**6.3数据库模型层·ORM模型基类优化**

# 现状说明

1. 接口字段为小驼峰
2. 数据库字段为下划线
3. 模型对象不能直接转成JSON字段串，需要转成dict
4. 表单字段也需要转成模型对象
5. 一个个转换不方便

# 优化思路

1. 模型基类新增to\_dict方法，可将属性对象转成字典对象
2. 提供key可为下划线或小驼峰的转换方式
3. 提供可根据表单数据对象构造成模型对象的构造方法

# 新增工具模块

mkdir tools

# 新增tools/\_\_init\_\_.py

def hump\_to\_underline(text):  
 """  
 驼峰转下划线  
 :param text:  
 :return:  
 """  
 res = []  
 for index, char in enumerate(text):  
 if char.isupper() and index != 0:  
 res.append("\_")  
 res.append(char)  
 return ''.join(res).lower()  
  
  
def underline\_to\_hump(text):  
 """  
 下划线转大驼峰  
 :param text:  
 :return:  
 """  
 arr = text.lower().split('\_')  
 res = []  
 for i in arr:  
 res.append(i[0].upper() + i[1:])  
 return ''.join(res)  
  
  
def underline\_to\_camel(text):  
 """  
 下划线转小驼峰  
 :param text:  
 :return:  
 """  
 s = underline\_to\_hump(text)  
 return s[0].lower() + s[1:]

# 修改models/\_\_init\_\_.py

from datetime import datetime  
  
from flask\_sqlalchemy import SQLAlchemy  
  
from tools import underline\_to\_camel, hump\_to\_underline  
  
db = SQLAlchemy()  
  
  
class BaseModel(db.Model):  
 """  
 普通模型基类  
 """  
 \_\_abstract\_\_ = True  
 id = db.Column(db.BigInteger, primary\_key=True, comment="主键")  
 create\_time = db.Column(db.DateTime, name="create\_time", default=datetime.now, comment="创建时间")  
 update\_time = db.Column(db.DateTime, name="update\_time", default=datetime.now, onupdate=datetime.now,  
 comment="更新时间")  
 is\_deleted = db.Column(db.Boolean, name="is\_deleted", default=False, comment="逻辑删除:0=未删除,1=删除")  
  
 def \_\_init\_\_(self, \*\*kwargs):  
 """  
 构造函数，将dict=>obj  
 :param kwargs: 校验表单的\*\*form.data  
 """  
 for (key, value) in kwargs.items():  
 underline\_key = hump\_to\_underline(key)  
 if hasattr(self, underline\_key):  
 self.\_\_dict\_\_.update({  
 underline\_key: value  
 })  
  
 def to\_dict(self, camel=True):  
 """  
 对象转成dict  
 :param camel: 是否转成小驼峰，默认True  
 :return:  
 """  
 m\_dict = {}  
 if self is not None:  
 for c in self.\_\_table\_\_.columns:  
 key = c.name  
 if hasattr(self, key):  
 value = getattr(self, key)  
 if value is not None:  
 if camel:  
 m\_dict[underline\_to\_camel(key)] = value  
 else:  
 m\_dict[key] = value  
  
 return m\_dict  
  
 @staticmethod  
 def to\_page(page):  
 # print(page.page) # 当前页码-从1开始  
 # print(page.per\_page) # 每页大小  
 # print(page.total) # 总记录数  
 # print(page.items) # 数据集  
 if page is None:  
 return None  
 rows = []  
 for u in page.items:  
 rows.append(u.to\_dict(camel=True))  
 return {  
 "recordCount": page.total,  
 "totalPage": int((page.total - 1) / page.per\_page) + 1,  
 "pageSize": page.per\_page,  
 "pageNum": page.page,  
 "rows": rows  
 }  
  
  
class TreeModel(BaseModel):  
 """  
 树型表模型基类  
 """  
 \_\_abstract\_\_ = True  
 parent\_id = db.Column(db.BigInteger, primary\_key=True, default=0, comment="父ID")  
 name = db.Column(db.String(64), name="name", nullable=False, comment="名称")  
 sort = db.Column(db.BigInteger, name="sort", default=99, nullable=False,  
 comment="排序")

