**6.4数据库模型层·通用查询设计与实现**

一般来说所有的系统都离不开查询，常规的做法都是前端传实体属性，后端通过实体的属性作为条件进行查询。此种方式会使用到sqlalchemy的模型方法动态构建查询条件。在遇到查询需求不是很明朗需要经常改动的情况下，可能并不是很方便。本工程化在查询上使用了一种比较特殊的方式，就是定义一个通用的查询规范，由前端根据需要，调整查询条件。

# 简单查询分析

## 主键查询

select \* from t\_user where id = ?

## 等于

select \* from t\_user where user\_name = ?

## 不等于

select \* from t\_user where user\_name <> ?

## 大于

select \* from t\_user where id > ?

## 大于等于

select \* from t\_user where id >= ?

## 小于

select \* from t\_user where id < ?

## 小于等于

select \* from t\_user where id <= ?

## 区间范围

select \* from t\_user where id between ? and ?

## 非区间范围

select \* from t\_user where id not between ? and ?

## 模糊(全/左/右)

select \* from t\_user where user\_name like ?

## 值集合

select \* from t\_user where id in (?,?,?)

## 非值集合

select \* from t\_user where id not in (?,?,?)

## 组合查询(and)

select \* from t\_user where user\_name = ? and id between ? and ?

# 操作说明

|  |  |
| --- | --- |
| 操作名 | 说明 |
| EQ | 等于= |
| NE | 不等于<> |
| GT | 大于> |
| GE | 大于等于>= |
| LT | 小于< |
| LE | 小于等于<= |
| BT | between 值1 and 值2 |
| LIKE | like '%值%' |
| NLIKE | not like '%值%' |
| LLIKE | like '%abc' |
| RLIKE | like 'abc%' |
| IN | in(值1,值2) |
| NIN | not in(值1,值2) |

# 入参设计

{  
 "pageNum": 1,  
 "pageSize": 15,  
 "m\_LIKE\_userName": "admin",  
 "m\_BT\_createTime": ["2020-01-01","2020-06-06"]  
}

无表别名：m\_EQ\_userName

有表别名：m\_t\_EQ\_userName

# 新增tools/db\_tool.py工具

from flask import request  
  
from models import db  
from tools import hump\_to\_underline  
  
  
class DbTool:  
 """  
 db工具  
 """  
 @staticmethod  
 def get\_condition\_params():  
 """  
 获取自定义查询参数  
 :return:  
 """  
 data = request.get\_json()  
 res = []  
 for (key, value) in data.items():  
 if key.startswith("m\_"):  
 arr = key.split("\_")  
 if len(arr) == 3:  
 res.append({  
 "table": None,  
 "op": arr[1],  
 "key": arr[2],  
 "value": value  
 })  
 elif len(arr) == 4:  
 res.append({  
 "table": arr[1],  
 "op": arr[2],  
 "key": arr[3],  
 "value": value  
 })  
 return res  
  
 @staticmethod  
 def filter\_by\_custom(model):  
 """  
 单表-自定义查询条件  
 :param model:  
 :return:  
 """  
 conditions = DbTool.get\_condition\_params()  
 q = db.session.query(model)  
 for item in conditions:  
 key = hump\_to\_underline(item.get('key'))  
 if not hasattr(model, key):  
 continue  
 op = item.get('op', 'EQ')  
 value = item.get('value')  
 if op == 'EQ':  
 q = q.filter(getattr(model, key) == value)  
 elif op == 'NE':  
 q = q.filter(getattr(model, key) != value)  
 elif op == 'GT':  
 q = q.filter(getattr(model, key) > value)  
 elif op == 'GE':  
 q = q.filter(getattr(model, key) >= value)  
 elif op == 'LT':  
 q = q.filter(getattr(model, key) < value)  
 elif op == 'LE':  
 q = q.filter(getattr(model, key) <= value)  
 elif op == 'BT':  
 if isinstance(value,list) and len(value) == 2:  
 q = q.filter(getattr(model, key).between(value[0], value[1]))  
 elif op == 'LIKE':  
 q = q.filter(getattr(model, key).like('%' + value + '%'))  
 elif op == 'NLIKE':  
 q = q.filter(getattr(model, key).notlike('%' + value + '%'))  
 elif op == 'LLIKE':  
 q = q.filter(getattr(model, key).like('%' + value))  
 elif op == 'RLIKE':  
 q = q.filter(getattr(model, key).like(value + '%'))  
 elif op == 'IN':  
 if isinstance(value, list):  
 q = q.filter(getattr(model, key).in\_(value))  
 elif op == 'NIN':  
 if isinstance(value, list):  
 q = q.filter(getattr(model, key).notin\_(value))  
 return q

# 修改controllers/user\_controller.py

主要修改分页查询接口

page = User.query.filter().paginate(form.pageNum.data, form.pageSize.data,False)  
# 修改为  
page = DbTool.filter\_by\_custom(User).paginate(form.pageNum.data, form.pageSize.data, False)

完整代码如下：

from flask import Blueprint  
  
from controllers import R  
from models import db  
from tools.db\_tool import DbTool  
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)  
 page = DbTool.filter\_by\_custom(User).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

