

同济跳蚤市场软件系统重构报告

报告人：TLUAFED 小组

时间：2022.8

一、 软件重构任务简述

1. 重构动机

本软件系统为同济跳蚤市场，其功能强大且便利，是一个非常有价值的软件系统。我们小组对其进行了源码的阅读和架构的解析，在仔细的研究过程中，我们发现了该项目依然存在一定的改进空间，其存在诸如过长函数、重复冗余代码、设计模式缺陷等一系列问题，一定程度影响了该软件的可维护性、可扩展性、可读性等方面。因此我们选择对其进行重构操作，使得其架构更清晰，方法更通用、代码更规范，对该软件系统的各方面“查漏补缺”，提高其质量。

2. 重构计划以及人员分工

本小组依然以邬嘉晟为组长，所有人全程参与对于软件系统的分析、重构、测试，贡献度较为平均。

在重构过程中各成员最主要的工作如下所示：

邬嘉晟：方法抽取和修改

徐铭骋：方法抽取和修改

胡浚桐：架构调整

陈新一：架构调整+设计模式修改

赵一凡：类重构

罗劲桐：类重构+设计模式修改

*个人具体贡献度见附件

二、 现有软件系统分析

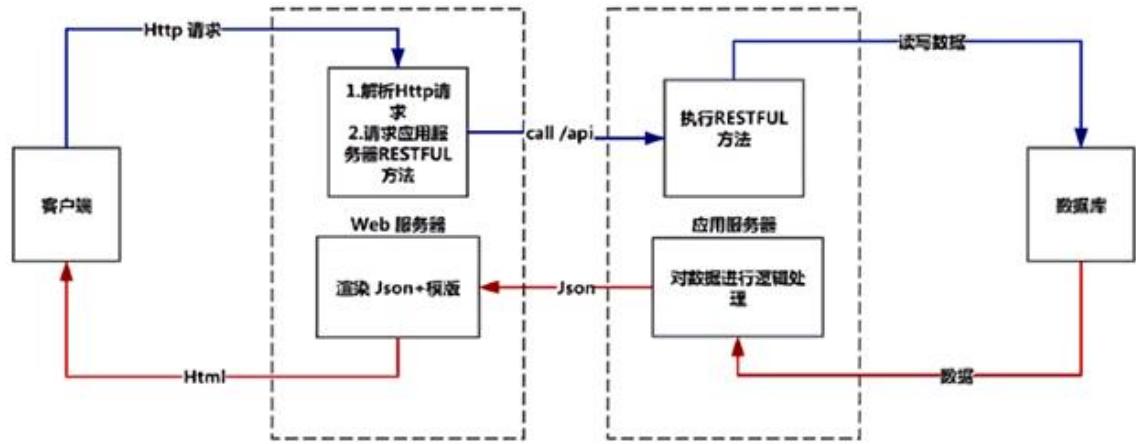
1. 软件系统架构分析

1.1 系统架构

该目标软件系统使用的是 MTV 架构。MTV 架构全称是 Model-Template-View。Model 负责业务对象和数据库的关系映射 (ORM)。Template 负责如何把页面展示给用户 (HTML)。View 负责业务逻辑，并在适当时候调用 Model 和 Template。

该架构的工作流程为：用户打开浏览器，浏览器发送请求，中间件接收用户请求，查找对应的视图并进行 URL 分发，视图层接受用户请求，接受完请求调用模型层，模型

层根据数据库创建模型，进行增删改查等操作，模型层处理完数据返回给视图层，视图层接收完数据调用模板层，模板层存放着 HTML 等页面，模板层将 HTML 模板页面返回给视图层，视图层填充数据到模板上，然后再返回给浏览器。



该架构有效地减小了 Model、Template、View 三部分的耦合，而且前后端均可根据接口进行独立开发，并且方便敏捷开发实践。一方面，我们使用的 flask 框架本身即适用 MTV 架构，另一方面，我们认为该架构较为适合该软件的实现，因此在重构中并未改变该架构。

1.2 优缺点分析

虽然系统使用的是 MTV 架构，但是目标软件的前后端并没有完全分开，项目被分为负责前端渲染的 route.py 部分和负责后端接口的 api 部分，这两部分虽然进行了简单的划分，但是耦合度仍较高，无法分开部署。

```
✓ TJ-FLEA-MARKET-RECONSTRUCTION
  > __pycache__
  > .vscode
  > admin
    ✓ api
      > __pycache__
      ⚡ __init__.py
      ⚡ admin_user_routes.py
      ⚡ chat_routes.py
      ⚡ item_routes.py
      ⚡ order_route_cs.py
      ⚡ order_routes.py
      ⚡ routes.py
      ⚡ send_verification_mail.py
      ⚡ utils.py
    { } verify_code.json
  > app
    > chat
      ✓ item
        > __pycache__
        > static
        > templates
        ⚡ __init__.py
        ⚡ models.py
        ⚡ routes.py
      > order
      > user
    ⚡ .gitignore
    ⚡ config.py
    ! environment.yml
    ⚡ main.py
    ⓘ README.md
    ⌂ requirements.txt
```

后端接口 ↵

前端接口 ↵

1.3 是否需要改进

在前端部分有大量的冗余代码，对于每一个页面路由都进行了用户身份验证、特定的权限管理等，这些部分应该汇集到一起，使用装饰器设计模式进行路由管理。

在后端部分接口划分的不够明确，许多功能相关性不强的视图函数、辅助函数混淆在同一个文件中。虽然软件能够正常运行，但是这存在着很多的不足。

前后端没有完全分开，应该使用 flask 来做后端、使用 Vue.js 等前端框架来做前端，这样能实现高内聚低耦合，减少应用服务器的并发/负载压力。

2. 代码味道分析

a) 设计模式相关

1. 缺少设计模式

在当前的 routes.py 下，实现了所有接口。然而，后端接口全部以单个函数的方式存在，没有进行组合和封装，不符合 MVC 模式中的 Controller 类要求。没有组合和封装的接口会导致接口混乱，难以查找接口和模块的对应关系。因此，需要创建 Controller 类以与相应的 Model 类对应。

- /user/routes.py 中的接口位于 user 蓝图下，引用了 User() 类都是 user 部分前端的接口实现。应该组合为 Controller 并与 User 的 model 相对应。

```
> @user_blue.route('/publish', methods=['GET', 'POST'])
> def publish():
>     if current_user.is_authenticated:
>         return render_template("user_publish.html", current_user=current_user)
>     else:
>         return redirect(url_for('login'))
>
>
> #个人中心
> @user_blue.route('/space', methods=['GET', 'POST'])
> def space():
>     return redirect(url_for('user.order'))
>     if current_user.is_authenticated:
>         return render_template('user_space.html', current_user=current_user)
>     else:
>         return redirect(url_for('login'))
>
>
> #个人信息管理
> @user_blue.route('/<opt_userid>/user_info', methods=['GET', 'POST'])
> def user_info(opt_userid: int): #opt_userid 为目标用户 ID
>     if current_user.is_authenticated:
>         try:
>             user = User.get(User.id == opt_userid)
>         except:
>             return render_template('404.html', message="找不到该用户")
>         return render_template('user_info.html',
>                               current_user=current_user,
>                               opt_userid=int(opt_userid))
>     else:
>         return render_template('404.html', error_code=401, message="请先登录")
```

- /item/routes.py 中的接口对应 Item, History, Favor 多个 model，需要对接口组合位控制类以明确相关的模型类

```

> @item_blue.route('/content/<item_id>', methods=['GET', 'POST'])
> def content(item_id: int): #goods_id/want_id
>     try:
>         item = Item.get(Item.id == item_id)
>     except:
>         return render_template('404.html', error_code=404, message="该商品不存在")
>     else:
>         if not current_user.is_authenticated:
>             isAdmin = False
>             isPub = False
>         else:
>             isAdmin = (current_user.state == User_state.Admin.value)
>             isPub = (item.user_id.id == current_user.id)
>             if item.state != Item_state.Sale.value and not isAdmin and not isPub:
>                 return render_template('404.html', error_code=401, message="该商品您现在无权
访问")
>
>         if current_user.is_authenticated: #已登录便加入历史
>             try:
>                 last = History.get(History.user_id == current_user.id,
>                                     History.item_id == item_id)
>             except Exception as e:
>                 last = History(user_id=current_user.id,
>                                item_id=item_id,
>                                visit_time=datetime.now())
>             else:
>                 last.visit_time = datetime.now()
>             finally:
>                 last.save()
>             # 加入: 当商品状态不为 0 时, 只有卖家和管理员可见, 其他人访问返回异常提示 (或 404 页面)
>             # (或开一个默认的, 代表被下架的商品)
>             return render_template('item_content.html',
>                                   item=item,
>                                   item_id=int(item_id))
>
>
> @item_blue.route('/publish/', methods=['GET', 'POST'])
> def publish():
>     if not current_user.is_authenticated:
>         return render_template('404.html', message="请先登录")
>     return render_template('item_publish.html')

```

2. 重复的验证方式

大部分接口类首先需要用户登录验证, 选择验证不合格跳转的网页 login,403,404 等网页。原先的 routes.py 中的接口类使用了大量重复的用户登录判断, 这既不易于统一修改也不易于代码的阅读。因此, 需要利用之前的 Controller 类, 绑定接口和对应的验证方法。

- /user/routes.py 中有大量接口使用完全一致的登录判断方法, 应该统一抽取这些方法。

```
> #历史
> @user_blue.route('/history', methods=['GET', 'POST'])
> def history():
>     if current_user.is_authenticated:
>         return render_template('user_history.html')
>     else:
>         return redirect(url_for('login'))
>
>
> #收藏
> @user_blue.route('/favor', methods=['GET', 'POST'])
> def favor():
>     if current_user.is_authenticated:
>         return render_template('user_favor.html')
>     else:
>         return redirect(url_for('login'))
>
>
> @user_blue.route('/address', methods=['GET', 'POST'])
> def address():
>     if current_user.is_authenticated:
>         return render_template('user_address.html')
>     else:
>         return redirect(url_for('login'))
>
>
> @user_blue.route("/feedback", methods=["GET", "POST"])
> def feedback():
>     if current_user.is_authenticated:
>         return render_template('user_feedback.html')
>     else:
>         return redirect(url_for('login'))
```