# 修改controllers/user\_controller.py

from flask import Blueprint  
  
from controllers import R  
from models import db  
from models.user import User  
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  
 # u = User.query.filter\_by(id=form.id.data).first()  
 # 通过主键查询  
 u = User.query.get(form.id.data)  
 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()  
 # 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=User.query.filter().paginate(form.pageNum.data, form.pageSize.data,False)  
 return R.data(User.to\_page(page))  
  
  
@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)  
 u = User(\*\*form.data)  
 db.session.add(u)  
 db.session.commit()  
 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)  
 u = User(\*\*form.data)  
 User.query.filter\_by(id=form.id.data).update(u.to\_dict(camel=False))  
 db.session.commit()  
 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.query.filter(User.id.in\_(form.ids.data)).delete()  
 db.session.commit()  
 return R.success("删除用户成功")

# 运行Flask服务

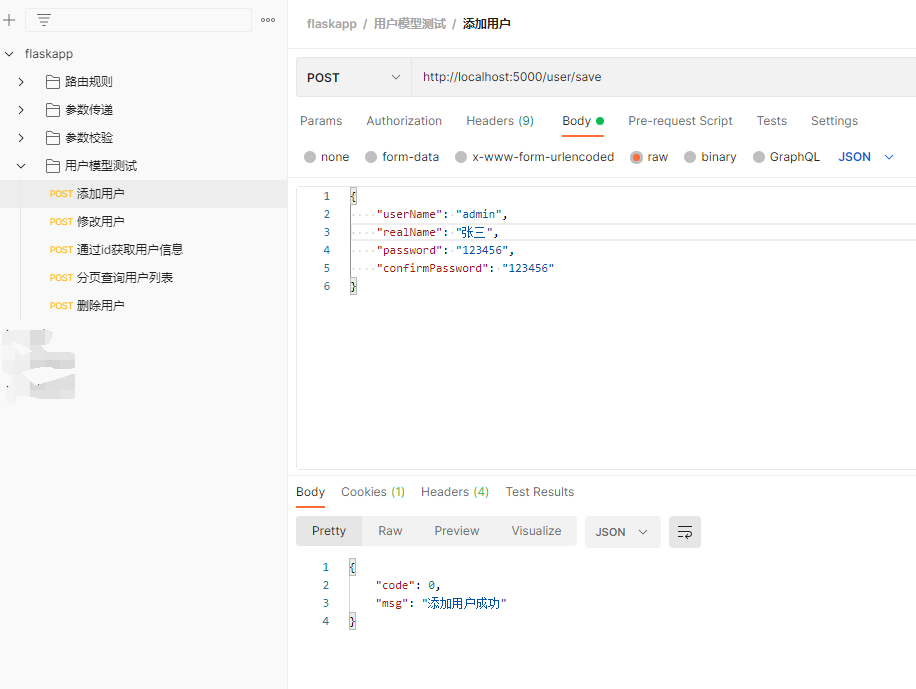
flask run

# 测试前重新建表

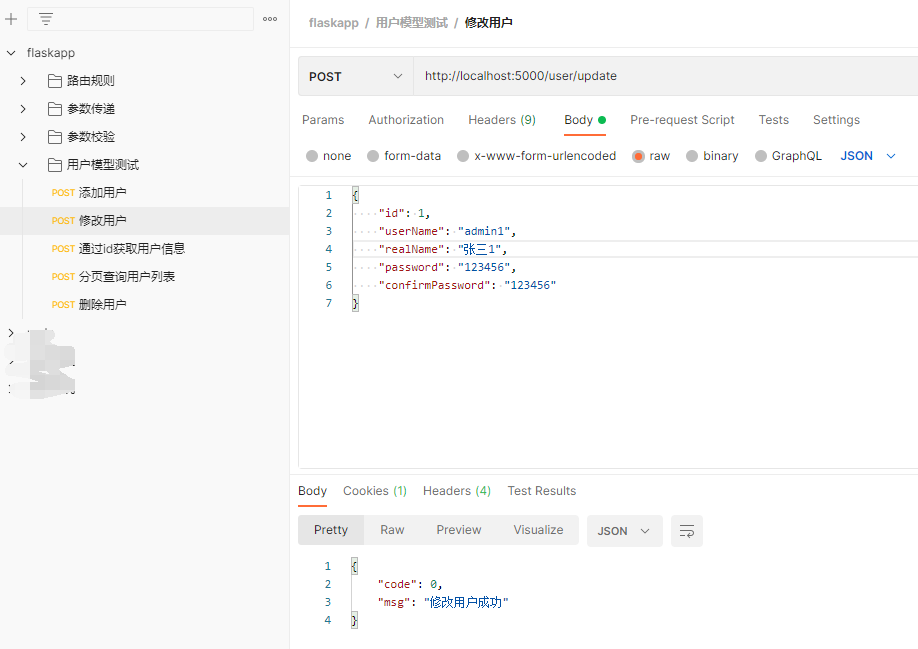
drop table if exists t\_user;  
CREATE TABLE `t\_user` (  
 `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '主键',  
 `user\_name` varchar(32) NOT NULL COMMENT '用户名',  
 `real\_name` varchar(32) NOT NULL COMMENT '姓名',  
 `password` varchar(64) NOT NULL COMMENT '密码',  
 `create\_time` datetime DEFAULT 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