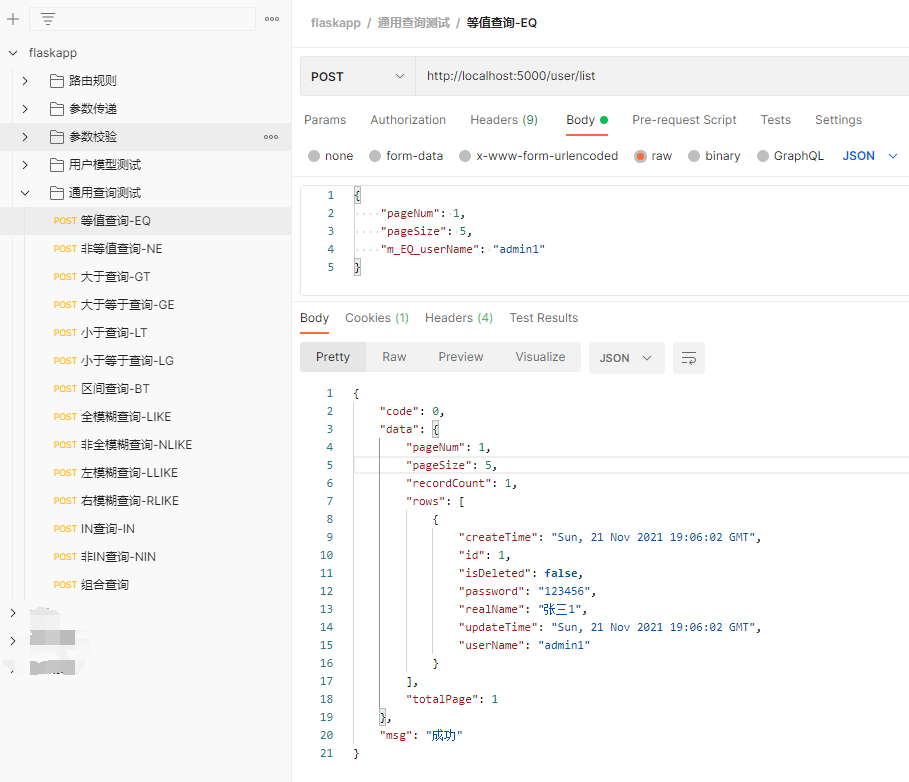
flask run

# 测试前重新创建表并导入数据

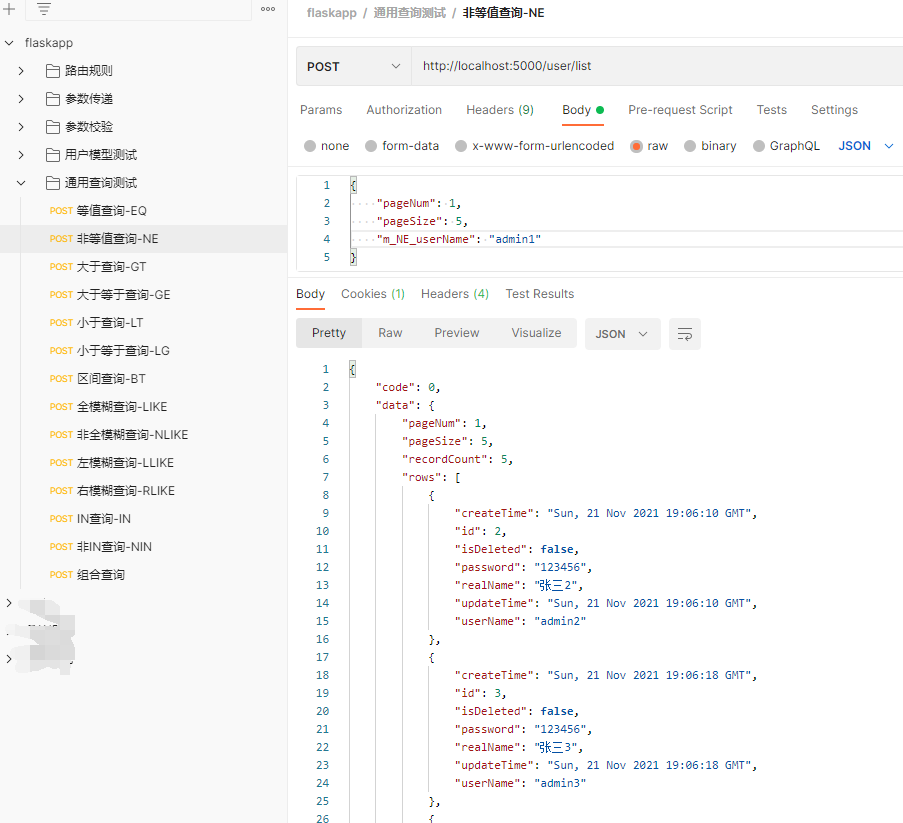
DROP TABLE IF EXISTS `t\_user`;  
CREATE TABLE `t\_user` (  
 `id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '主键',  
 `user\_name` varchar(32) NOT NULL COMMENT '用户名',  
 `real\_name` varchar(32) NOT NULL COMMENT '姓名',  
 `password` varchar(64) NOT NULL COMMENT '密码',  
 `create\_time` datetime DEFAULT 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';  
  