- /order/routes.py 中接口同样使用了登陆验证，但跳转目标不同，需要使用带参数的装饰器。

```

>     @order_blue.route('/review/<order_id>', methods=['GET', 'POST'])
>     def review(order_id: int): #order_id 为订单 ID
>         if not current_user.is_authenticated:
>             return render_template('404.html', message="请先登录")
>         try:
>             order = Order.get(Order.id == order_id,
>                               Order.state == Order_state.End.value)
>         except:
>             return render_template('404.html',
>                                   message="未找到对应已完成订单",
>                                   error_code=404)
>         try:
>             _order_item = Order_Item.get(Order_Item.order_id == order.id)
>         except:
>             return render_template('404.html',
>                                   message="未找到对应已完成订单明细",
>                                   error_code=404)
>         if _order_item.item_id.user_id.id != current_user.id and order.user_id.id != current_user.id: #不是订单双方
>             return render_template('404.html',
>                                   message='您不是订单双方, 无法评价',
>                                   error_code=403)
>         return render_template('order_review.html', order_id=order_id)
>
>
>     @order_blue.route('/generate/<int:item_id>', methods=['GET', 'POST'])
>     def generate(item_id: int):
>         if not current_user.is_authenticated:
>             return render_template('404.html', message="请先登录", error_code=403)
>         try:
>             item = Item.get(Item.id == item_id)
>         except:
>             return render_template('404.html', message="未找到该物品", error_code=404)
>         if item.state == Item_state.Sale.value and item.shelved_num > 0 and item.user_id.id != current_user.id: #正常在售且有存量且发布者不是当前用户
>             return render_template('order_generate.html', item=item)
>         else:
>             message = "不允许自己同自己做生意"
>             if item.shelved_num <= 0:
>                 message = "该商品售罄"
>             elif item.state != Item_state.Sale.value:
>                 message = "该商品或悬赏被下架或冻结"
>             return render_template('404.html', message=message, error_code=403)

```

b) 类封装相关

1.重复的类声明

BaseModel 类在不同文件中重复声明，这样在修改 BaseModel 类时，导致不同类的表现不一致，产生预期之外的情况。

```
> import peewee as pw
> # py_peewee 连接的数据库名:database
> database = pw.MySQLDatabase(
>     database=database_name,
>     host='127.0.0.1',
>     user=user,
>     passwd=password, #记得改密码, 不然你可能调试不了
>     charset='utf8',
>     port=3306)
>
>
> class BaseModel(pw.Model):
>
>     class Meta:
>         database = database # 将实体与数据库进行绑定
>
>
> # 连接数据库
> database.connect()
```

c) 方法抽取相关

1. 存在“过长的函数”

过长的长代码块很难重用和理解，因为它们通常负责不止一件事。将它们分成几个具有正确名称的短代码有助于代码重用，并使得我们能够更好地理解它。在本项目中存在一些过长的函数块，其中比较典型的案例具体如下：

- fake_data 函数过长

该函数用于初始化数据库，整体占数百行，其中完成对各种不同种类、功能的数据库的初始化，可以分拆

- `post_item_info` 函数过长，其中涵盖新建商品项目的流程中多种不同的操作，可以分拆

```
370 @api_blue.route('/post_item_info', methods=['POST'])
371 def post_item_info():
372     if not current_user.is_authenticated:
373         return make_response_json(401, "当前用户未登录")
374     if current_user.state == User_state.Under_ban.value:
375         return make_response_json(401, "当前用户已被封号")
376     data = request.get_json()
377     if len(data["name"]) > 40:
378         return make_response_json(400, "名称过长")
379     if len(data["description"]) > Item.description.max_length:
380         return make_response_json(400, "描述过长,应限制在[Item.description.max_length]字以内")
381     if "tag_id" not in data:
382         return make_response_json(400, "请选择物品类型")
383     if data["tag_id"] not in item_tag_type_value_member_map_:
384         return make_response_json(400, "请求格式错误")
385     try:
386
387         price = float(data["price"])
388         item_type = int(data["type"])
389         shelved_num = int(data["shelved_num"])
390     except Exception as e:
391         return make_response_json(400, "数据类型错误")
392     if price == float("inf") or price == float("nan"):
393         return make_response_json(400, "价格越界")
394     if price < 1e-8:
395         return make_response_json(400, "价格越界")
396     if item_type != Item_type.Goods.value and item_type != Item_type.Want.value:
397         return make_response_json(400, "仅能上传物品")
398     if shelved_num <= 0:
399         return make_response_json(400, "数量错误")
400     if shelved_num.bit_length() > Item.shelved_num.__sizeof__() - 1:
401         return make_response_json(400, "数量越界")
402     data["user_id"] = current_user.id
403     data["publish_time"] = datetime.now()
404     try:
405         new = Item.create(**data)
406     except Exception as e:
407
408         os.path.join(item_blue.static_folder,
409                      f'resource/item_pic/{new.id}/pic'))
410         default_pic = os.path.join(item_blue.static_folder,
411                                    'resource/default_pic/test.jpg')
412         curpath = os.path.join(item_blue.static_folder,
413                               f'resource/item_pic/{new.id}/')
414         tempath = os.path.join(item_blue.static_folder, f'resource/temp/')
415         if len(data["urls"]) == 0:
416             #给一个默认图
417             shutil.copy(default_pic, os.path.join(curpath, 'head/'))
418             shutil.copy(default_pic, os.path.join(curpath, 'pic/'))
419
420     else:
421
422         head_pics = [i["MD5"] for i in data["urls"] if i["is_cover_pic"]]
423         if len(head_pics) == 0:
424             head_pic = data["urls"][0]["MD5"]
425         elif len(head_pics) > 1:
426             #new.delete_instance()
427             #return make_response_json(400, "仅能选定一张头图")
428             head_pic = head_pics[0]
429         else:
430             head_pic = head_pics[0]
431             shutil.copy(os.path.join(tempath, head_pic),
432                        os.path.join(curpath, 'head/'))
433
434         img = Image.open(os.path.join(curpath, 'head/ ', head_pic))
435         img = trans_square(img)
436         img.show()
437         img.save(os.path.join(curpath, 'head/ ', head_pic), 'WEBP')
438
439         for j in data["urls"]:
440             shutil.move(os.path.join(tempath, j["MD5"]),
441                         os.path.join(curpath, 'pic/'))
442
443         #将所有的图片转到用户对应的文件夹
444         return make_response_json(200, "上传成功",
445                                  {"url": url_for('item.content', item_id=new.id)})
```

- `change_order_state` 函数过长，其中涵盖管理员、订单发起方、商品提供方的不同操作，可以分拆

- `get_search` 函数过长，其中涵盖请求数据检查、搜索依据添加、进行搜索并排序输出三个小功能，可以分拆

```

def get_search():
    data = request.get_json()
    if "range_min" in data or "range_max" in data:
        if "range_min" in data and "range_max" in data:
            try:
                range_min = int(data["range_min"])
                range_max = int(data["range_max"])
            except Exception as e:
                return make_response_json(400, "请求格式错误")
        else:
            return make_response_json(400, "请求格式错误")
    else:
        range_min = 0
        range_max = 50
    try:
        search_type = int(data["search_type"])
    except Exception as e:
        return make_response_json(400, "请求格式错误")
    if "key_word" in data:
        key_word = data["key_word"]
    else:
        return make_response_json(400, "请输入关键词")
    if "order_type" in data:
        order_type = data["order_type"]
    else:
        order_type = "name"
    #get_data = Item.select().where(_).execute()
    get_data = Item.select("need").where("select_need").execute()
    data = [i._data for i in get_data]
    new_data = list()
    if order_type == "time":
        data.sort(key=lambda x: x["publish_time"], reverse=True)
    elif order_type == "price":
        data.sort(key=lambda x: x["price"], reverse=False)
    else:
        #orderWay = (Item.publish_time.desc(),) # 改: 默认其实为倒叙
        data.sort(
            key=lambda x: SequenceMatcher(a=key_word, b=x["name"]).ratio(),
            reverse=True)
    for i in data:
        i['price'] = float(i['price'])
        i['publish_time'] = str(i['publish_time'])
    new_data.append(i)
    range_max = min(len(new_data), range_max)
    final_data = {
        "total_count": len(new_data),
        "item_list": new_data[range_min:range_max]
    }
    return make_response_json(200, "搜索结果如下", data=final_data)

```

- address 函数过长，其用了三个请求方式，分别是 post、put、delete，可以根据这三种请求方式将 address 拆分为三个函数，并对应同一个接口，这样使得函数更加精简

```

@api_blue.route("/address", methods=["POST", "PUT", "DELETE"])
def address():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    data = request.get_json()
    temp = []
    for i in range(len(data)):
        if request.method == "DELETE":
            data = list(set(map(lambda x: x["contact_id"], data)))
        for i, j in enumerate(data):
            try:
                temp[i] = Contact.get(Contact.id == int(j))
            except Exception as e:
                return make_response_json(401, "不存在的联络地址")
            else:
                if temp[i].user_id != current_user.id:
                    return make_response_json(401, "不可删除其他用户的联络地址")
        delete_default = False
    for i, j in enumerate(temp):
        try:
            j.delete_instance()
        except Exception as e:
            for t in range(i):
                temp[t].save()
            return make_response_json(500, f"发生错误 ({repr(e)})")
        else:
            if j.default:
                delete_default = True
    if delete_default:
        try:
            data = Contact.select().where(
                Contact.user_id == current_user.id).order_by(
                    Contact.id.desc()).execute()
        except Exception as e:
            print(repr(e))
        else:
            if len(data) > 0:
                data[0].default = True
                data[0].save()

```

2. 存在“重复（相似）的代码”

重复代码也是一种坏的代码味道，其指的是在不同函数中，存在着高度相似、重复或者功能相同的代码段，这样既会造成代码的冗余，也有可能在维护时对于功能的一致性造成影响，需要进行提炼。在本项目中存在一些重复的代码，其中比较典型的案例具体如下：

- `change_item_state` 和 `change_item_num` 中获取商品数据判断存在重复和共同点，可以提取

```
226     try:
227         data["item_id"] = int(data["item_id"])
228         data["state"] = int(data["state"])
229     except Exception as e:
230         return make_response_json(400, "请求格式不对")
231     try:
232         item = Item.get(Item.id == data["item_id"])
233     except Exception as e:
234         return make_response_json(404, "此商品不存在")
```

```

272    try:
273        data["item_id"] = int(data["item_id"])
274        data["num"] = int(data["num"])
275    except Exception as e:
276        return make_response_json(400, "请求格式不对")
277    if data["num"] < 0:
278        return make_response_json(400, "不允许负数物品存在")
279    if data["num"].bit_length() > Item.shelved_num.__sizeof__()-1:
280        return make_response_json(400, "数量越界")
281    try:
282        item = Item.get(Item.id == data["item_id"])
283    except Exception as e:
284        return make_response_json(404, "此商品不存在")
285

```

- post_item_info 和 change_item_data 中对于商品属性的获取和判断存在重复和共同点，可以提取

```

308     data = request.get_json()
309     if "tag" not in data:
310         return make_response_json(400, "请选择物品类型")
311     if data["tag"] not in Item_tag_type._value2member_map_:
312         return make_response_json(400, "请求格式错误")
313     if len(data["name"])>40:
314         return make_response_json(400,"名称过长")
315     if len(data["description"]) > Item.description.max_length:
316         return make_response_json(400,"描述过长")
317     try:
318         data["id"] = int(data["id"])
319         if "shelved_num" in data:
320             data["shelved_num"] = int(data["shelved_num"])
321         if "price" in data:
322             data["price"] = float(data["price"])
323     except Exception as e:
324         return make_response_json(400, "请求格式不对")
325     if data["shelved_num"] < 0:
326         return make_response_json(400, "不允许负数商品个数")
327     if data["price"] == float("inf") or data["price"] == float("nan"):
328         return make_response_json(400, "价格越界")
329     if data["price"] <= 0:
330         return make_response_json(400, "不允许非正数价格")
331     if data["shelved_num"].bit_length() > Item.shelved_num.__sizeof__()-1:
332         return make_response_json(400, "数量越界")
333     try:
334         item = Item.get(Item.id == data["id"])
335     except Exception as e:
336         return make_response_json(404, "此商品不存在")

