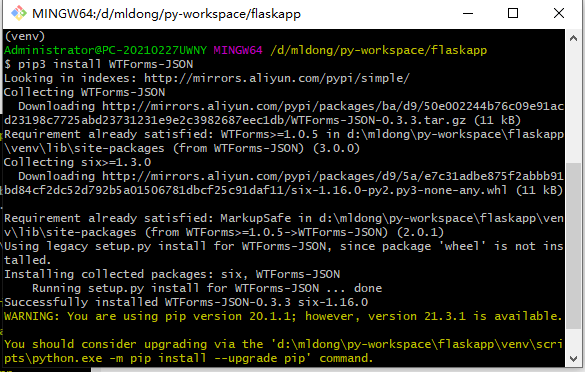
**5.2控制层·参数校验**

客户端提交的参数都是不可信的，所以后端需要对参数进行校验。这里我们为了解决参数校验的问题，一定是要将参数校验的部分抽离出来，按照面向对象的思想，隐藏参数校验的具体过程，交给特定的类去解决。这样，我们在控制层函数中，不会出现冗余的参数校验代码，会使整个控制函数显得简短易读。

# 安装WTForms

# 安装最新版本  
pip3 install WTForms  
# 或指定安装版本  
pip3 install WTForms==3.0.0





# 创建校验模块

mkdir validators

# 新增validators/\_\_init\_\_.py文件

from flask import request  
from wtforms import Form, validators, IntegerField  
  
  
class BaseForm(Form):  
 """  
 校验基类  
 """  
  
 def \_\_init\_\_(self, data=None):  
 if data:  
 super(BaseForm, self).\_\_init\_\_(data=data)  
 else:  
 data = request.get\_json()  
 args = request.args.to\_dict()  
 super(BaseForm, self).\_\_init\_\_(data=data, \*\*args)  
  
 def validate\_for\_api(self):  
 valid = super(BaseForm, self).validate()  
 if not valid:  
 raise Exception(self.errors)  
 return self  
  
  
class BasePageForm(BaseForm):  
 """  
 基础的分页参数类  
 """  
 pageNum = IntegerField("pageNum", [validators.DataRequired(message="页码不能为空")])  
 pageSize = IntegerField("pageSize", [validators.DataRequired(message="每页条数不能为空")])

* 获取请求的参数并完成数据初始化
* 新增校验方法validate\_for\_api，提供外部调用校验

# 新增id与ids校验类validators/id\_validator.py

from wtforms import IntegerField, validators, FieldList  
  
from validators import BaseForm  
  
  
class IdForm(BaseForm):  
 """  
 校验规则：主键只能为>0的整数{"id": 1}  
 """  
 id = IntegerField("id", [validators.DataRequired(message="id不能为空")])  
  
  
class IdsForm(BaseForm):  
 """  
 校验规则：ids集合>0{"ids": [1,3,4]}  
 """  
 ids = FieldList(IntegerField("id",[validators.DataRequired(message="id不能为空")]), min\_entries=1)

# 新增user校验类validator/user\_validator.py

from wtforms import IntegerField, StringField, validators, PasswordField  
  
from validators import BaseForm  
  
  
class UserForm(BaseForm):  
 """  
 用户表单校验类  
 """  
 id = IntegerField()  
 userName = StringField("用户名", [validators.DataRequired(message="用户名不能为空")])  
 realName = StringField("姓名", [validators.DataRequired(message="姓名不能为空")])  
 password = PasswordField("密码", [validators.DataRequired(message="密码不能为空")])  
 confirmPassword = StringField("确认密码", [validators.EqualTo("password", message="两密码不一致")])  
  
 @staticmethod  
 def validate\_userName(form, field):  
 if field.data == "error":  
 raise Exception("自定义方法校验测试")

# 修改controllers/user\_controller.py

from flask import Blueprint  
  
from controllers import R  
from validators import BasePageForm  
from validators.id\_validator import IdForm, IdsForm  
from validators.user\_validator import UserForm  
  
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:  
 """  
 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  
 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:  
 """  
 form = BasePageForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # 可通过form.pageNum.data获取pageNum  
 # 可通过form.pageSize.data获取pageSize  
 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:  
 """  
 form = UserForm()  
 form.validate\_for\_api()  
 # 可通过form.data获取所有提交参数  
 # print(form.data)  
 return R.success("添加用户成功")  
  
