**7.1业务逻辑层·业务逻辑层样例**

当遇到复杂的业务时，仅使用控制层去处理业务，可能就不是很方便了。按照面向对象的思想，隐藏业务逻辑的具体过程，交给特定的类去解决。这样，我们在控制层函数中，不会出现冗余的业务代码，会使整个控制函数显得简短易读。同时，业务逻辑上代码可以提供给多个控制层函数调用，更好的做到代码的复用。这里说明一下分层关系。

* controllers

控制层，也叫视图层，用于接收客户端提交的参数，然后返回业务处理接口的，该过程主要做两步操作，第一步是调用参数校验类，进行参数校验，第二步则是调用本文中的业务逻辑类，进行业务处理，最终返回处理结果。

* validators

参数校验层，对客户端提交的参数进行校验。

* services

调用models层，进行具体的业务操作。

* models

数据库操作

# 新增services/\_\_init\_\_.py文件

自定义service基类，主要负责基础的增删改查。

from tools.db\_tool import DbTool  
  
  
class BaseService(ｏｂｊｅｃｔ):  
 """  
 业务逻辑怪基类  
 """  
  
 def \_\_init\_\_(self, db=None, model=None):  
 if db is not None:  
 self.db = db  
 else:  
 from models import db  
 self.db = db  
 if model is not None:  
 self.model = model  
 else:  
 raise Exception("model不能为空")  
  
 def get(self, form):  
 """  
 通过id获取用户信息  
 :param form:  
 :return:  
 """  
 model = self.db.session.query(self.model).get(form.id.data)  
 return model  
  
 def list(self, form):  
 """  
 分页查询用户列表  
 :param form:  
 :return:  
 """  
 # 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  
 # page=self.db.query(User).filter().paginate(form.pageNum.data, form.pageSize.data,False)  
 page = DbTool.filter\_by\_custom(self.model).paginate(form.pageNum.data, form.pageSize.data, False)  
 return self.model.to\_page(page)  
  
 def save(self, form):  
 """  
 添加用户  
 :param form:  
 :return:  
 """  
 model = self.model(\*\*form.data)  
 self.db.session.add(model)  
 self.db.session.commit()  
  
 def update(self, form):  
 """  
 修改用户  
 :param form:  
 :return:  
 """  
 model = self.model(\*\*form.data)  
 self.db.session.query(self.model).filter\_by(id=form.id.data).update(model.to\_dict(camel=False))  
 self.db.session.commit()  
  
 def delete(self, form):  
 """  
 删除用户  
 :param form:  
 :return:  
 """  
 self.db.session.query(self.model).filter(self.model.id.in\_(form.ids.data)).delete()  
 self.db.session.commit()

# 新增services/user\_service.py

业务层类要继承业务基类

from services import BaseService  
  
  
class UserService(BaseService):  
 """  
 用户模块业务处理类  
 """  
 pass

如果业务不一样，可以重写业务方法

from services import BaseService  
  
  
class UserService(BaseService):  
 """  
 用户模块业务处理类  
 """  
 def save(self, form):  
 """  
 重写父类方法  
 :param form:   
 :return:   
 """  
 super().save(form)

当然，也可以自己新增业务方法

from models.user import User  
from services import BaseService  
  
  
class UserService(BaseService):  
 """  
 用户模块业务处理类  
 """  
 def login(self, user\_name, password):  
 """  
 用户密码登录  
 :param user\_name:  
 :param password:  
 :return:  
 """  
 u = self.db.session.query(User).filter(User.user\_name == user\_name).first()  
 if u is None:  
 raise Exception("用户名或密码错误")  
 if u.password != password:  
 raise Exception("用户名或密码错误")  
 res = u.to\_dict(camel=True)  
 del res['password']  
 del res['createTime']  
 del res['updateTime']  
 del res['isDeleted']  
 return res

# 修改validators/user\_validator.py

新增登录校验类

from wtforms import IntegerField, StringField, validators, PasswordField  
  
from validators import BaseForm, BasePageForm  
  
  
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 = PasswordField("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\_user\_name(form, field):  
 if field.data == "error":  
 raise Exception("自定义方法校验测试")  
  
  
  
class LoginForm(BaseForm):  
 """  
 用户名密码登录  
 """  
 userName = StringField("用户名", [validators.DataRequired(message="用户名不能为空")])  
 password = PasswordField("密码", [validators.DataRequired(message="密码不能为空")])

# 修改controllers/user\_controller.py

主要是修改调用业务类方法

from flask import Blueprint  
  
from controllers import R  
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'])  
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'])  
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'])  
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'])  
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'])  
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)