```

```

377     if len(data["name"])>40:
378         return make_response_json(400,"名称过长")
379     if len(data["description"]) > Item.description.max_length:
380         return make_response_json(400,f"描述过长,应限制在{Item.description.max_length}字以内")
381     if "tag" not in data:
382         return make_response_json(400, "请选择物品类型")
383     if data["tag"] not in Item_tag_type._value2member_map_:
384         return make_response_json(400, "请求格式错误")
385     try:
386         price = Float(data["price"])
387         item_type = int(data["type"])
388         shelved_num = int(data["shelved_num"])
389     except Exception as e:
390         return make_response_json(400, "数据类型错误")
391     if price == Float("inf") or price == Float("nan"):
392         return make_response_json(400, "价格越界")
393     if price <= 1e-8:
394         return make_response_json(400, "价格越界")
395     if item_type != Item_type.Goods.value and item_type != Item_type.Want.value:
396         return make_response_json(400, "仅能上传物品")
397     if shelved_num <= 0:
398         return make_response_json(400, "数量越界")
399     if shelved_num.bit_length() > Item.shelved_num.__sizeof__()-1:
400         return make_response_json(400, "数量越界")

```

- `get_pillow_img_form_data_stream` 和 `save_pic` 非常相似，存在公共部分可以提取

```

def get_pillow_img_form_data_stream(data):
    ...
    传入request.files.get('something') (data类型为werkzeug.filestorage)
    将图片读取后按WEBP转换，保存入临时图床文件夹
    最后返回{400, 失败}或{200, 成功, [md5(str)]}
    ...
try:
    #os.path.join(item_blue.static_folder, f'resource/temp')
    #或
    #url_for('item.static', filename=f'resource/item_pic/{item_id}/[head|pic]')
    curpath = os.path.join(item_blue.static_folder, f'resource/temp')
    createPath(curpath)

    path_name = os.path.join(curpath, data.filename)
    createPath(curpath)
    data.save(path_name)

    img = Image.open(path_name)
    w, h = img.size
    ratio = max(w, h) / 1920
    if ratio > 1:
        img = img.resize((int(w / ratio), int(h / ratio)))
    ratio = 250 / min(w, h)
    if ratio > 1:
        img = img.resize((int(w * ratio), int(h * ratio)))
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    img.save(path_name_new, 'WEBP')
    img = Image.open(path_name_new)
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name_new)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    #if os.path.exists(path_name_new):
    #    return make_response_json(400, f"上传图片失败：请勿重复上传图片")
    img.save(path_name_new, 'WEBP')
except Exception as e:
    print(e)
    return make_response_json(400, f"上传图片失败：文件格式错误或损坏")
else:
    return make_response_json(200, "上传图片成功", md5_str)

def save_pic(path,data):
    ...
    传入request.files.get('something') (data类型为werkzeug.filestorage)
    将图片读取后按WEBP转换，保存入临时图床文件夹
    最后返回{400, 失败}或{200, 成功, [md5(str)]}
    ...
try:
    #os.path.join(item_blue.static_folder, f'resource/temp')
    #或
    #url_for('item.static', filename=f'resource/item_pic/{item_id}/[head|pic]')
    curpath = path
    createPath(curpath)

    path_name = os.path.join(curpath, data.filename)
    createPath(curpath)
    data.save(path_name)
    img = Image.open(path_name)
    w, h = img.size
    ratio = max(w, h) / 1920
    if ratio > 1:
        img = img.resize((int(w / ratio), int(h / ratio)))
    ratio = 250 / min(w, h)
    if ratio > 1:
        img = img.resize((int(w * ratio), int(h * ratio)))
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    img.save(path_name_new, 'WEBP')
    img = Image.open(path_name_new)
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name_new)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    #if os.path.exists(path_name_new):
    #    return make_response_json(400, f"上传图片失败：请勿重复上传图片")
    img.save(path_name_new, 'WEBP')
except Exception as e:
    print(e)
    return make_response_json(400, f"上传图片失败：文件格式错误或损坏")
else:
    return make_response_json(200, "上传图片成功", md5_str)

```

- `get_favor` 和 `get_history` 函数非常相似，存在重复和相似部分，可以提取

```

@api_blue.route("/get_favor", methods=["GET"])
def get_favor():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    req = dict(request.args)
    if "range_min" in req or "range_max" in req:
        if "range_min" in req and "range_max" in req:
            try:
                range_min = int(req["range_min"])
                range_max = int(req["range_max"])
            except Exception as e:
                return make_response_json(400, "请求格式错误")
        else:
            return make_response_json("请求格式错误")
    else:
        range_min = 0
        range_max = 50
    tep = Favor.select().where(Favor.user_id == current_user.id).order_by(
        Favor.collect_time.desc())
    fav_data = []
    for i in tep:
        res = dict()
        res['id'] = i.id
        res['item_id'] = i.item_id.id
        res['collect_time'] = str(i.collect_time)
        fav_data.append(res)
    range_max = min(len(fav_data), range_max)
    data = {
        "total_count": len(fav_data),
        "favor_list": fav_data[range_min:range_max]
    }
    return make_response_json(200, "操作成功", data)

@api_blue.route("/get_history", methods=["GET"])
def get_history():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    req = dict(request.args)
    if "range_min" in req or "range_max" in req:
        if "range_min" in req and "range_max" in req:
            try:
                range_min = int(req["range_min"])
                range_max = int(req["range_max"])
            except Exception as e:
                return make_response_json(400, "请求格式错误")
        else:
            return make_response_json("请求格式错误")
    else:
        range_min = 0
        range_max = 50
    tep = History.select().where(History.user_id == current_user.id).order_by(
        History.visit_time.desc())
    his_data = []
    for i in tep:
        res = dict()
        res['id'] = i.id
        res['item_id'] = i.item_id.id
        res['visit_time'] = str(i.visit_time)
        his_data.append(res)
    range_max = min(len(his_data), range_max)
    data = {
        "total_count": len(his_data),
        "history_list": his_data[range_min:range_max]
    }
    return make_response_json(200, "操作成功", data)

```

- `del_favor` 和 `item_delete_history` 函数非常相似，存在重复和相似部分，可以提取

```

@api_blue.route("/delete_favor", methods=["DELETE"])
def delete_favor():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    print(request.get_json())
    req = request.get_json()["item_id_list"]

    #tep = Item.select().where(Item.id << req) #在一个列表中查询
    #if tep.count() != len(req): #长度对不上
    #    return make_response_json(404, "不存在对应物品")
    try:
        NotFound = False
        for i in req:
            tep = Favor.select().where((Favor.user_id == current_user.id) & (Favor.item_id == i))

            if tep.count() <= 0:
                NotFound = True
            else:
                Favor.delete().where((Favor.user_id == current_user.id) & (Favor.item_id == i)).execute()

        if NotFound == True:
            return make_response_json(404, "不存在对应的收藏")
        return make_response_json(200, "删除成功")
    except Exception as e:
        return make_response_json(500, f"发生错误 {repr(e)}")

```

```

@api_blue.route("/delete_history", methods=["DELETE"])
def item_delete_history():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    print(request.get_json())
    req = request.get_json()["item_id_list"]

    #tep = Item.select().where(Item.id << req) #在一个列表中查询
    #if tep.count() != len(req): #长度对不上
    #    return make_response_json(404, "不存在对应物品")
    try:
        NotFound = False
        for i in req:
            tep = History.select().where((History.user_id == current_user.id) & (History.item_id == i))

            if tep.count() <= 0:
                NotFound = True
            else:
                History.delete().where((History.user_id == current_user.id) & (History.item_id == i)).execute()

        if NotFound == True:
            return make_response_json(404, "不存在对应的历史")
        return make_response_json(200, "删除成功")
    except Exception as e:
        return make_response_json(500, f"发生错误 {repr(e)}")

```

- `get_item_pics` 与 `get_item_head_pic` 函数非常相似，存在重复和相似部分，可以提取

```

def get_item_head_pic():
    data = dict(request.args)
    try:
        item_id = int(data["item_id"])
    except Exception as e:
        return make_response_json(400, f"请求格式错误 {repr(e)}")
    try:
        item = Item.get(Item.id == item_id)
    except Exception as e:
        return make_response_json(400, "此物品不存在")
    pic_path = os.path.join(item_blue.static_folder,
                           f'resource/item_pic/{item_id}/head')
    default_pic = os.path.join(item_blue.static_folder,
                               'resource/default_pic/test.jpg')
    if not os.path.exists(pic_path):
        createPath(pic_path)
    if len(os.listdir(pic_path)) == 0:
        shutil.copy(default_pic, pic_path)
    pic_list = os.listdir(pic_path)
    pic = url_for('item_static',
                  filename=f'resource/item_pic/{item_id}/head/{pic_list[0]}')
    return make_response_json(200, "图片查找到成功", data={"url": pic})

def get_item_pics():
    data = dict(request.args)
    try:
        item_id = int(data["item_id"])
    except Exception as e:
        return make_response_json(400, f"请求格式错误 {repr(e)}")
    try:
        item = Item.get(Item.id == item_id)
    except Exception as e:
        return make_response_json(400, "此物品不存在")
    pic_path = os.path.join(item_blue.static_folder,
                           f'resource/item_pic/{item_id}/pic')
    default_pic = os.path.join(item_blue.static_folder,
                               'resource/default_pic/test.jpg')
    if not os.path.exists(pic_path):
        createPath(pic_path)
    if len(os.listdir(pic_path)) == 0:
        shutil.copy(default_pic, pic_path)
    pic_list = os.listdir(pic_path)
    pics = list()
    for pic_name in pic_list:
        pics.append(

```

- order_post 函数中有一段代码重复出现，可以提取

```

if call_back[0] != 200:
    for t in range(start_num):
        item_list[t].shelved_num += data["item_info"][t]["num"]
        item_list[t].locked_num -= data["item_info"][t]["num"]
        item_list[t].save()
    return make_response_json(call_back[0], call_back[1])
try:
    contact = Contact.get(Contact.id == contact_id)
except Exception as e:
    for t in range(len(data["item_info"])):
        item_list[t].shelved_num += data["item_info"][t]["num"]
        item_list[t].locked_num -= data["item_info"][t]["num"]
        item_list[t].save()
    raise e
    od_it_list.append(od_it)
    od_it.item_id.type == Item_type.Goods.value:
    send_message(SYS_ADMIN_NO, od_it.item_id.user_id.id,
                 f"已有用户购买你的商品<{od_it.item_id.name}>, 请前往个人中心确认或取消订单")

```

d) 变量相关

有部分变量命名方式不统一，有的采取下划线式命名，有的采取驼峰式命名，可以将这些命名方式进行统一。

```

name = pw.CharField(verbose_name="商品名",
                    max_length=128,
                    index=True,
                    null=False)
user_id = pw.ForeignKeyField(User, verbose_name="发布的id")
publish_time = pw.DateTimeField(verbose_name="发布时间",
                                 null=False,
                                 default=datetime.now())
price = pw.FloatField(verbose_name="单价", default=0, null=False)
tag = pw.CharField(verbose_name="tag,用于分类", max_length=128, index=True)
state = pw.IntegerField(verbose_name="物品状态",
                        default=item_state.Sale.value,
                        constraints=[pw.Check("state >=1")])
type = pw.IntegerField(verbose_name="物品类型: 普通商品",
                       default=item_type.Goods.value,
                       constraints=[pw.Check("type>=1")])