  
@user.route("/update", methods=['POST'])  
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)  
 return R.success("修改用户成功")  
  
  
@user.route("/delete", methods=['POST'])  
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)  
 return R.success("删除用户成功")

# 临时注释掉app.py下的权限拦截

后面内容会优化

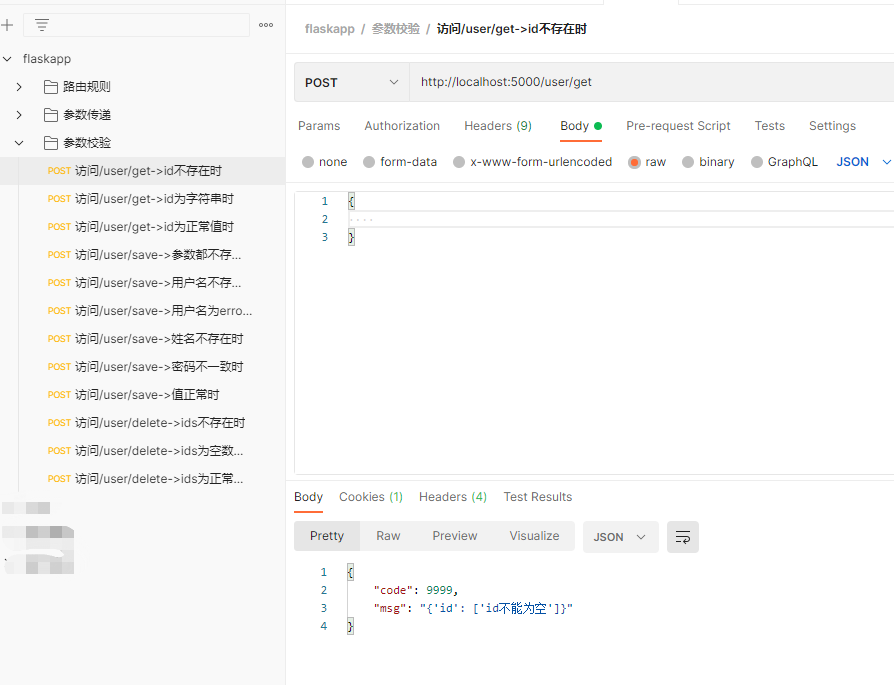
# @app.before\_request  
def auth():  
 """  
 权限拦截  
 :return:  
 """  
 # 简单处理一下，非白名单路由，提示登录  
 if request.path not in app.config['WHITE\_LIST']:  
 return R.fail("请先登录")