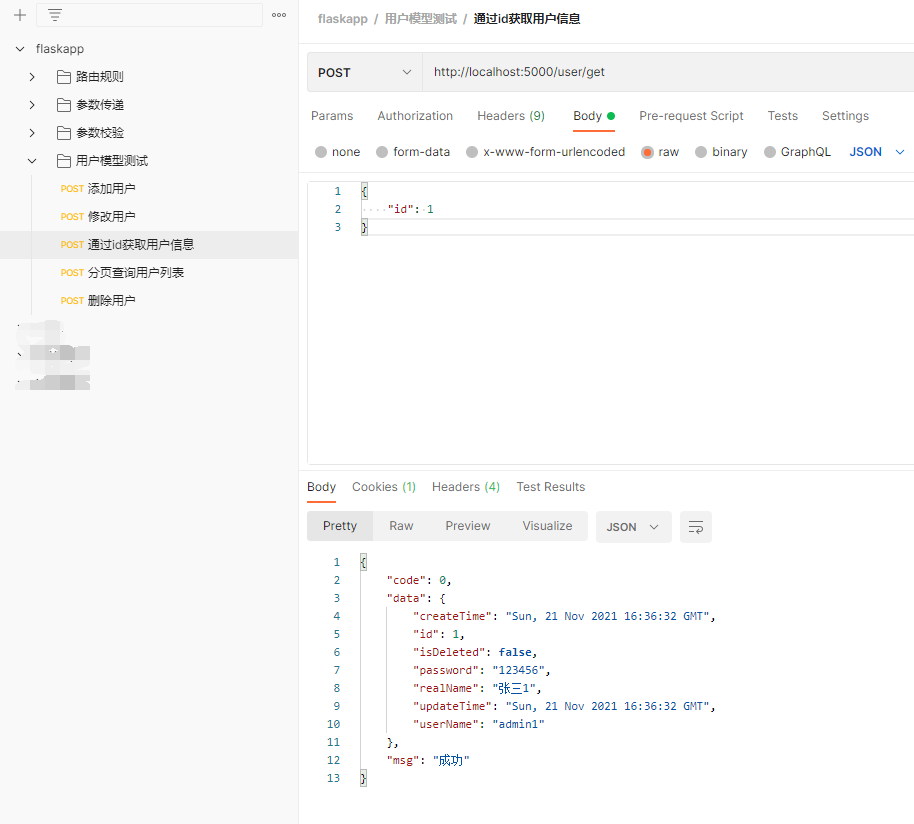
## 访问/user/save添加用户



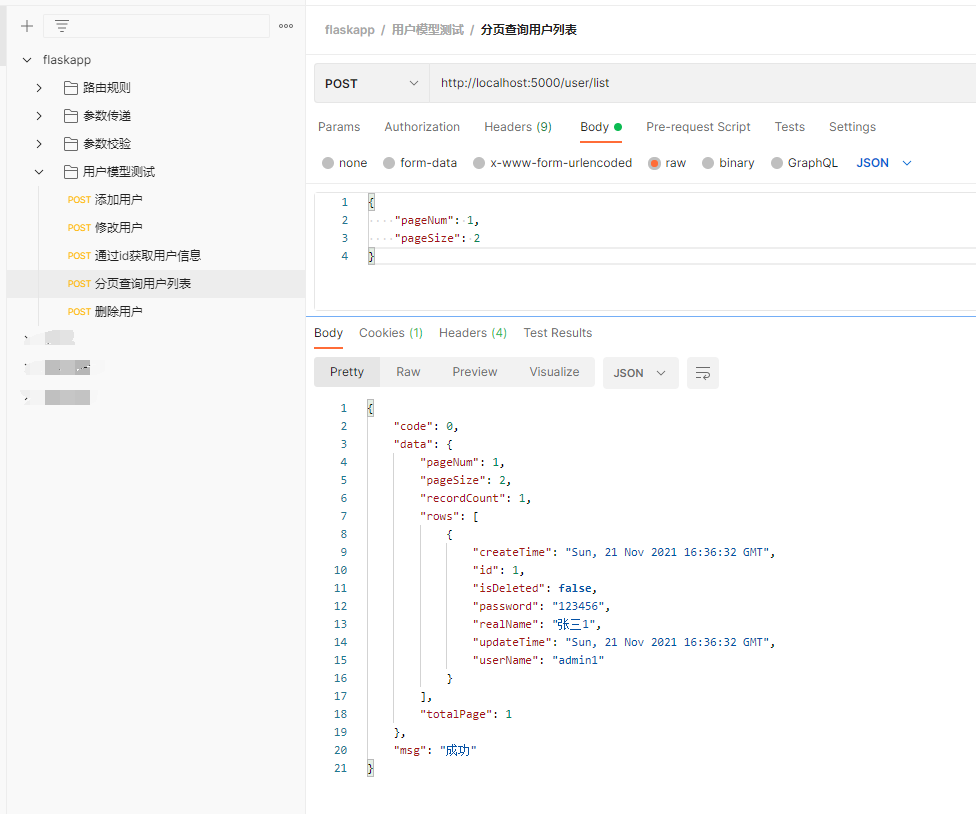
## 访问/user/update修改用户