```

```

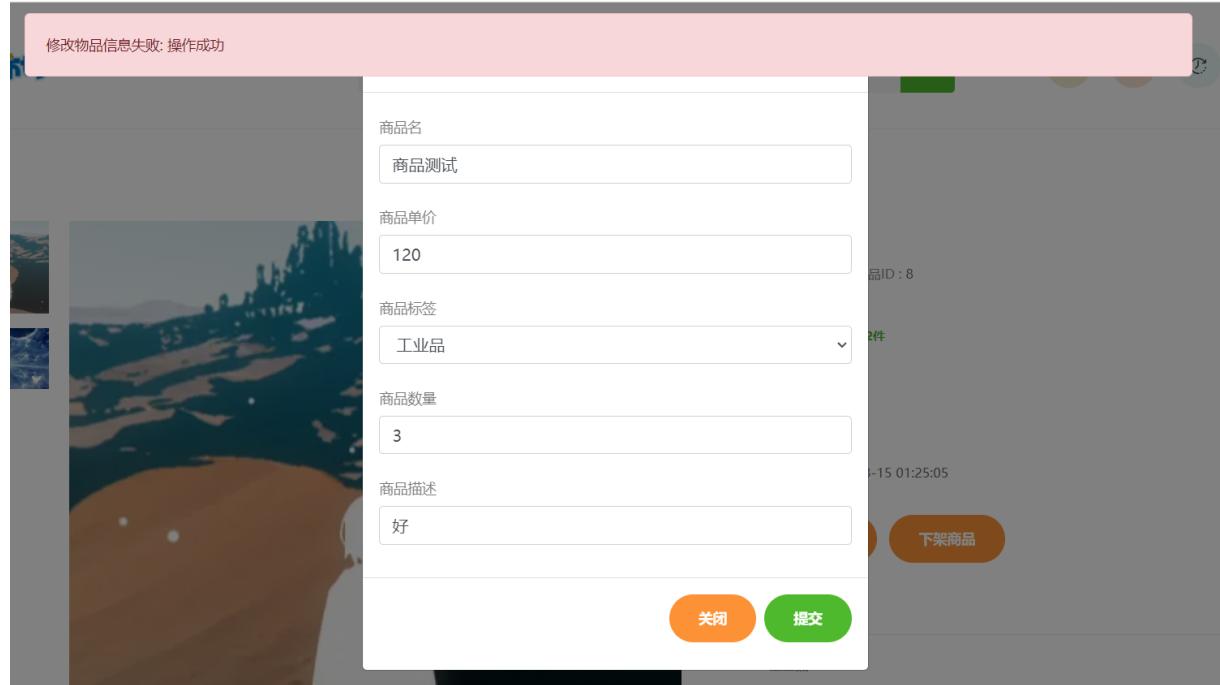
@item_blue.route('/content/<item_id>', methods=['GET', 'POST'])
def content(item_id: int): #goods_id/want_id
    try:
        item = Item.get(Item.id == item_id)
    except:
        return render_template('404.html', error_code=404, message="该商品不存在")
    else:
        if not current_user.is_authenticated:
            isAdmin = False
            isPub = False
        else:
            isAdmin = [current_user.state == User_state.Admin.value]
            isPub = (item.user_id.id == current_user.id)
            if item.state != item_state.Sale.value and not isAdmin and not isPub:
                return render_template('404.html', error_code=401, message="该商品您现在无权访问")

```

e) Bug

根据其原本的测试报告，该软件绝大多数 bug 都已经在交付前被成功修复了，但在我们的检查中，还发现了零星的 bug 存在，例如：

在成功修改商品信息时，提示框仍为错误提示



三、 对软件系统的改进

1. 对系统架构的改进

改进 1.1 分离后端接口模块

● 改进原因

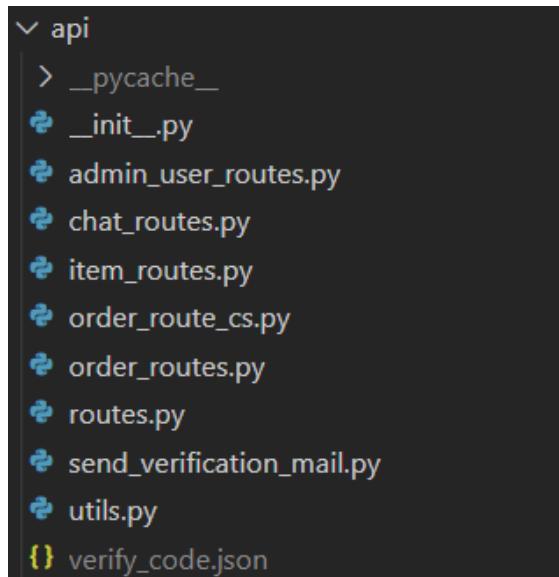
原项目中，后端的所有接口都放在了/api 文件夹下，许多功能相关性不强的视图函数、辅助函数混淆在同一个文件中。虽然能够使软件正常运行，但是这存在着很多的不足。最严重的两大问题、也是确实影响过我们的开发的两大问题是：

- a) 可读性差：这使得软件的质量较低，再次阅读代码时，很难寻找想要阅读的代码。
- b) 可扩展性差：当想要增加一个功能时，需要在原有的文件中进行添加与修改，这样会导致在添加、修改的过程中容易影响原有的软件，可能会导致函数重名、误删误加等等问题。

● 改进结果

将所有的后端接口按照功能分解为用户管理（user）、物品管理（item）、订单管理（order）三个模块。网盘，我们将上述四个模块分别实现为一个代码包，将公共函数放在 utils.py 中，使整个项目的结构更加清晰。同时我们使用了 flask 的蓝图来进行改进，将后端接口的每一个模块放在指定的路径下。

重构前项目后端接口如下图所示：



重构后项目后端接口架构如下图所示：

```
▽ api
  > __pycache__
  ▽ item
    + __init__.py
    + item_routes.py
  ▽ order
    + __init__.py
    + order_route_cs.py
    + order_routes.py
  ▽ user
    + __init__.py
    + admin_user_routes.py
    + chat_routes.py
    + send_verification_mail.py
    + __init__.py
    + routes.py
    + utils.py
  {} verify_code.json
```

- 测试结果

经过调整后很好的通过了自动化测试以及后续的人工测试。

改进 1.2 引入装饰器模式重构前端路由权限管理

- 改进原因

原项目中，在前端的路由部分存在大量冗余的判断。为了实现一个很简单的权限管理，原团队成员使用大量重复的 if-else 语句进行特判，这样的代码在后期维护的时候不易管理，可扩展性差。应当引入装饰器模式来对路由权限判断进行重构。

- 改进结果

引入 flask 的装饰器钩子函数 @app.before_request, @app.before_request 函数被修饰以后，每一次被 @app.route 装饰请求到来后，都会先执行它，如果没问题即正常执行，那么就会进入到正常的被 app.route 修饰的函数中进行响应，如果有多个函数被 app.before_request 修饰了，那么这些函数会被依次执行。

将权限管理放在 @app.before_request 中，可以做到统一的路由权限管理，同时提高项目的可扩展性。

重构前项目后端接口如下图所示，含有大量冗余的权限管理。

```

@user_blue.route('/index', methods=['GET', 'POST'])
def index():
    if current_user.is_authenticated:
        return render_template("user_index.html",
                               current_user=current_user) #html的名字不能相同
    else:
        return redirect(url_for('login'))

@user_blue.route('/publish', methods=['GET', 'POST'])
def publish():
    if current_user.is_authenticated:
        return render_template("user_publish.html", current_user=current_user)
    else:
        return redirect(url_for('login'))

#个人中心
@user_blue.route('/space', methods=['GET', 'POST'])
def space():
    return redirect(url_for('user.order'))
    if current_user.is_authenticated:
        return render_template('user_space.html', current_user=current_user)
    else:
        return redirect(url_for('login'))

```

重构后项目后端接口架构如下图所示：

```

@user_blue.before_request
def before_request():
    if database.is_closed():
        database.connect()
    if current_user.is_authenticated:
        return render_template("user_publish.html", current_user=current_user)

@user_blue.route('/index', methods=['GET', 'POST'])
def index():
    return redirect(url_for('login'))

@user_blue.route('/publish', methods=['GET', 'POST'])
def publish():
    return redirect(url_for('login'))

#个人中心
@user_blue.route('/space', methods=['GET', 'POST'])
def space():
    return redirect(url_for('user.order'))

```

- 测试结果

通过了自动化测试以及后续的人工测试。

2. 对设计模式和类封装的改进

考虑到 flask 框架的特性，以及该软件的实现情况，我们没有对总设计模式——MTV 模式进行改变，而是在 MTV 模式内部进行部分改进，以增强软件的可扩展性和灵活性，提高软件系统的可维护性和可复用性。

改进 2.1Controller 类定义

- 改进原因

后端接口全部以单个函数的方式存在，没有进行组合和封装，不符合 MVC 模式中的 Controller 类要求。以 /user/routes.py 和 /item/routes.py 为例

```
> @user_blue.route('/publish', methods=['GET', 'POST'])
> def publish():
>     if current_user.is_authenticated:
>         return render_template("user_publish.html", current_user=current_user)
>     else:
>         return redirect(url_for('login'))
>
>
> #个人中心
> @user_blue.route('/space', methods=['GET', 'POST'])
> def space():
>     return redirect(url_for('user.order'))
>     if current_user.is_authenticated:
>         return render_template('user_space.html', current_user=current_user)
>     else:
>         return redirect(url_for('login'))
>
>
> #个人信息管理
> @user_blue.route('/<opt_userid>/user_info', methods=['GET', 'POST'])
> def user_info(opt_userid: int): #opt_userid 为目标用户 ID
>     if current_user.is_authenticated:
>         try:
>             user = User.get(User.id == opt_userid)
>         except:
>             return render_template('404.html', message="找不到该用户")
>         return render_template('user_info.html',
>                               current_user=current_user,
>                               opt_userid=int(opt_userid))
>     else:
>         return render_template('404.html', error_code=401, message="请先登录")
```

```

> @item_blue.route('/content/<item_id>/', methods=['GET', 'POST'])
> def content(item_id: int): #goods_id/want_id
>     try:
>         item = Item.get(Item.id == item_id)
>     except:
>         return render_template('404.html', error_code=404, message="该商品不存在")
>     else:
>         if not current_user.is_authenticated:
>             isAdmin = False
>             isPub = False
>         else:
>             isAdmin = (current_user.state == User_state.Admin.value)
>             isPub = (item.user_id.id == current_user.id)
>             if item.state != Item_state.Sale.value and not isAdmin and not isPub:
>                 return render_template('404.html', error_code=401, message="该商品您现在无权
访问")
>
>         if current_user.is_authenticated: #已登录便加入历史
>             try:
>                 last = History.get(History.user_id == current_user.id,
>                                     History.item_id == item_id)
>             except Exception as e:
>                 last = History(user_id=current_user.id,
>                                item_id=item_id,
>                                visit_time=datetime.now())
>             else:
>                 last.visit_time = datetime.now()
>             finally:
>                 last.save()
>             # 加入: 当商品状态不为 0 时, 只有卖家和管理员可见, 其他人访问返回异常提示 (或 404 页面)
>             # (或开一个默认的, 代表被下架的商品)
>             return render_template('item_content.html',
>                                   item=item,
>                                   item_id=int(item_id))
>
>
> @item_blue.route('/publish/', methods=['GET', 'POST'])
> def publish():
>     if not current_user.is_authenticated:
>         return render_template('404.html', message="请先登录")
>     return render_template('item_publish.html')