# 运行Flask服务

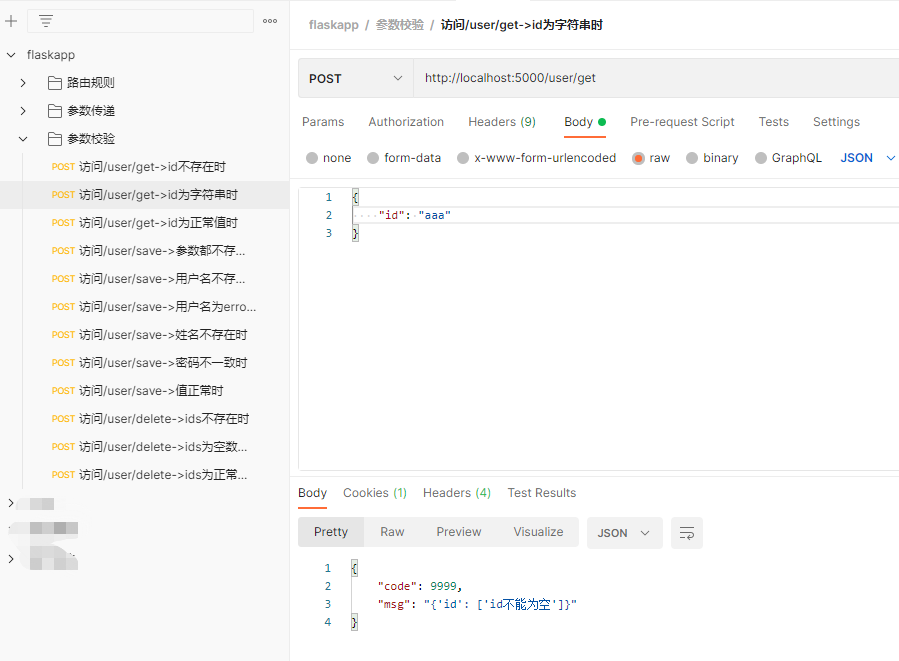
flask run

# 使用Postman接口测试工具访问

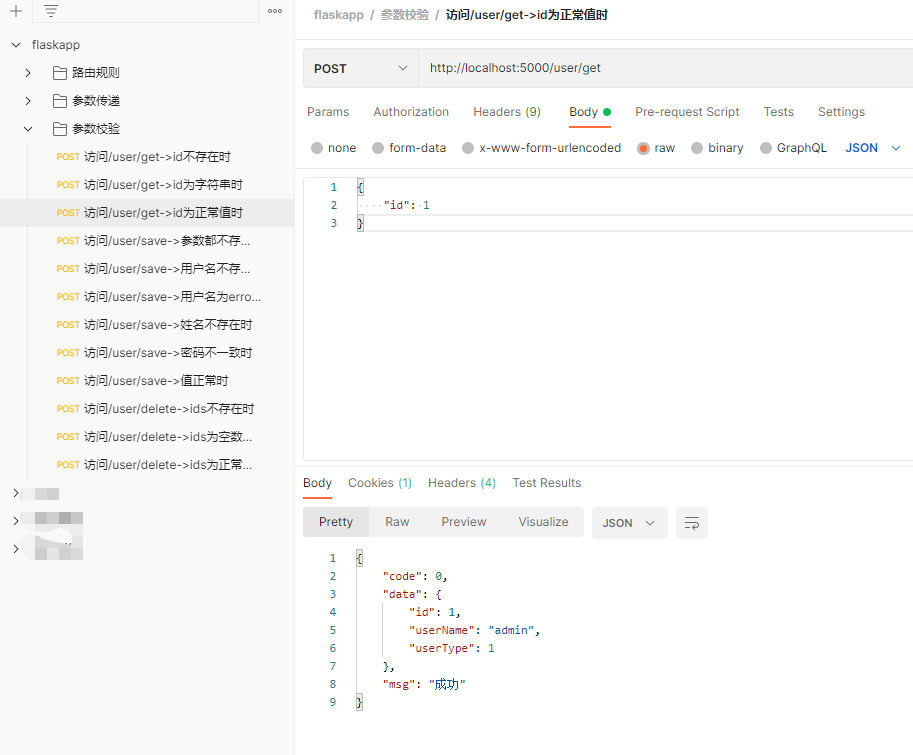
## 访问/user/get->id不存在时



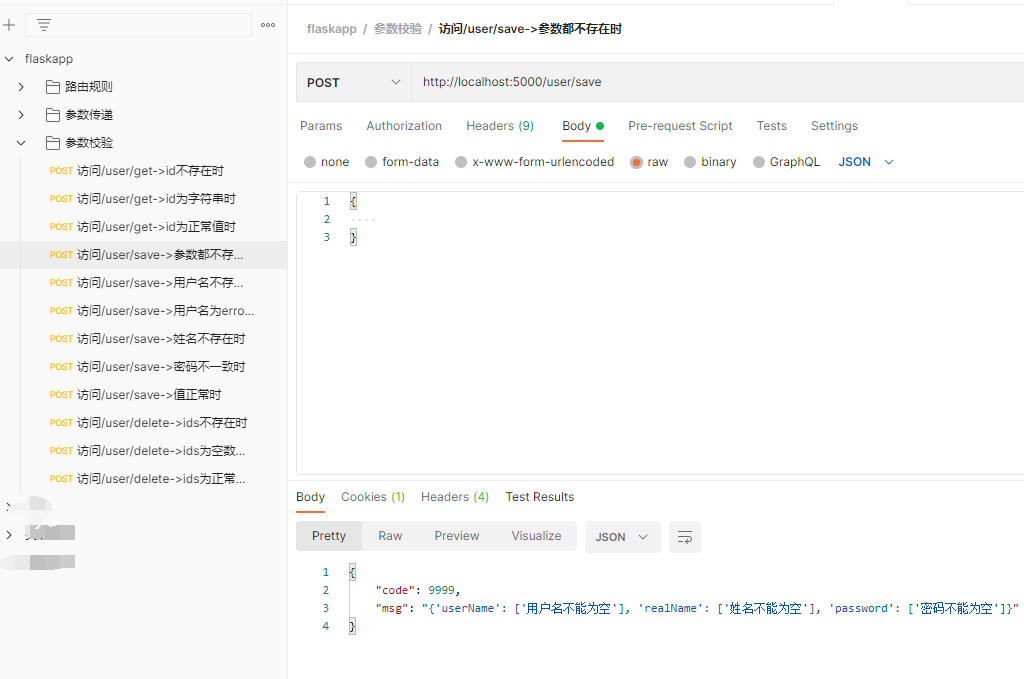