-- ----------------------------  
-- Records of t\_user  
-- ----------------------------  
INSERT INTO `t\_user` VALUES ('1', 'admin1', '张三1', '123456', '2021-11-21 19:06:02', '2021-11-21 19:06:02', '0');  
INSERT INTO `t\_user` VALUES ('2', 'admin2', '张三2', '123456', '2021-11-21 19:06:10', '2021-11-21 19:06:10', '0');  
INSERT INTO `t\_user` VALUES ('3', 'admin3', '张三3', '123456', '2021-11-21 19:06:18', '2021-11-21 19:06:18', '0');  
INSERT INTO `t\_user` VALUES ('4', 'demo1', '张三3', '123456', '2021-11-21 19:06:30', '2021-11-21 19:06:30', '0');  
INSERT INTO `t\_user` VALUES ('5', 'demo2', '张三3', '123456', '2021-11-21 19:06:34', '2021-11-21 19:06:34', '0');  
INSERT INTO `t\_user` VALUES ('6', 'demo3', '张三3', '123456', '2021-11-21 19:06:38', '2021-11-21 19:06:38', '0');

# 使用Postman接口测试工具访问

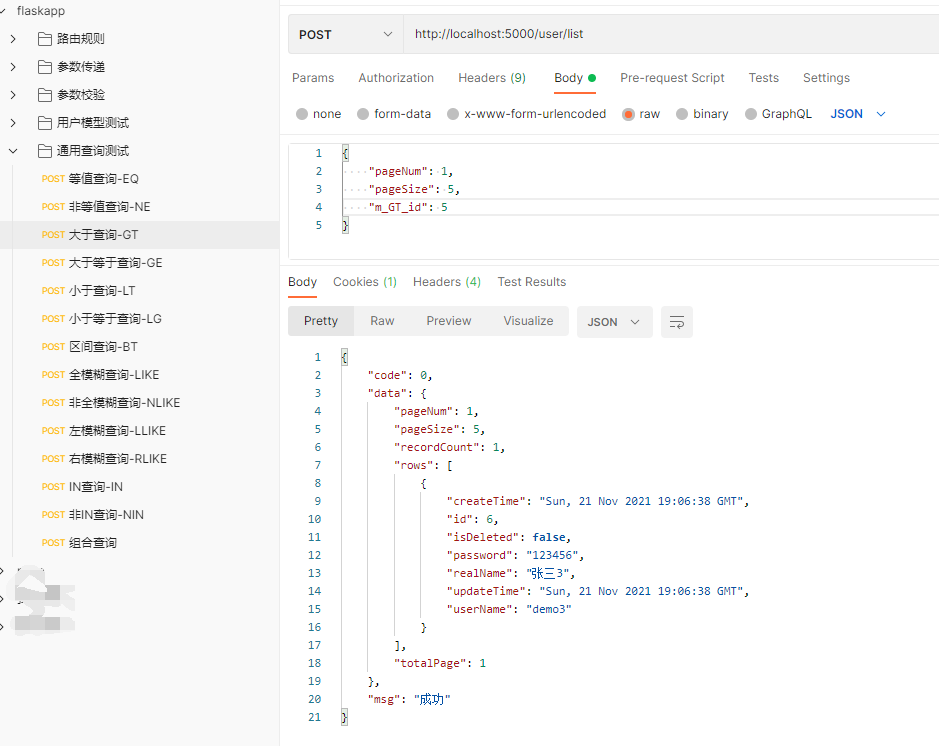
## 等值查询-EQ



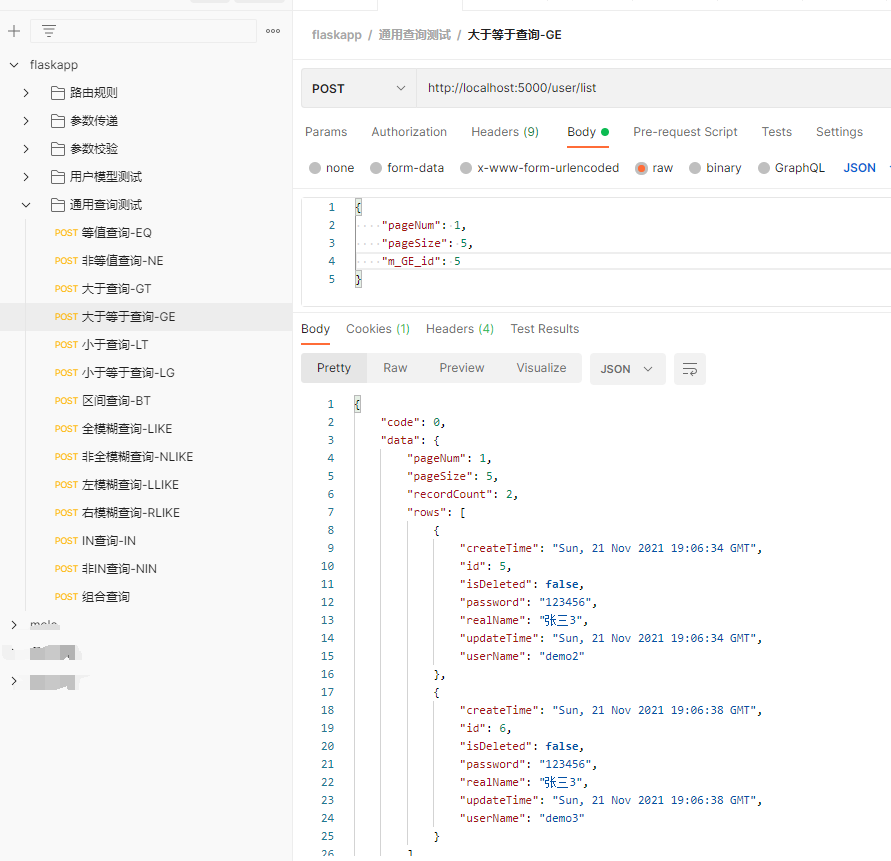
## 非等值查询-NE



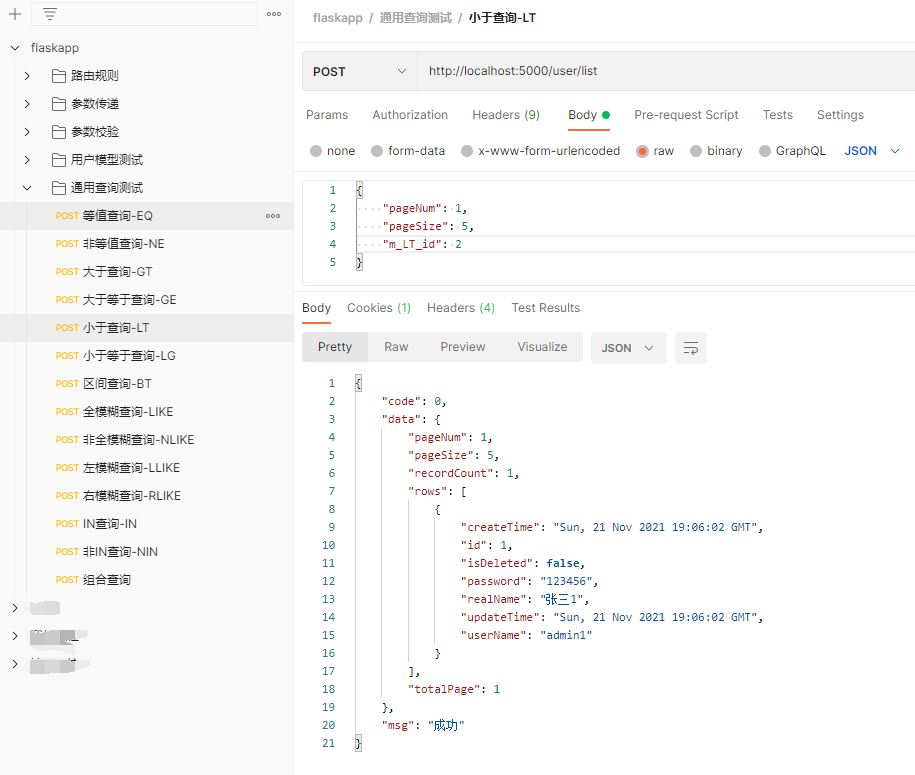
## 大于查询-GT



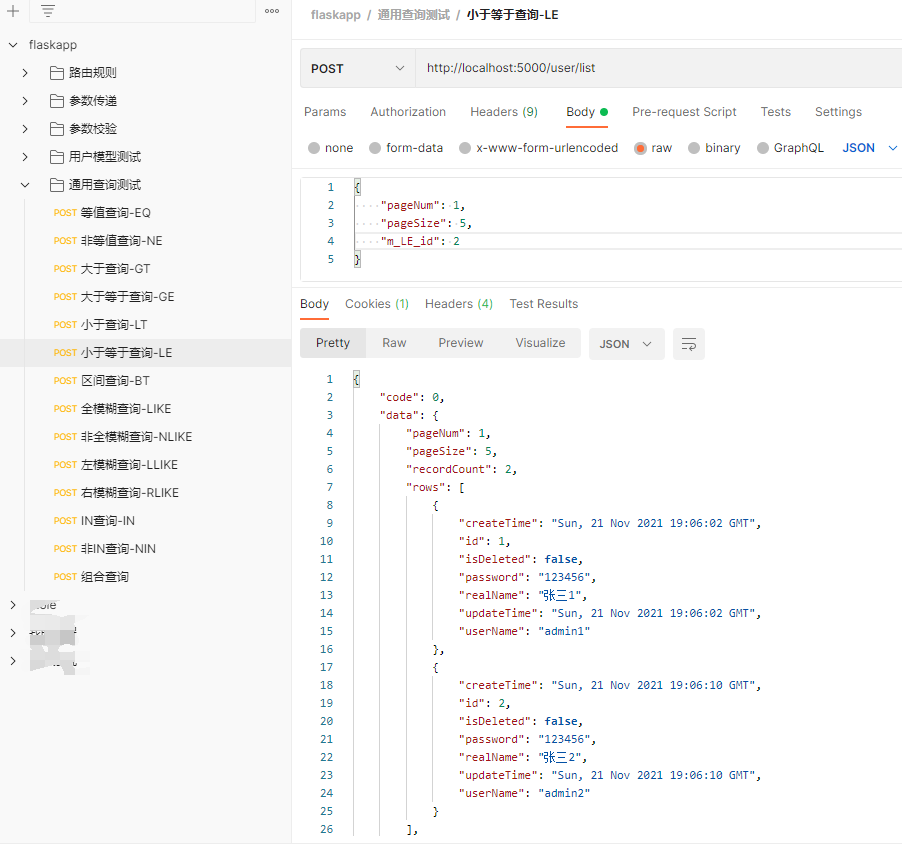