```

- 改进结果

将属于同一类的接口组合成 Controller 类。另外, 由于 flask 依赖代码注入, 因此 controller 类接口使用静态成员函数实现。

```
> class UserController:
>     @staticmethod
>     @user_blue.route('/publish', methods=['GET'])
>     @requires_auth()
>     def publish():
>         # if current_user.is_authenticated:
>             return render_template("user_publish.html", current_user=current_user)
>         # else:
>             #     return redirect(url_for('login'))
>
>         # 个人中心
>         @staticmethod
>         @user_blue.route('/space', methods=['GET'])
>         def space():
>             return redirect(url_for('user.order'))
>             if current_user.is_authenticated:
>                 return render_template('user_space.html', current_user=current_user)
>             else:
>                 return redirect(url_for('login'))
>
>         # 个人信息管理
>         @staticmethod
>         @user_blue.route('/<opt_userid>/user_info', methods=['GET'])
>         @requires_auth(show_err=401)
>         def user_info(opt_userid: int): # opt_userid为目标用户 ID
>             # if current_user.is_authenticated:
>             try:
>                 user = User.get(User.id == opt_userid)
>             except:
>                 return render_template('404.html', message="找不到该用户")
>             return render_template('user_info.html',
>                                   current_user=current_user,
>                                   opt_userid=int(opt_userid))
>
>             # else:
>             #     return render_template('404.html', error_code=401, message="请先登录")
```

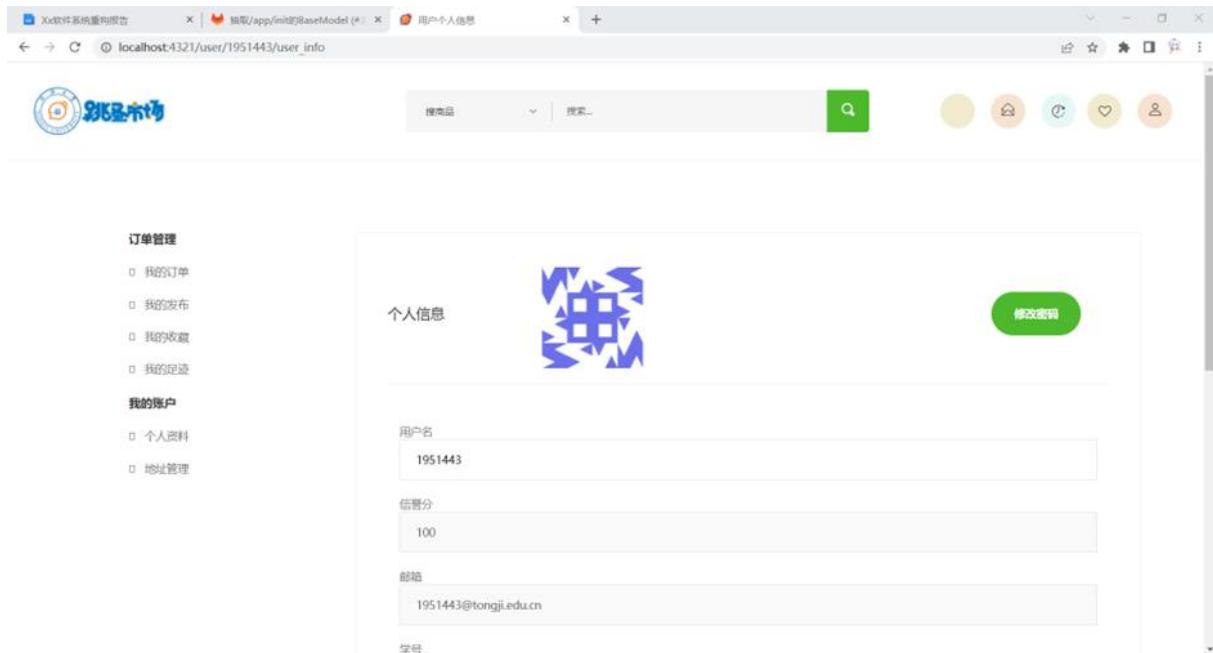
```

>     class ItemController:
>         @staticmethod
>         @item_blue.route('/content/<item_id>', methods=['GET'])
>         def content(item_id: int): # goods_id/want_id
>             try:
>                 item = Item.get(Item.id == item_id)
>             except:
>                 return render_template('404.html', error_code=404, message="该商品不存在")
>             else:
>                 if not current_user.is_authenticated:
>                     is_admin = False
>                     is_pub = False
>                 else:
>                     is_admin = (current_user.state == User_state.Admin.value)
>                     is_pub = (item.user_id.id == current_user.id)
>                     if item.state != Item_state.Sale.value and not is_admin and not is_pub:
>                         return render_template('404.html', error_code=401, message="该商品您现在无权访问")
>
>                 if current_user.is_authenticated: # 已登录便加入历史
>                     try:
>                         last = History.get(History.user_id == current_user.id,
>                                             History.item_id == item_id)
>                     except Exception as e:
>                         last = History(user_id=current_user.id,
>                                         item_id=item_id,
>                                         visit_time=datetime.now())
>                     else:
>                         last.visit_time = datetime.now()
>                     finally:
>                         last.save()
>                     # 加入: 当商品状态不为0时, 只有卖家和管理员可见, 其他人访问返回异常提示(或404页面) (或开一个默认的, 代表被下架的商品)
>                     return render_template('item_content.html',
>                                           item=item,
>                                           item_id=int(item_id))
>
>         @staticmethod
>         @item_blue.route('/publish/', methods=['GET'])
>         @auth_func
>         def publish():
>             # if not current_user.is_authenticated:
>             #     return render_template('404.html', message="请先登录")
>             return render_template('item_publish.html')

```

- 测试结果

上例中个人信息界面、商品界面都能正常打开访问。



改进 2.2 用户登录验证，自动跳转

● 改进原因

大部分接口类使用完全一致的登录判断方法，选择验证不合格跳转的网页 login, 403, 404 等网页。这既不易于统一修改也不易于代码的阅读。因此，需要利用之前的 Controller 类，绑定接口和对应的验证方法，应该统一抽取这些方法。

```
> #历史
> @user_blue.route('/history', methods=['GET', 'POST'])
> def history():
>     if current_user.is_authenticated:
>         return render_template('user_history.html')
>     else:
>         return redirect(url_for('login'))
>
>
> #收藏
> @user_blue.route('/favor', methods=['GET', 'POST'])
> def favor():
>     if current_user.is_authenticated:
>         return render_template('user_favor.html')
>     else:
>         return redirect(url_for('login'))
>
>
> @user_blue.route('/address', methods=['GET', 'POST'])
> def address():
>     if current_user.is_authenticated:
>         return render_template('user_address.html')
>     else:
>         return redirect(url_for('login'))
>
>
> @user_blue.route("/feedback", methods=["GET", "POST"])
> def feedback():
>     if current_user.is_authenticated:
>         return render_template('user_feedback.html')
>     else:
>         return redirect(url_for('login'))
```

```

>     @order_blue.route('/review/<order_id>', methods=['GET', 'POST'])
>     def review(order_id: int): #order_id 为订单 ID
>         if not current_user.is_authenticated:
>             return render_template('404.html', message="请先登录")
>         try:
>             order = Order.get(Order.id == order_id,
>                               Order.state == Order_state.End.value)
>         except:
>             return render_template('404.html',
>                                   message="未找到对应已完成订单",
>                                   error_code=404)
>         try:
>             _order_item = Order_Item.get(Order_Item.order_id == order.id)
>         except:
>             return render_template('404.html',
>                                   message="未找到对应已完成订单明细",
>                                   error_code=404)
>         if _order_item.item_id.user_id.id != current_user.id and order.user_id.id != current_user.id: #不是订单双方
>             return render_template('404.html',
>                                   message='您不是订单双方, 无法评价',
>                                   error_code=403)
>         return render_template('order_review.html', order_id=order_id)

>
>

>     @order_blue.route('/generate/<int:item_id>', methods=['GET', 'POST'])
>     def generate(item_id: int):
>         if not current_user.is_authenticated:
>             return render_template('404.html', message="请先登录", error_code=403)
>         try:
>             item = Item.get(Item.id == item_id)
>         except:
>             return render_template('404.html', message="未找到该物品", error_code=404)
>         if item.state == Item_state.Sale.value and item.shelved_num > 0 and item.user_id.id != current_user.id: #正常在售且有存量且发布者不是当前用户
>             return render_template('order_generate.html', item=item)
>         else:
>             message = "不允许自己同自己做生意"
>             if item.shelved_num <= 0:
>                 message = "该商品售罄"
>             elif item.state != Item_state.Sale.value:
>                 message = "该商品或悬赏被下架或冻结"
>             return render_template('404.html', message=message, error_code=403)

```

● 改进结果

使用装饰器，定义在 controller 类中，并修饰 controller 类中接口函数。维持源代码逻辑，采用带参数的装饰器，选择验证不合格跳转的网页 login, 403, 404 等网页。

```
>     class UserController:
>         @staticmethod
>         def requires_auth(show_err=0):
>             def auth_func(f):
>                 def authenticate():
>                     if show_err == 0:
>                         return redirect(url_for('login'))
>                     elif show_err == 401:
>                         return render_template('404.html', error_code=401, message="请先登录")
>                     elif show_err == 404:
>                         return render_template("404.html", error_code=404, message="此反馈不存在")
>                     else:
>                         raise Exception("unexpected requires_auth param")
>
>                 @wraps(f)
>                 def decorated(*args, **kwargs):
>                     from flask_login import current_user
>                     if current_user.is_authenticated:
>                         return f(*args, **kwargs)
>                     else:
>                         return authenticate()
>
>                 return decorated
>
>             return auth_func
>
>         # 历史
>         @staticmethod
>         @user_blue.route('/history', methods=['GET'])
>         @requires_auth()
>         def history():
>             # if current_user.is_authenticated:
>             return render_template('user_history.html')
>
>             # else:
>             #     return redirect(url_for('login'))
>
>         # 收藏
>         @staticmethod
>         @user_blue.route('/favor', methods=['GET'])
>         @requires_auth()
>         def favor():
>             # if current_user.is_authenticated:
>             return render_template('user_favor.html')
>
>             # else:
>             #     return redirect(url_for('login'))
>
>         @staticmethod
>         @user_blue.route('/address', methods=['GET'])
>         @requires_auth()
>         def address():
>             # if current_user.is_authenticated:
>             return render_template('user_address.html')
>
>             # else:
```