## 访问/user/get->id为字符串时



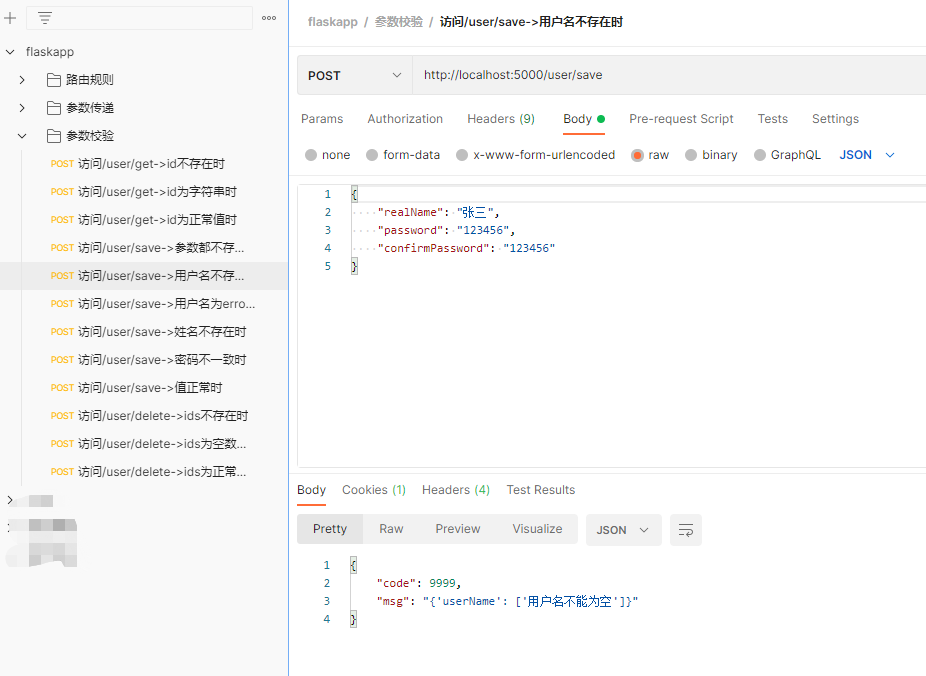
## 访问/user/get->id为正常值时



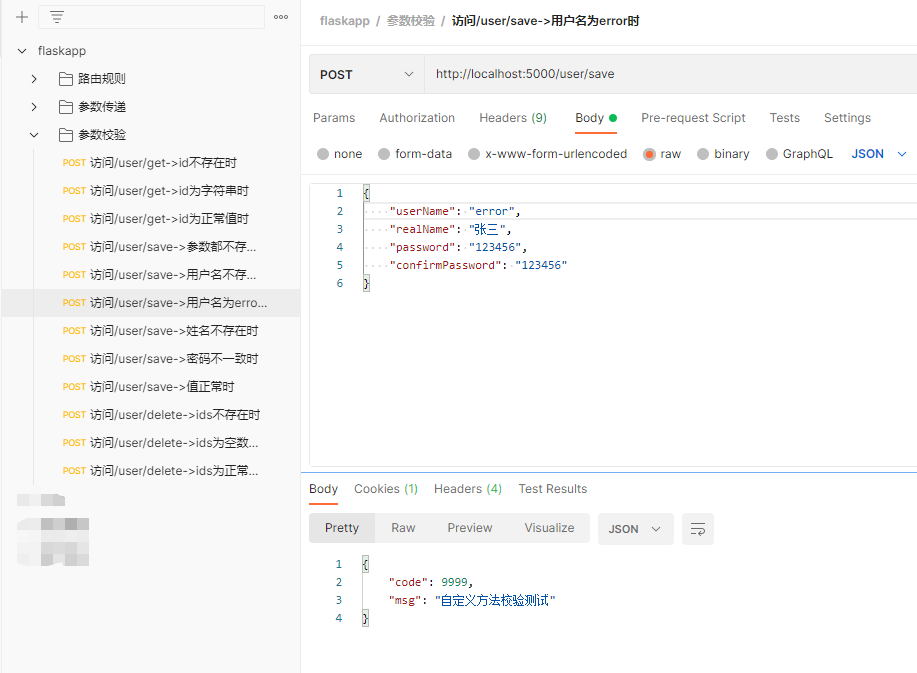