# 运行Flask服务

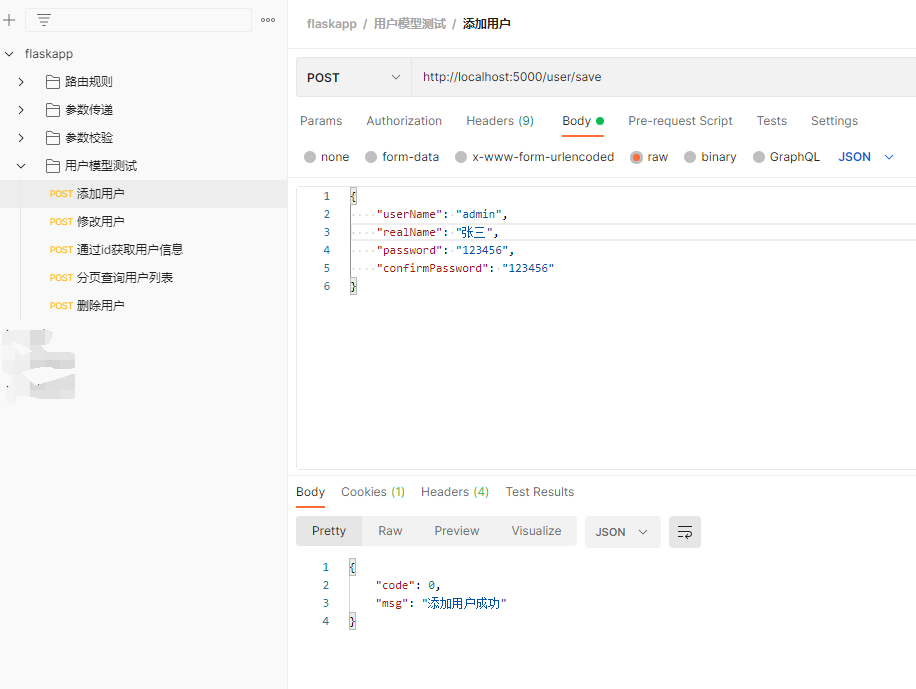
flask run

# 测试前重新建表

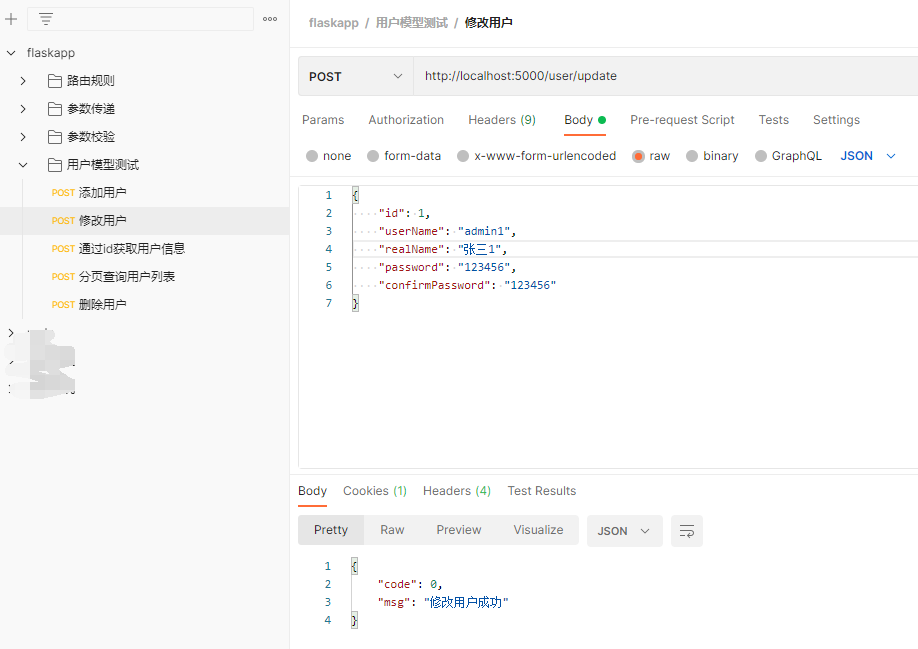
drop table if exists t\_user;  
CREATE TABLE `t\_user` (  
 `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '主键',  
 `user\_name` varchar(32) NOT NULL COMMENT '用户名',  
 `real\_name` varchar(32) NOT NULL COMMENT '姓名',  
 `password` varchar(64) NOT NULL COMMENT '密码',  
 `create\_time` datetime DEFAULT NULL COMMENT '创建时间',  
 `update\_time` datetime DEFAULT NULL COMMENT '更新时间',  
 `is\_deleted` tinyint(1) DEFAULT NULL COMMENT '逻辑删除:0=未删除,1=删除',  
 PRIMARY KEY (`id`),  
 UNIQUE KEY `user\_name` (`user\_name`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户';