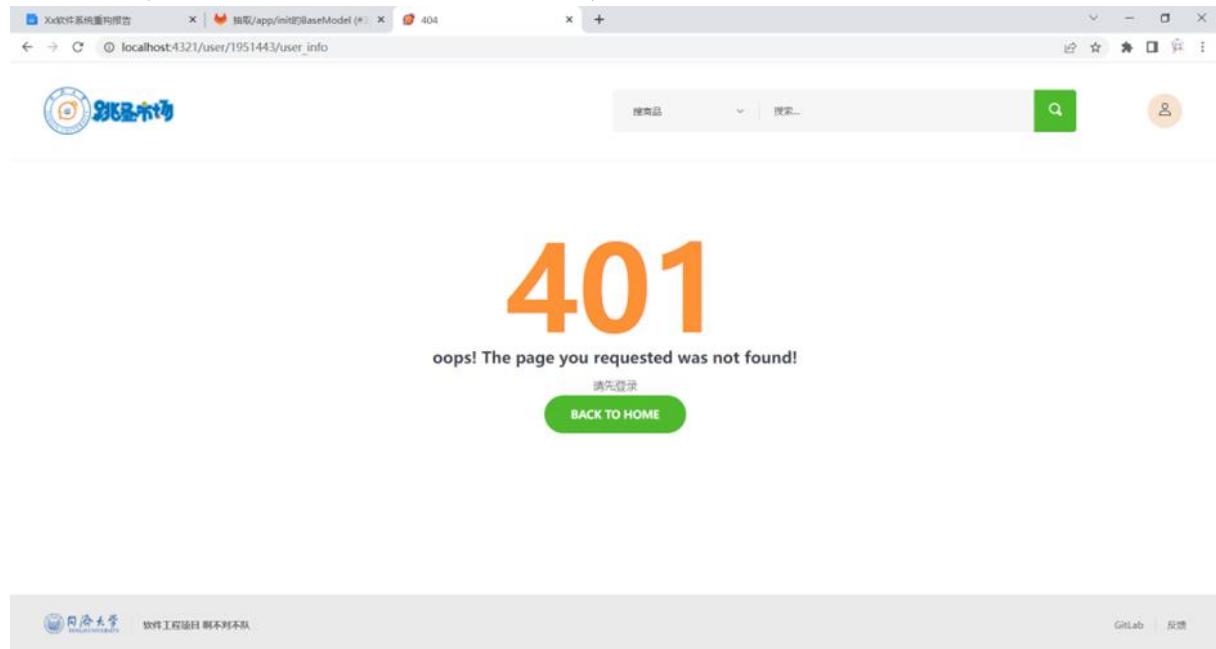
```

>     class OrderController:
>         @staticmethod
>         def auth_func(f):
>             def authenticate():
>                 return render_template('404.html', message="请先登录", error_code=403)
>
>             @wraps(f)
>             def decorated(*args, **kwargs):
>                 from flask_login import current_user
>                 if current_user.is_authenticated:
>                     return f(*args, **kwargs)
>                 else:
>                     return authenticate()
>
>             return decorated
>
>         @staticmethod
>         @order_blue.route('/<int:order_id>', methods=['GET'])
>         def order_view(order_id: int):
>             return render_template("order_view.html", order_id=order_id)
>
>         @staticmethod
>         @order_blue.route('/review/<order_id>', methods=['GET'])
>         @auth_func
>         def review(order_id: int): # order_id 为订单 ID
>             # if not current_user.is_authenticated:
>             #     return render_template('404.html', message="请先登录")
>             try:
>                 order = Order.get(Order.id == order_id,
>                               Order.state == Order_state.End.value)
>             except:
>                 return render_template('404.html',
>                                       message="未找到对应已完成订单",
>                                       error_code=404)
>             try:
>                 _order_item = Order_Item.get(Order_Item.order_id == order.id)
>             except:
>                 return render_template('404.html',
>                                       message="未找到对应已完成订单明细",
>                                       error_code=404)
>             if _order_item.item_id.user_id.id != current_user.id and order.user_id.id != current_user.id: # 不是订单双方
>                 return render_template('404.html',
>                                       message='您不是订单双方, 无法评价',
>                                       error_code=403)
>             return render_template('order_review.html', order_id=order_id)
>
>         @staticmethod
>         @order_blue.route('/generate/<int:item_id>', methods=['GET'])
>         @auth_func
>         def generate(item_id: int):
>             # if not current_user.is_authenticated:
>             #     return render_template('404.html', message="请先登录", error_code=403)
>             try:
>                 item = Item.get(Item.id == item_id)
>             except:
>                 return render_template('404.html', message="未找到该物品", error_code=404)

```

- 测试结果

在无登录状态下，自动跳转到登录页面或报错页面



改进 2.3 抽取/app/init 的 BaseModel

● 改进原因

BaseModel 类在不同文件中重复声明，这样在修改 BaseModel 类时，导致不同类的表现不一致，产生预期之外的情况。

```
> import peewee as pw
> # py_peewee 连接的数据库名:database
> database = pw.MySQLDatabase(
>     database=database_name,
>     host='127.0.0.1',
>     user=user,
>     passwd=password, #记得改密码，不然你可能调试不了
>     charset='utf8',
>     port=3306)
>
>
>
> class BaseModel(pw.Model):
>
>     class Meta:
>         database = database # 将实体与数据库进行绑定
>
>
>     # 连接数据库
>     database.connect()
```

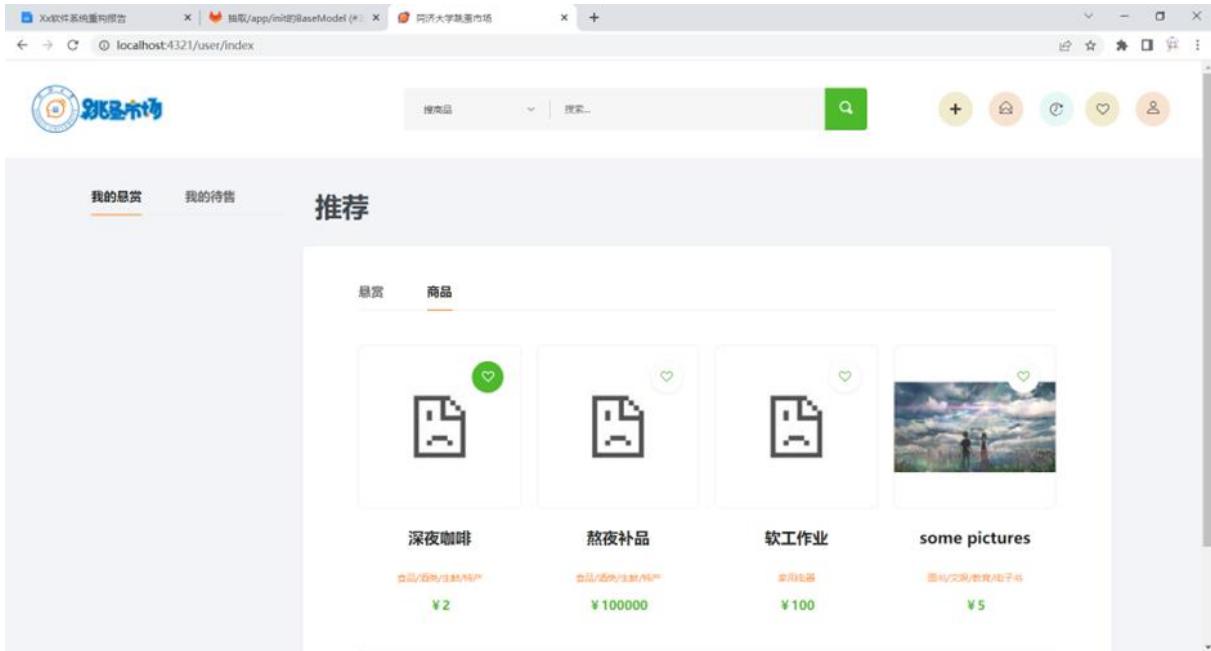
● 改进结果

在/app/init 定义 BaseModel，并全局连接数据库。在其他文件中删除 BaseModel 的定义，并使用 import 代替

```
> from app import BaseModel
> import peewee as pw
```

● 测试结果

网页正常载入



改进 2.4 将 API 合并为 Controller 类

- 改进原因

各种相关的接口都是零散的放在文件中，但是这些接口都是和用户账号相关，因此可以将其合并为一个控制类。

```
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-
from api.utils import *
from api import api_blue
from datetime import datetime, timedelta
from .send_verification_mail import send_email

def judge_user_id(user_id: str):
    try:
        user_id = str(user_id).strip()
    except:
        return [400, "账号格式错误"]
    if '@' not in user_id:
        user_id += "@tongji.edu.cn"
    pattern = re.compile(r'^\d{5,7}@tongji\.edu\.cn$')
    result = pattern.findall(user_id)
    if len(result) > 0:
        return [0, "验证通过"]
    pattern = re.compile(r'@tongji\.edu\.cn$')
    result = pattern.findall(user_id)
    if len(result) > 1:
        return [400, "邮箱账号必须为学号!"]
    return [400, "账号格式错误"]

curpath = os.path.dirname(__file__)
config = os.path.join(curpath, 'verify_code.json') # [{"time": int(time.time()), "user_id":str, "code":str}]
if not os.path.exists(config):
    with open(config, "w", encoding="utf-8") as fp:
        print("[]", file=fp)
```

- 改进结果

将所有相关接口合并为一个 AccountController 类

```

#!/usr/bin/env python3
# -*- coding: UTF-8 -*-
import string
from api.utils import *
from api import api_blue
from datetime import datetime, timedelta
from .send_verification_mail import send_email


class AccountController:
    def __init__(self):
        self.curpath = os.path.dirname(__file__)
        self.config = os.path.join(self.curpath,
                                  'verify_code.json') # [{"time": int(time.time()), "u
        if not os.path.exists(self.config):
            with open(self.config, "w", encoding="utf-8") as fp:
                print("[]", file=fp)

    def __judge_user_id(self, user_id: str):
        try:
            user_id = str(user_id).strip()
        except:
            return [400, "账号格式错误"]
        # if '@' not in user_id:
        #     user_id += "@tongji.edu.cn"
        pattern = re.compile(r'^\d{5,7}@tongji\.edu\.\cn$')
        result = pattern.findall(user_id)
        if len(result) > 0:
            return [0, "验证通过"]

```

- 测试结果

所有功能都运行正常

4. 对代码与规范的改进

改进 4.1 统一变量命名规范

- 改进原因

程序中变量命名规范不统一，在原因分析中可以看出程序中对于变量的命名规范并不统一，因此要进行改进。

- 改进结果

将代码中存在的量采取驼峰命名的变量名更改为下划线命名方法，如图所示

```

@item_blue.route('/content/<item_id>', methods=['GET', 'POST'])
def content(item_id: int): #goods_id/want_id
    try:
        item = Item.get(Item.id == item_id)
    except:
        return render_template('404.html', error_code=404, message="该商品不存在")
    else:
        if not current_user.is_authenticated:
            is_admin = False
            is_pub = False
        else:
            is_admin = (current_user.state == User_state.Admin.value)
            is_pub = (item.user_id.id == current_user.id)
            if item.state != Item_state.Sale.value and not is_admin and not is_pub:
                return render_template('404.html', error_code=401, message="该商品您现在无权访问")