## 访问/user/save->参数都不存在时



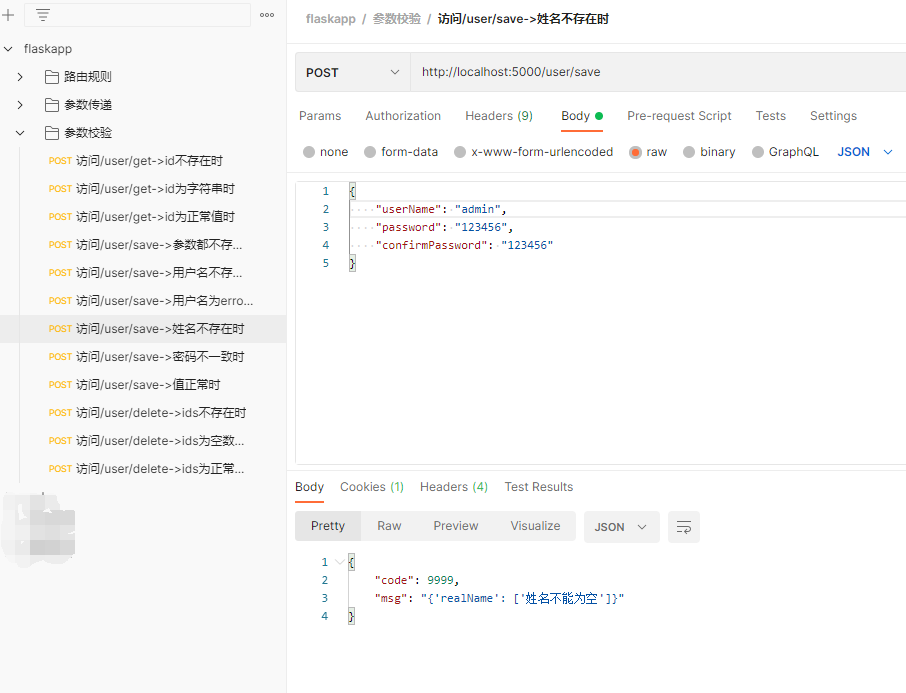
## 访问/user/save->用户名不存在时



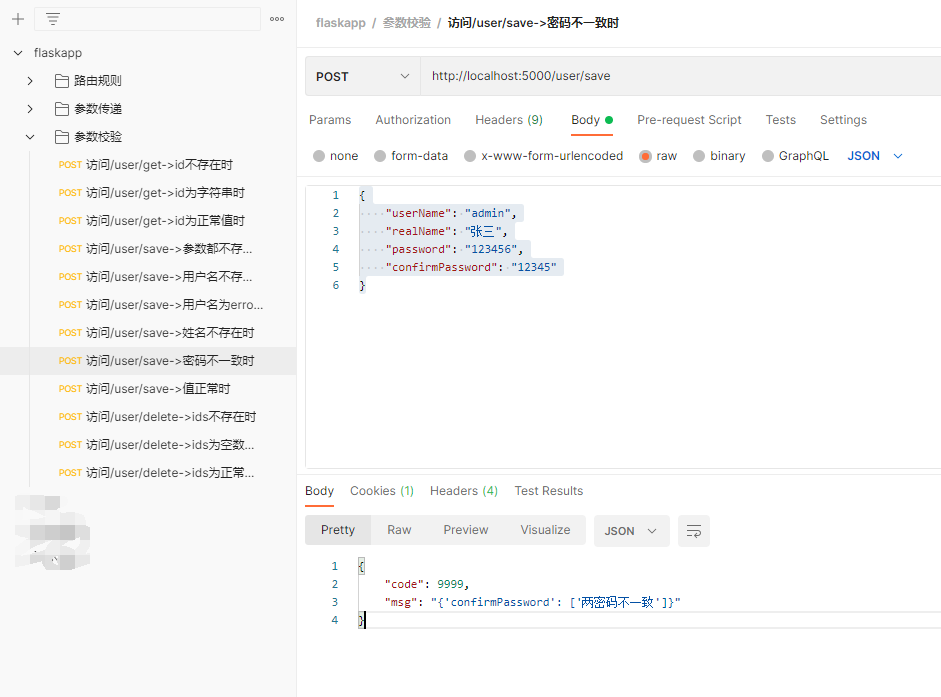
## 访问/user/save->用户名为error时