## 大于等于查询-GE



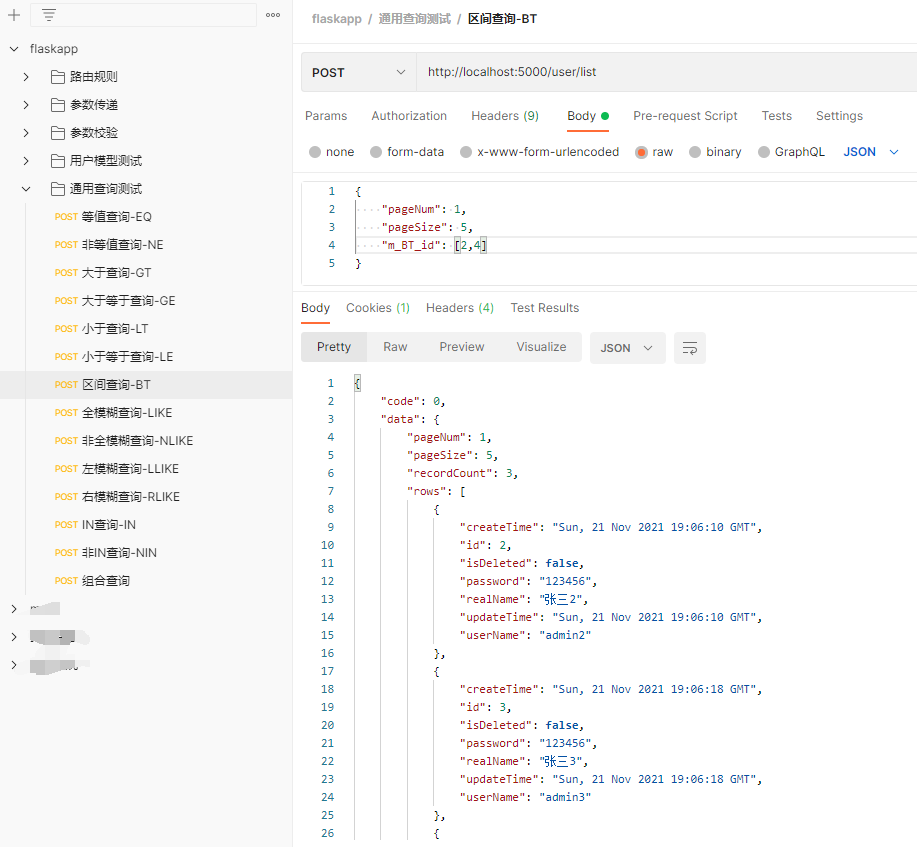
## 小于查询-LT



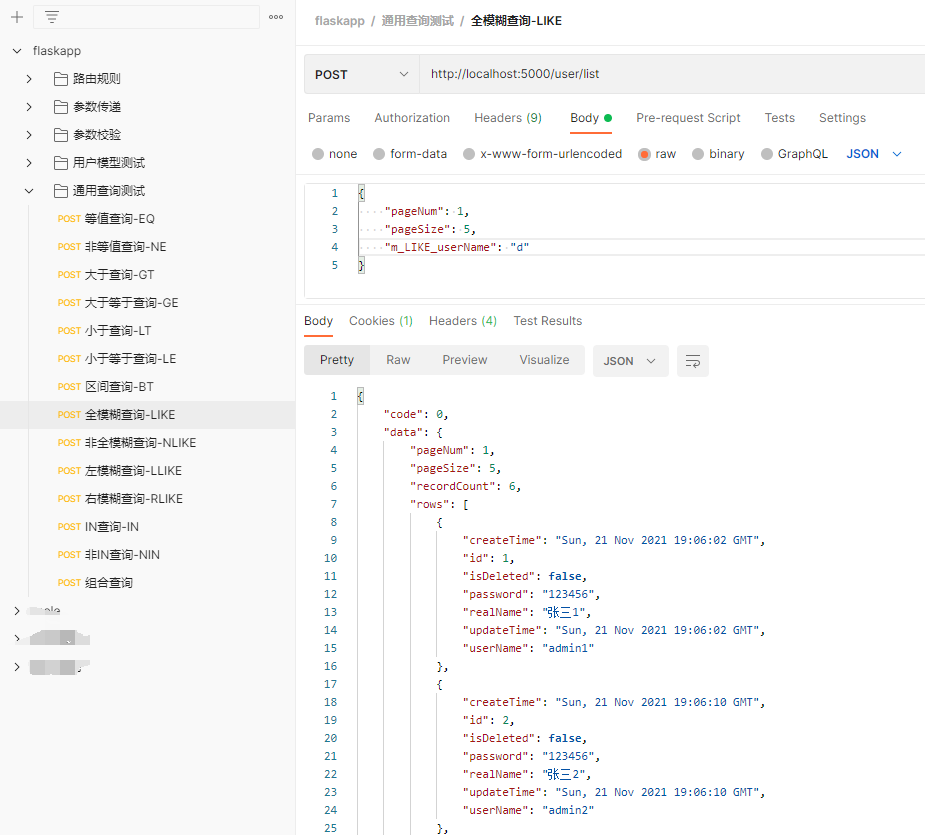
## 小于等于查询-LE



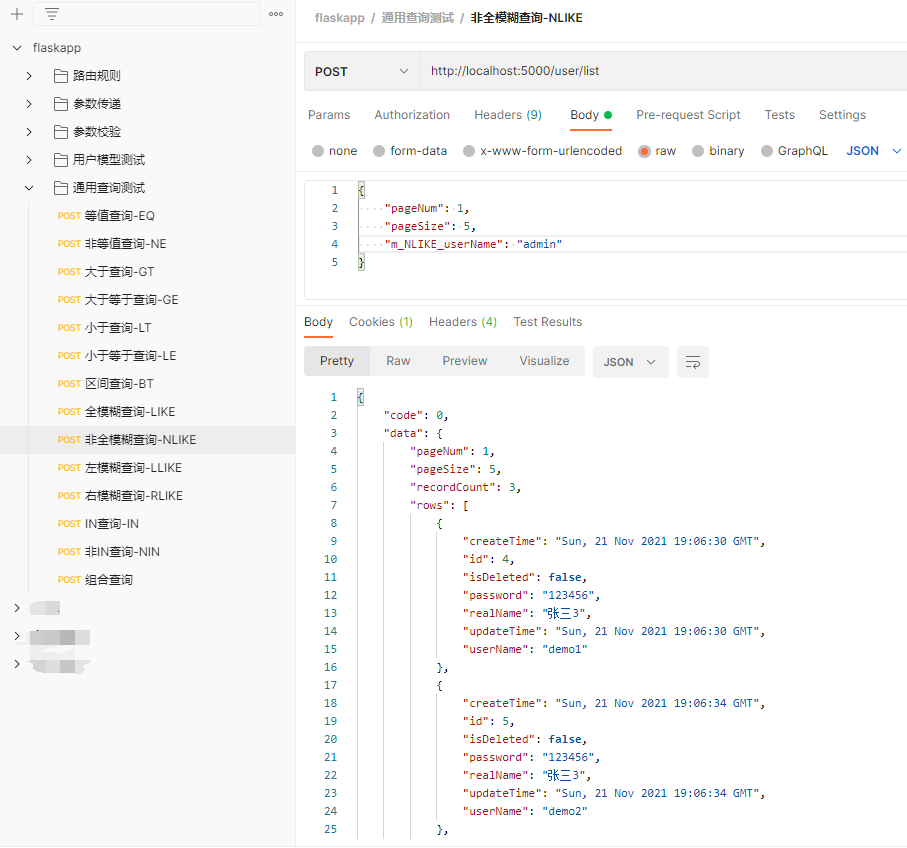
## 区间查询-BT



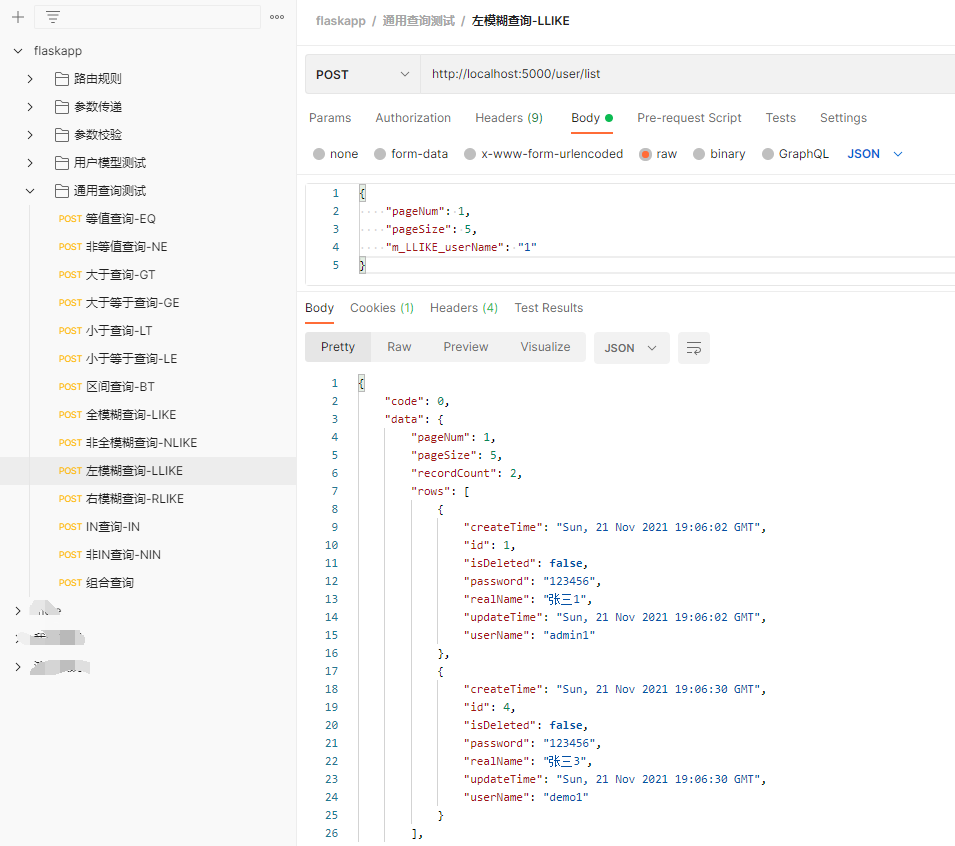
## 全模糊查询-LIKE



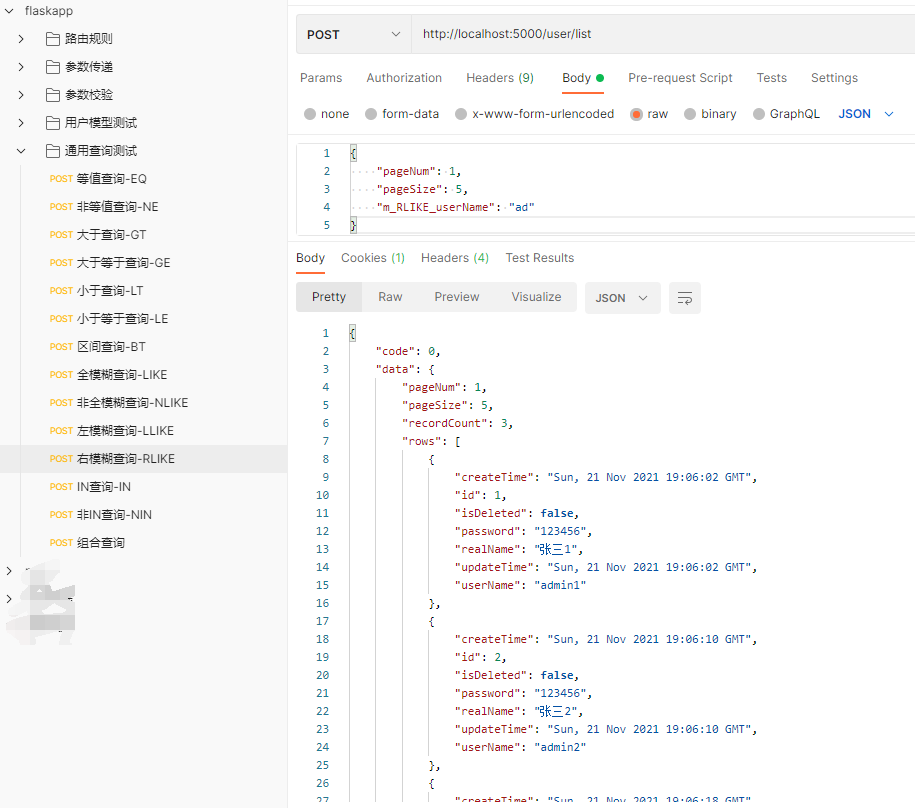