```

● 测试结果

经过测试，更改变量名对系统无影响，可以正常运行。

改进 4.2 将一些常量保存到单独文件中

● 改进原因

在部分代码中存在着一些常量类，常量名等保存在路由配置文件中，将其统一保存到单独文件中使得代码结构更加清晰，同时使得验证等过程使用不会冲突

● 改进结果

将常量类等整理到单独的文件中，如下所示：

```

from enum import Enum, unique

#添加 unique 装饰器
@unique
class User_state(Enum):
    #用户状态: 0为普通用户, -1为封号, 1为管理员
    Normal = 0
    Admin = 1
    Under_ban = -1

    # -1 封号
    # 0 游客
    # 1 普通用户
    # 20 管理员
    # 999 系统管理员


@unique
class User_Campus_state(Enum):
    SiPing = "四平路校区"
    JiaDing = "嘉定校区"
    HuXi = "沪西校区"
    HuBei = "沪北校区"


@unique
class Gender(Enum):
    Male='男'
    Female='女'
    Undisclosed='保密'

```

- 测试结果

经过测试，整理后对于系统无影响，可以正常运行。

5. 对方法抽取的改进

改进 5.1 拆分 fake_data 方法

- 改进原因

fake_data 函数完成了对于不同数据库的初始化，其代码过于冗长(从 76 行到 321 行)，可读性比较差而且功能分割不够明确，可以按照数据库的种类将负责初始化不同数据库的部分抽取成更小的函数。

```
75
76     def fake_data():
77         User.create(id=80000000,
78                     username="系统管理员",
79                     state=1,
80                     password_hash=generate_password_hash("admin"),
81                     email="admin@admin.cn")
82
83         User.create(
84             id=1951705,
85             user_no="1951705",
86             username="高曾谊",
87             name="高曾谊",
88             state=1, #管理员
89             password_hash=generate_password_hash("1951705"),
90             email=str(1950084) + "@tongji.edu.cn")
91
92         User.create(id=1950084,
93                     user_no="1950084",
94                     username="陈泓仰",
95                     name="陈泓仰",
96                     state=1,
97                     password_hash=generate_password_hash("1950084"),
98                     email=str(1950084) + "@tongji.edu.cn")
99
100        User.create(id=1951566,
101                     user_no="1951566",
102                     username="贾仁军",
103                     name="贾仁军",
104                     state=1,
105                     password_hash=generate_password_hash("1951566"),
106                     email=str(1951566) + "@tongji.edu.cn")
107
108        User.create(
109             id=1953493,
110             user_no="1953493",
111             username="程森",
112             state=1, #被封号
113             password_hash=generate_password_hash("1953493"),
114             email=str(1953493) + "@tongji.edu.cn")
115
116        User.create(id=1952219,
117                     user_no="1952219",
118                     username="彭斐然",
119                     password_hash=generate_password_hash("1952219"),
120                     email=str(1952219) + "@tongji.edu.cn")
121
122        User.create(id=1951859,
123                     user_no="1951859",
124                     username="杨可盈",
125                     password_hash=generate_password_hash("1951859"),
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
```

- 改进结果

fake_data 函数在提取后如下，变得非常简洁清晰，可读性高

```
def fake_data():
    fake_data_user()
    fake_data_feedback()
    fake_data_userManagement()
    fake_data_item()
    fake_data_history()
    fake_data_favor()
    fake_data_contact()
    fake_data_review()
    fake_data_order()
    fake_data_orderState()
    fake_data_orderItem()
```

其中更小的子方法以下为例：

```

def fake_data_review(): #评价初始数据
    Review.create(id=1, user_id=1951705, feedback_content="默认好评")
    Review.create(id=2, user_id=1950084, feedback_content="默认好评")
    Review.create(id=3, user_id=1953493, feedback_content="默认好评")
    Review.create(id=4, user_id=1951566, feedback_content="默认好评")

```

- 测试结果

能够成功做到数据库的初始化

改进 5.2 从 change_item_state 和 change_item_num 中抽取方法

- 改进原因

这两个方法都是用来对于商品的某些属性做一些修改，其中有一些共通之处，其提取商品属性时的判断和流程是非常相似的，因此可以对其进行提取。

```

226     try:
227         data["item_id"] = int(data["item_id"])
228         data["state"] = int(data["state"])
229     except Exception as e:
230         return make_response_json(400, "请求格式不对")
231     try:
232         item = Item.get(Item.id == data["item_id"])
233     except Exception as e:
234         return make_response_json(404, "此商品不存在")
235
236     try:
237         data["item_id"] = int(data["item_id"])
238         data["num"] = int(data["num"])
239     except Exception as e:
240         return make_response_json(400, "请求格式不对")
241     if data["num"] < 0:
242         return make_response_json(400, "不允许负数物品存在")
243     if data["num"].bit_length() > Item.shelved_num.__sizeof__()-1:
244         return make_response_json(400, "数量越界")
245     try:
246         item = Item.get(Item.id == data["item_id"])
247     except Exception as e:
248         return make_response_json(404, "此商品不存在")
249
250     elif current_user.state == User_state.Under_ban.value:
251         return make_response_json(401, "您当前已被封号,请联系管理员解封")
252     else:
253         if current_user.id != item.user_id_id:
254             return make_response_json(401, "不可改变其他人的商品状态")
255         else:
256             if data["state"] == Item_state.Freeze.value:
257                 return make_response_json(401, "权限不足")
258             else:
259                 item.shelved_num = data["num"]
260                 item.save()
261                 return make_response_json(200, "操作成功")
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302

```

```

253     elif current_user.state == User_state.Under_ban.value:
254         return make_response_json(401, "您当前已被封号,请联系管理员解封")
255     else:
256         if current_user.id != item.user_id.id:
257             return make_response_json(401, "不可改变其他人的商品状态")
258         else:
259             if data["state"] == Item_state.Freeze.value:
260                 return make_response_json(401, "权限不足")
261             else:
262                 item.state = data["state"]
263                 item.save()
264             return make_response_json(200, "操作成功")
```

```

## ● 改进结果

提取获得商品属性并判断的方法

```

def getItem(data,name):
 try:
 data["item_id"] = int(data["item_id"])
 data[name] = int(data[name])
 except Exception as e:
 return (-1,400, "请求格式不对",NULL)
 try:
 item = Item.get(Item.id == data["item_id"])
 except Exception as e:
 return (-1,404, "此商品不存在",NULL)
 #全都成功
 return (0,0,"" ,item)
```

```

抽取操作商品属性的方法

```

def checkWrongItemOperation(current_user,data,item):
    if current_user.state == User_state.Under_ban.value:
        return (-1,401, "您当前已被封号,请联系管理员解封")
    else:
        if current_user.id != item.user_id.id:
            return (-1,401, "不可改变其他人的商品状态")
        else:
            if data["state"] == Item_state.Freeze.value:
                return (-1,401, "权限不足")
            else:
                return (0,0,"操作成功")
```

```

通过参数在调用时进行属性的区分

```

#读取item_id和state, 并获得item
result=getItem(data,"num")
if result[0]==-1:
 return make_response_json(result[1], result[2])
item=result[3]

#读取item_id和num, 并获得item
result=getItem(data,"num")
if result[0]==-1:
 return make_response_json(result[1], result[2])
item=result[3]
```

```

统一调用进行商品操作判断

```
    else:  
        ret=checkWrongItemOperation(current_user,data,item)  
        if ret[0]==-1:  
            return make_response_json(ret[1],ret[2])
```

- 测试结果

能够成功获取商品信息并操作

修改商品信息

商品名

软工作业22

商品单价

120

商品标签

工业品

▼

商品数量

2

商品描述

好

改进 5.3 从 post_item_info 和 change_item_data 中抽取方法

- 改进原因

这两个方法之中对于商品数据的判断存在着许多相似之处，虽然代码本身的写法有所区别，但实际上实现的功能是完全一致的，因此应该将这一部分全部抽取出来作为一个新的方法进行调用，这样既可以增强代码可读性，又提高了判断的一致性。

```

308     data = request.get_json()
309     if "tag" not in data:
310         |    return make_response_json(400, "请选择物品类型")
311     if data["tag"] not in Item_tag_type._value2member_map_:
312         |    return make_response_json(400, "请求格式错误")
313     if len(data["name"])>40:
314         |    return make_response_json(400,"名称过长")
315     if len(data["description"]) > Item.description.max_length:
316         |    return make_response_json(400,"描述过长")
317     try:
318         |    data["id"] = int(data["id"])
319         |    if "shelved_num" in data:
320             |        |    data["shelved_num"] = int(data["shelved_num"])
321         |    if "price" in data:
322             |        |    data["price"] = Float(data["price"])
323     except Exception as e:
324         |    return make_response_json(400, "请求格式不对")
325     if data["shelved_num"] < 0:
326         |    return make_response_json(400, "不允许负数商品个数")
327     if data["price"] == Float("inf") or data["price"] == Float("nan"):
328         |    return make_response_json(400, "价格越界")
329     if data["price"] <= 0:
330         |    return make_response_json(400, "不允许非正数价格")
331     if data["shelved_num"].bit_length() > Item.shelved_num.__sizeof__()-1:
332         |    return make_response_json(400, "数量越界")
333     try:
334         |    item = Item.get(Item.id == data["id"])
335     except Exception as e:
336         |    return make_response_json(404, "此商品不存在")

377     if len(data["name"])>40:
378         |    return make_response_json(400,"名称过长")
379     if len(data["description"]) > Item.description.max_length:
380         |    return make_response_json(400,f"描述过长,应限制在{Item.description.max_length}字以内")
381     if "tag" not in data:
382         |    return make_response_json(400, "请选择物品类型")
383     if data["tag"] not in Item_tag_type._value2member_map_:
384         |    return make_response_json(400, "请求格式错误")
385     try:
386         |    price = Float(data["price"])
387         |    item_type = int(data["type"])
388         |    shelved_num = int(data["shelved_num"])
389     except Exception as e:
390         |    return make_response_json(400, "数据类型错误")
391     if price == Float("inf") or price == Float("nan"):
392         |    return make_response_json(400, "价格越界")
393     if price <= 1e-8:
394         |    return make_response_json(400, "价格越界")
395     if item_type != Item_type.Goods.value and item_type != Item_type.Want.value:
396         |    return make_response_json(400, "仅能上传物品")
397     if shelved_num <= 0:
398         |    return make_response_json(400, "数量越界")
399     if shelved_num.bit_length() > Item.shelved_num.__sizeof__()-1:
400         |    return make_response_json(400, "数量越界")

```

● 改进结果

统一抽取检查商品属性的方法

```

def checkItemData(data):
    if len(data["name"])>40:
        return (-1,400,"名称过长")
    if len(data["description"]) > Item.description.max_length:
        return (-1,400,f"描述过长,应限制在{Item.description.max_length}字以内")
    if "tag" not in data:
        return (-1,400, "请选择物品类型")
    if data["tag"] not in Item_tag_type._value2member_map_:
        return (-1,400, "请求格式错误")
    try:
        price = Float(data["price"])
        if "type" in data:
            item_type = int(data["type"])
            shelved_num = int(data["shelved_num"])
    except Exception as e:
        return (-1,400, "数据类型错误")
        #return (-1,400, str(e))
    if price == Float("inf") or price == Float("nan"):
        return (-1,400, "价格越界")
    if price <= 1e-8:
        return (-1,400, "价格越界")
    if "type" in data:
        if item_type != Item_type.Goods.value and item_type != Item_type.Want.value:
            return (-1,400, "仅能上传物品")
    if shelved_num <= 0:
        return (-1,400, "数量越界")
    if shelved_num.bit_length() > Item.shelved_num.__sizeof__()-1:
        return (-1,400, "数量越界")
    return (0,0,"")

```

方法调用：

```

#检查商品格式
result=checkItemData(data)
if result[0]==-1:
    return make_response_json(result[1],result[2])
...