# 使用Postman接口测试工具访问

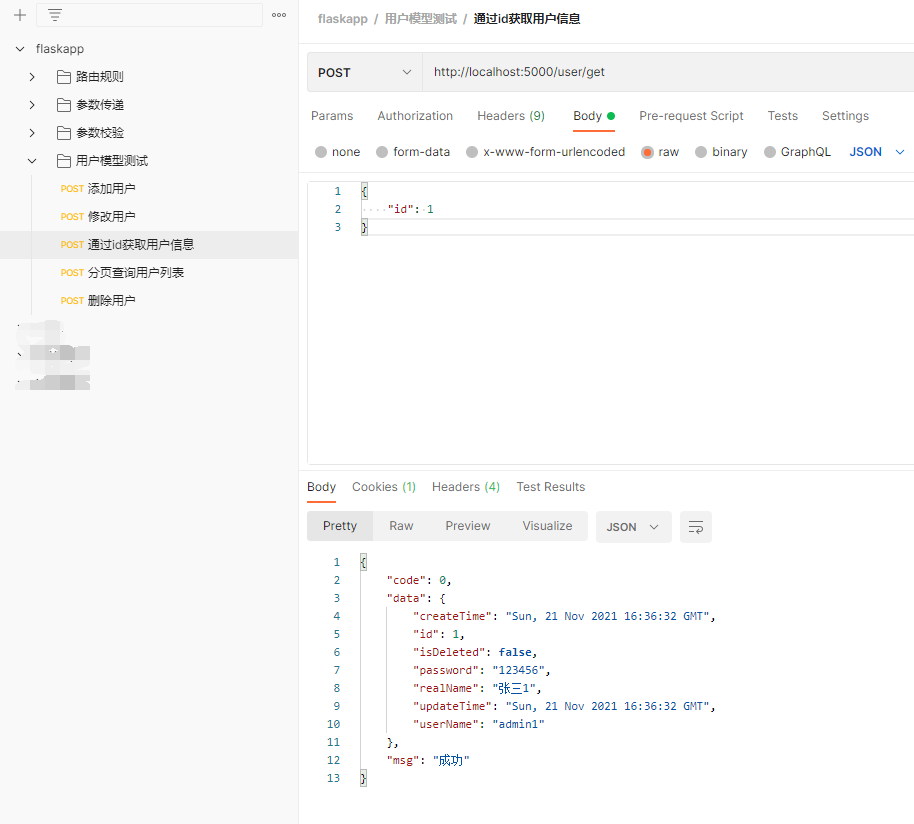
## 访问/user/save添加用户



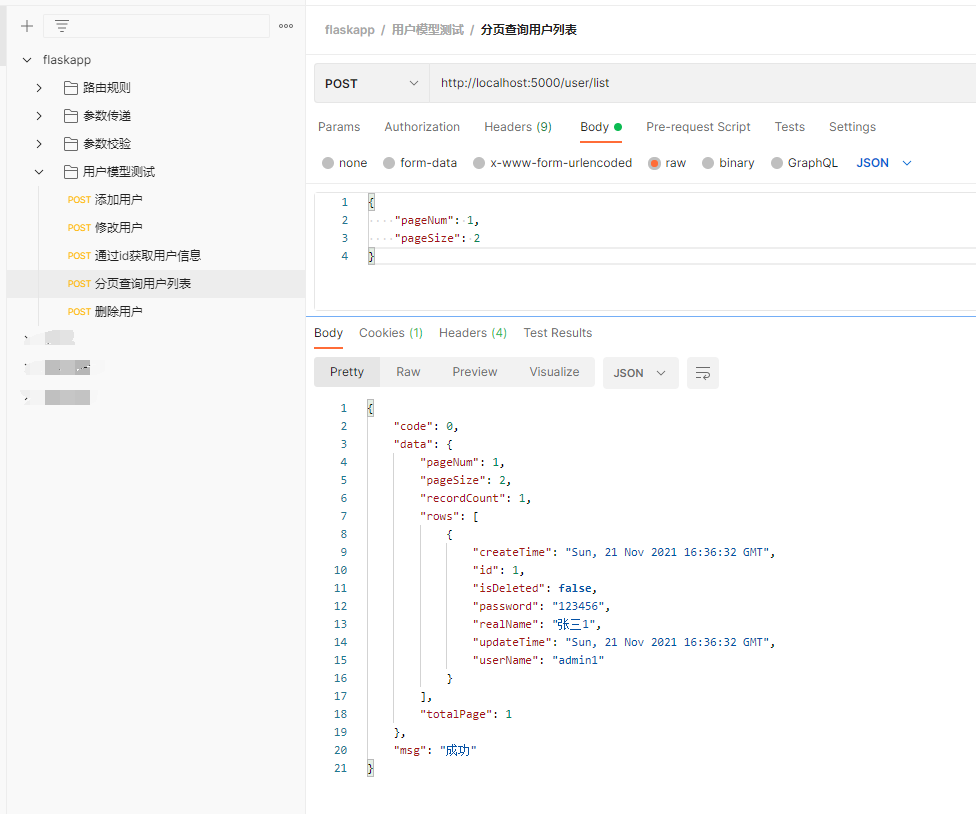
## 访问/user/update修改用户



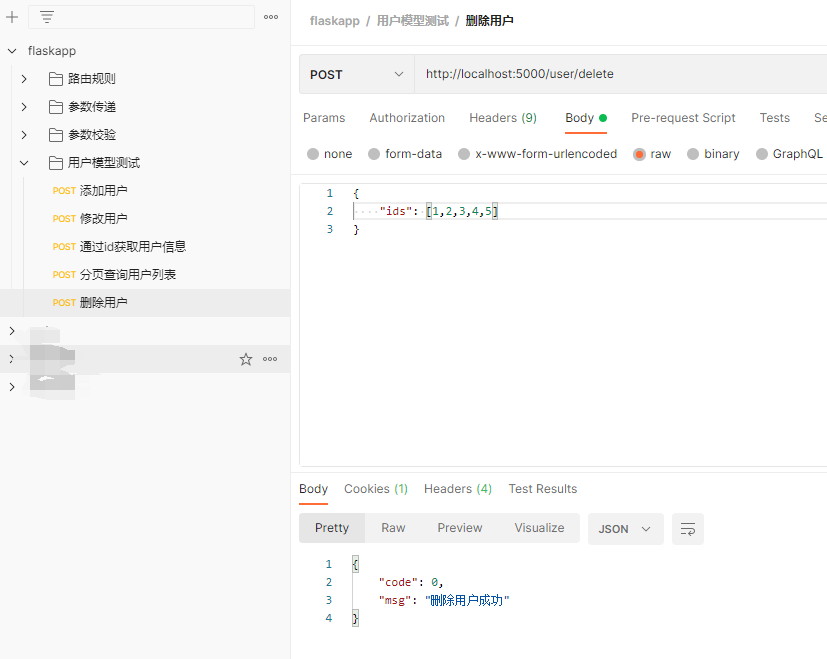