## 非全模糊查询-NLIKE



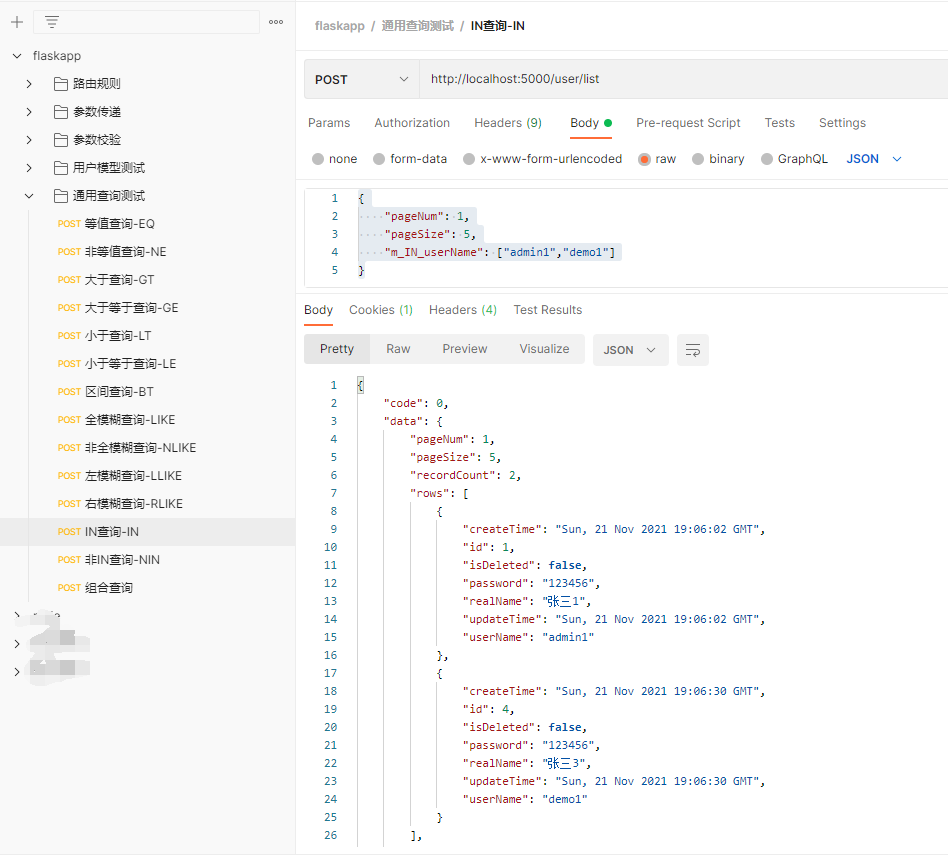
## 左模糊查询-LLIKE



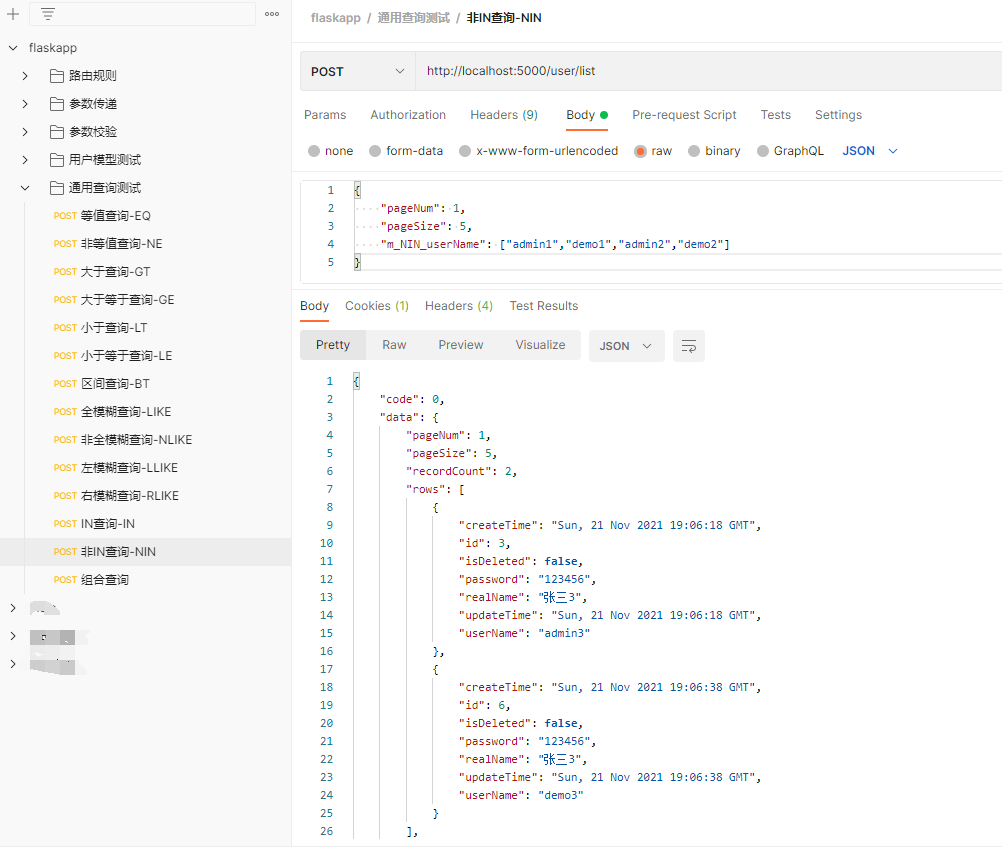
## 右模糊查询-RLIKE



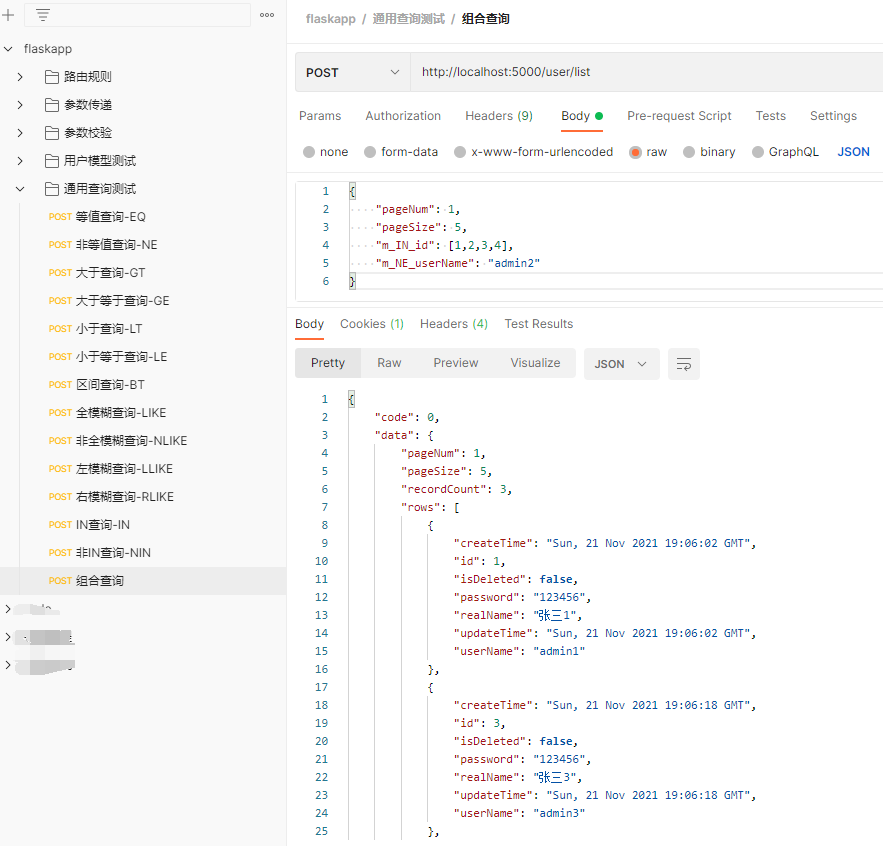
## IN查询-IN



## 非IN查询-NIN



## 组合查询



# postman导出文件