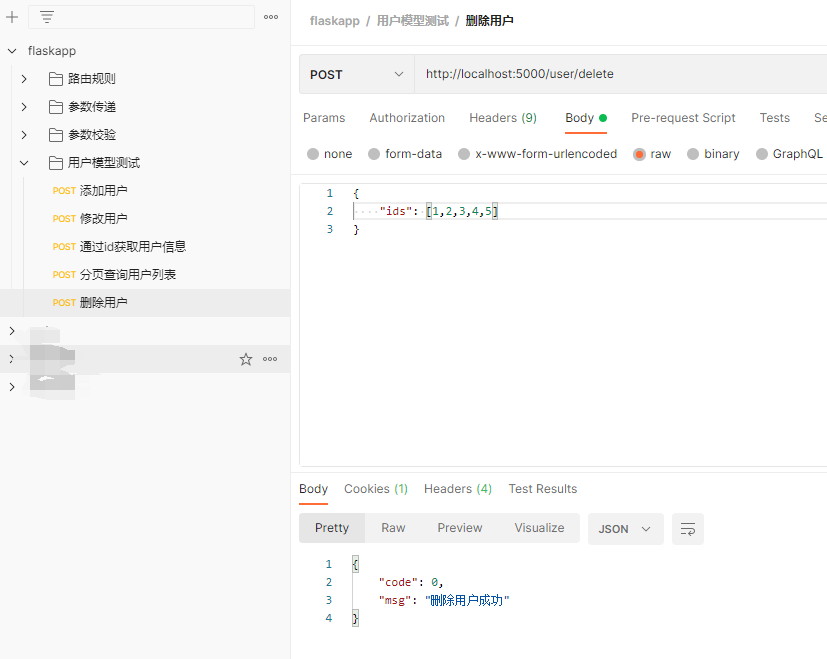
## 访问/user/get通过id获取用户信息



## 访问/user/list分页查询用户列表



## 访问/user/delete删除用户



# postman导出文件