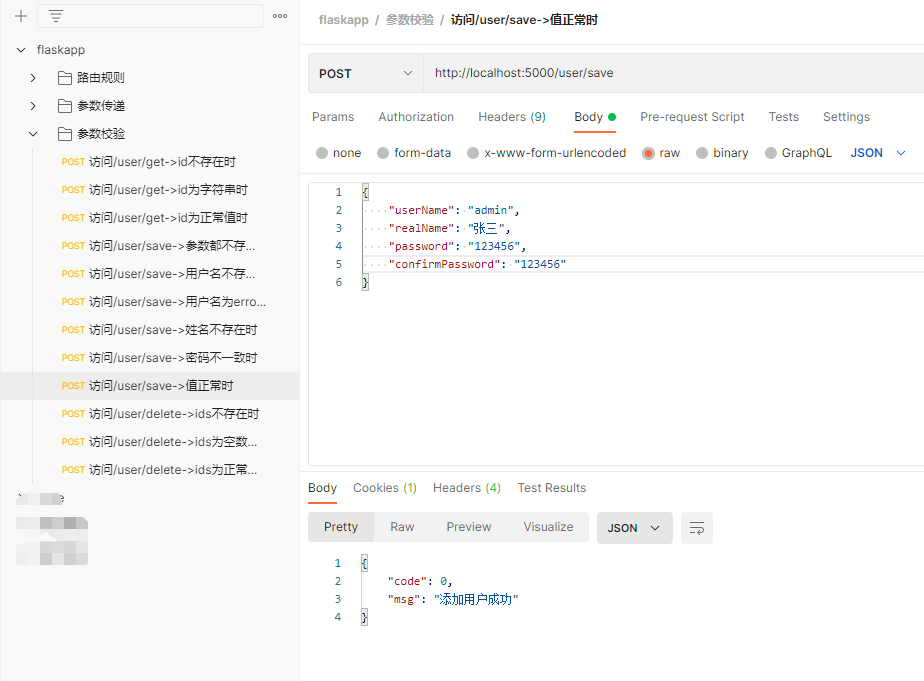
## 访问/user/save->姓名不存在时



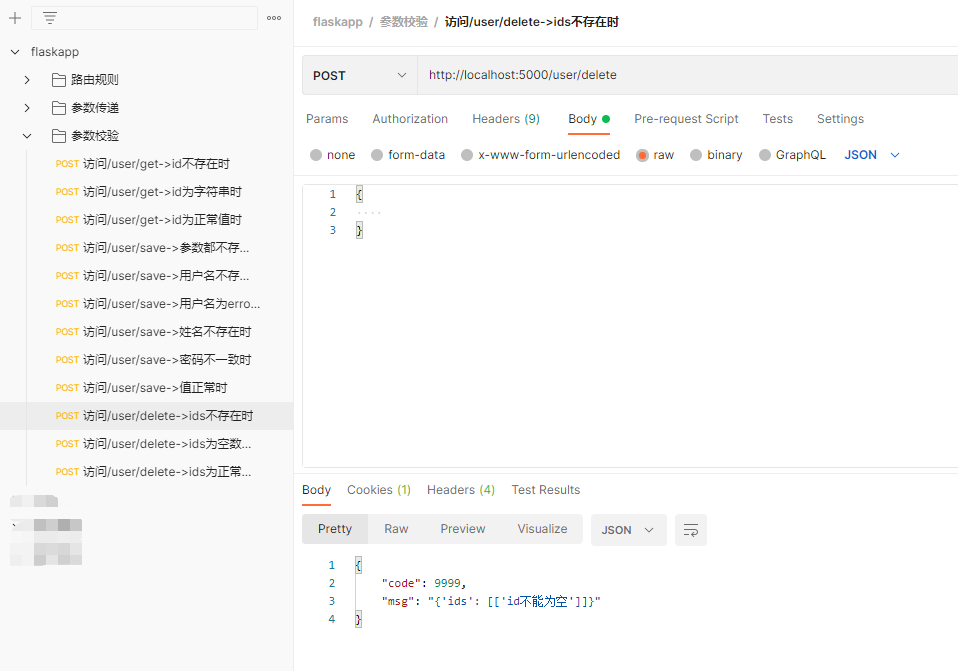
## 访问/user/save->密码不一致时



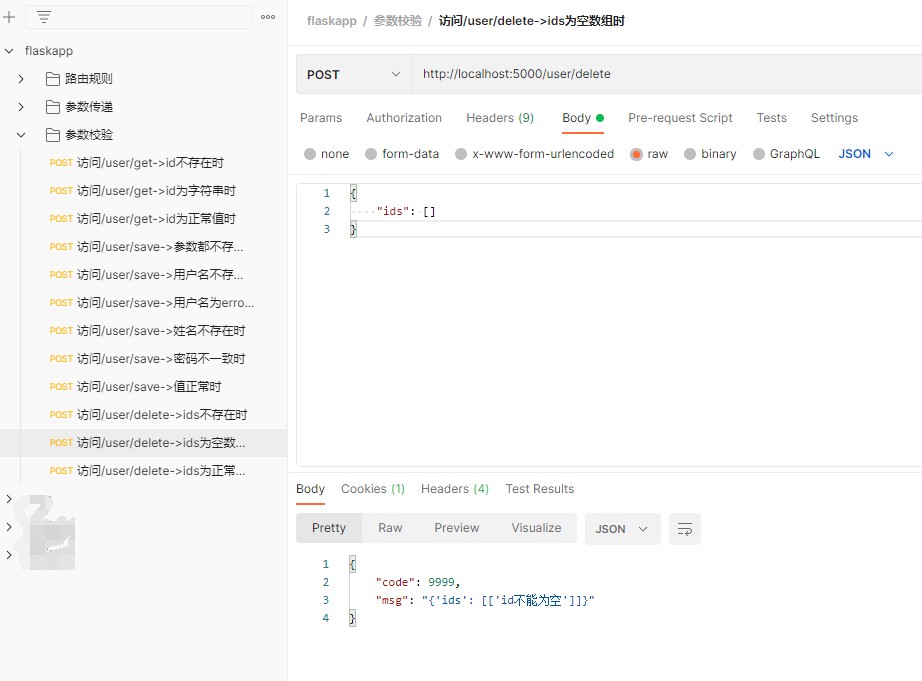