## 访问/user/get通过id获取用户信息



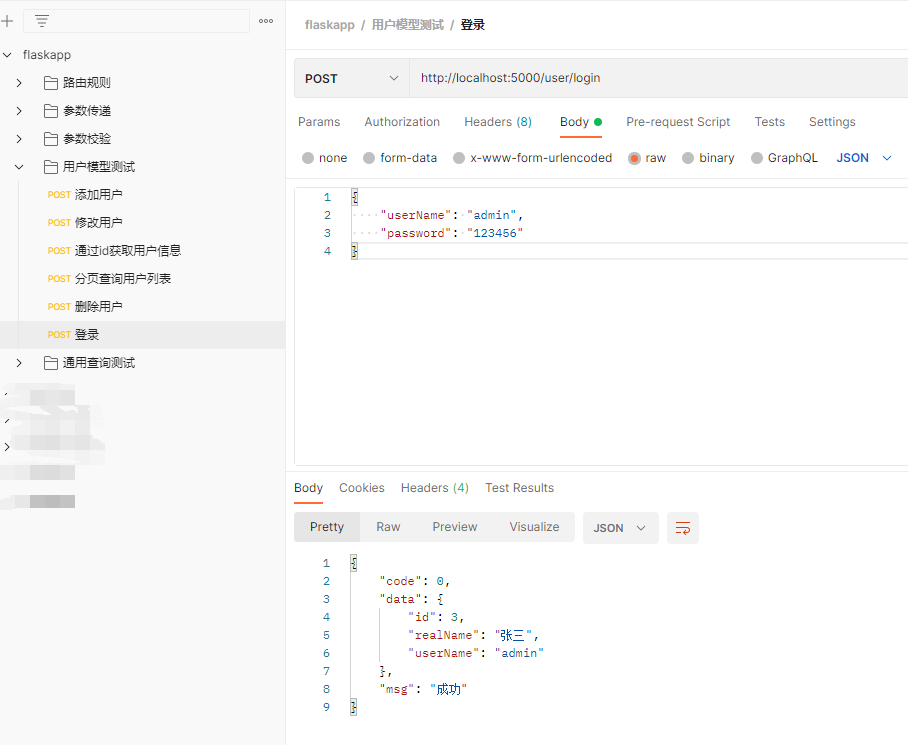
## 访问/user/list分页查询用户列表



## 访问/user/delete删除用户



## 访问/user/login用户登录



# postman导出文件