## 访问/user/save->值正常时



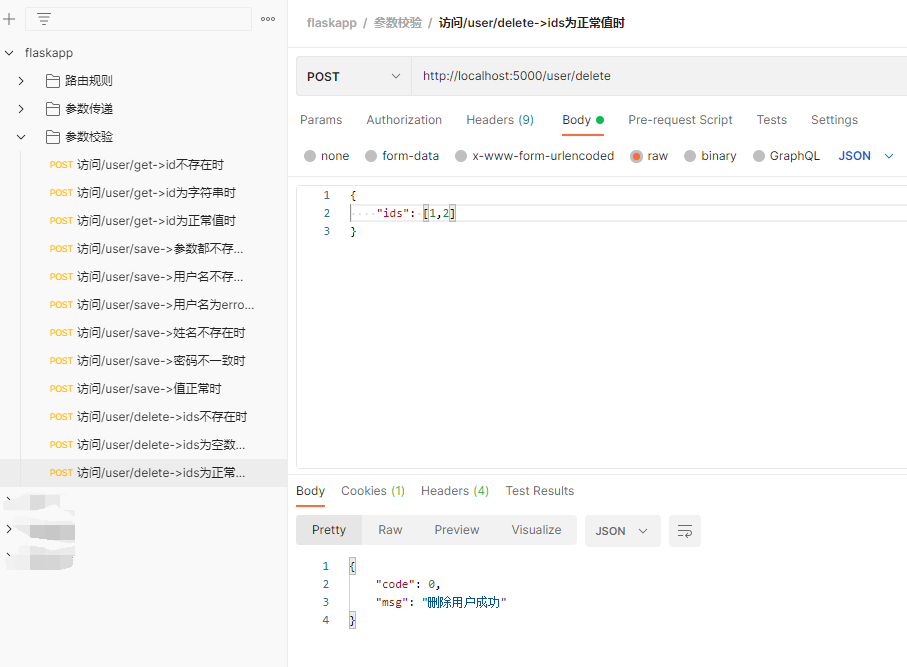
## 访问/user/delete->ids不存在时



## 访问/user/delete->ids为空数组时



## 访问/user/delete->ids为正常值时



# postman导出文件