```

● 测试结果

能够判断各种商品的属性格式错误



改进 5.4 分拆 post_item_info 方法

● 改进原因

post_item_info 作为一个非常核心的方法，其长度较长，而且其中由很多不同功能组合而成，对于可读性和可维护性都提出了挑战，因此我决定将其按照功能分割并抽取函数。

```

370     @api_blue.route("/post_item_info", methods=["POST"])
371     def post_item_info():
372         if not current_user.is_authenticated:
373             return make_response_json(401, "当前用户未登录")
374         if current_user.state == User_state.Under_ban.value:
375             return make_response_json(401, "当前用户已被封号")
376         data = request.get_json()
377         if len(data["name"])>40:
378             return make_response_json(400, "名称过长")
379         if len(data["description"]) > Item.description.max_length:
380             return make_response_json(400, f"描述过长, 应限制在{Item.description.max_length}字以内")
381         if "tag" not in data:
382             return make_response_json(400, "请选择物品类型")
383         if data["tag"] not in Item.tag_type_value2member_map_:
384             return make_response_json(400, "请求格式错误")
385         try:
386             price = float(data["price"])
387             item_type = int(data["type"])
388             shelfed_num = int(data["shelfed_num"])
389         except Exception as e:
390             return make_response_json(400, "数据类型错误")
391         if price == float("inf") or price == float("nan"):
392             return make_response_json(400, "价格越界")
393         if price <= 0:
394             return make_response_json(400, "价格越界")
395         if item_type != Item_type.Goods.value and item_type != Item_type.Want.value:
396             return make_response_json(400, "仅能上传物品")
397         if shelfed_num <= 0:
398             return make_response_json(400, "数量越界")
399         if shelfed_num.bit_length() > Item.shelfed_num._sizeof_-1:
400             return make_response_json(400, "数量越界")
401         data["user_id"] = current_user.id
402         data["publish_time"] = datetime.now()
403         try:
404             new = Item.create(**data)
405         except Exception as e:
406             os.path.join(item_blue.static_folder,
407                         f'resource/item_pic/{new.id}/head'))
408             default_pic = os.path.join(item_blue.static_folder,
409                                     'resource/default_pic/test.jpg')
410             curpath = os.path.join(item_blue.static_folder,
411                                   f'resource/item_pic/{new.id}/')
412             tempath = os.path.join(item_blue.static_folder, f'resource/temp/')
413             default_pic = os.path.join(item_blue.static_folder,
414                                         f'resource/item_pic/{new.id}/pic')
415             default_pic = os.path.join(item_blue.static_folder,
416                                         'resource/default_pic/test.jpg')
417             curpath = os.path.join(item_blue.static_folder,
418                                   f'resource/item_pic/{new.id}/')
419             tempath = os.path.join(item_blue.static_folder, f'resource/temp/')
420             if len(data["urls"]) == 0:
421                 #给一个默认图
422                 shutil.copy(default_pic, os.path.join(curpath, 'head/'))
423                 shutil.copy(default_pic, os.path.join(curpath, 'pic/'))
424             else:
425                 head_pics = [i["MD5"] for i in data["urls"] if i["is_cover_pic"]]
426                 if len(head_pics) == 0:
427                     head_pic = data["urls"][0]["MD5"]
428                 elif len(head_pics) > 1:
429                     new_delete_instance()
430                     #return make_response_json(400, "仅能选定一张头图")
431                     head_pic = head_pics[0]
432                 else:
433                     head_pic = head_pics[0]
434                 shutil.copy(os.path.join(tempath, head_pic),
435                             os.path.join(curpath, 'head/'))
436                 img = Image.open(os.path.join(curpath, 'head/'))
437                 img = trans_square(img)
438                 img.show()
439                 img.save(os.path.join(curpath, 'head/'), 'WEBP')
440             for j in data["urls"]:
441                 shutil.move(os.path.join(tempath, j["MD5"]),
442                             os.path.join(curpath, 'pic/'))
443             #将所有的图片转到用户对应文件夹
444             return make_response_json(200, "上传成功",
445                                     {"url": url_for('item.content', item_id=new.id)})
446         except Exception as e:
447             print(e)
448

```

● 改进结果

除了之前提到的判断商品属性格式的函数，我还抽取了创建本地缓存路径和上传图片的函数：

```

def createPicPath(new):
    createPath(
        os.path.join(item_blue.static_folder,
                    f'resource/item_pic/{new.id}/head'))
    createPath(
        os.path.join(item_blue.static_folder,
                    f'resource/item_pic/{new.id}/pic'))
    default_pic = os.path.join(item_blue.static_folder,
                               'resource/default_pic/test.jpg')
    curpath = os.path.join(item_blue.static_folder,
                          f'resource/item_pic/{new.id}/')
    tempath = os.path.join(item_blue.static_folder, f'resource/temp/')
    return default_pic, curpath, tempath

```

```

def uploadpic(data,default_pic,curpath,tempath):
    if len(data["urls"]) == 0:
        #给一个默认图
        shutil.copy(default_pic, os.path.join(curpath, 'head/'))
        shutil.copy(default_pic, os.path.join(curpath, 'pic/'))
    else:
        head_pics = [i["MD5"] for i in data["urls"] if i["is_cover_pic"]]
        if len(head_pics) == 0:
            head_pic = data["urls"][0]["MD5"]
        elif len(head_pics) > 1:
            #new.delete_instance()
            #return make_response_json(400, "仅能选定一张头图")
            head_pic = head_pics[0]
        else:
            head_pic = head_pics[0]
        shutil.copy(os.path.join(tempath, head_pic),
                    os.path.join(curpath, 'head/'))

        img = Image.open(os.path.join(curpath, 'head/', head_pic))
        img = trans_square(img)
        #img.show()
        img.save(os.path.join(curpath, 'head/', head_pic), 'WEBP')

    for j in data["urls"]:
        shutil.move(os.path.join(tempath, j["MD5"]),
                    os.path.join(curpath, 'pic/'))

```

这些更小的方法最终组合成新的 post_item_info 函数，不难看出，其整体比之前清晰明了很多：

```

data = request.get_json()
#检查商品格式
result=checkItemData(data)
if result[0]==-1:
    return make_response_json(result[1], result[2])
#继续添加信息
data["user_id"] = current_user.id
data["publish_time"] = datetime.now()
#创建item
try:
    new = Item.create(**data)
except Exception as e:
    return make_response_json(500, f"上传失败: {repr(e)}")
#创建图片目录
default_pic,curpath,tempath=createPicPath(new)
#上传图片
uploadpic(data,default_pic,curpath,tempath)
#将所有的图片转到用户对应文件夹
return make_response_json(200, "上传成功",
                           {"url": url_for('item.content', item_id=new.id)})

```

● 测试结果

可以成功创建新的商品

商品测试

卖家: 1952893 | 物品ID: 8

¥ 120 - 需要2件

好

发布类型: 悬赏
发布时间: 2022-08-15 01:25:05

修改商品信息

下架商品

加入收藏

TAG: 工业品

商品状态: 正常发布

改进 5.5 从 get_pillow_img_form_data_stream 和 save_pic 中抽取方法

● 改进原因

这两个方法同为上传图片的操作，其中有很多部分是重复和相似的，而保存图片的操作又经常使用，从中抽出新的方法较为合适。

```
def get_pillow_img_form_data_stream(data):
    ...
    传入request.files.get('something') (data类型为werkzeug.filestorage)
    将图片读取后按WEBP转换，保存入临时图标文件夹
    最后返回{400, 失败}或{200, [md5(str)]}
    ...
try:
    #os.path.join(item_blue.static_folder, f'resource/temp')
    #或
    #url_for('item.static', filename=f'resource/item_pic/{item_id}/{head|pic}')
    curpath = os.path.join(item_blue.static_folder, f'resource/temp')
    createPath(curpath)

    path_name = os.path.join(curpath, data.filename)
    createPath(curpath)
    data.save(path_name)

    img = Image.open(path_name)
    w, h = img.size
    ratio = max(w, h) / 1920
    if ratio > 1:
        img = img.resize((int(w / ratio), int(h / ratio)))
    ratio = 250 / min(w, h)
    if ratio > 1:
        img = img.resize((int(w * ratio), int(h * ratio)))
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    if os.path.exists(path_name_new):
        #    return make_response_json(400, f"上传图片失败: 请勿重复上传图片")
        img.save(path_name_new, 'WEBP')
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name_new)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    #if os.path.exists(path_name_new):
    #    return make_response_json(400, f"上传图片失败: 请勿重复上传图片")
    img.save(path_name_new, 'WEBP')
except Exception as e:
    print(e)
    return make_response_json(400, f"上传图片失败: 文件格式错误或损坏")
else:
    return make_response_json(200, "上传图片成功", md5_str)
```

```
def save_pic(path,data):
    ...
    传入request.files.get('something') (data类型为werkzeug.filestorage)
    将图片读取后按WEBP转换，保存入临时图标文件夹
    最后返回{400, 失败}或{200, [md5(str)]}
    ...
try:
    #os.path.join(item_blue.static_folder, f'resource/temp')
    #或
    #url_for('item.static', filename=f'resource/item_pic/{item_id}/{head|pic}')
    curpath = path
    createPath(curpath)

    path_name = os.path.join(curpath, data.filename)
    createPath(curpath)
    data.save(path_name)
    img = Image.open(path_name)
    w, h = img.size
    ratio = max(w, h) / 1920
    if ratio > 1:
        img = img.resize((int(w / ratio), int(h / ratio)))
    ratio = 250 / min(w, h)
    if ratio > 1:
        img = img.resize((int(w * ratio), int(h * ratio)))
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    img.save(path_name_new, 'WEBP')
    img = Image.open(path_name_new)
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name_new)

    path_name_new = os.path.join(curpath, f'{md5_str}')
    #if os.path.exists(path_name_new):
    #    return make_response_json(400, f"上传图片失败: 请勿重复上传图片")
    img.save(path_name_new, 'WEBP')
except Exception as e:
    print(e)
    return make_response_json(400, f"上传图片失败: 文件格式错误或损坏")
else:
    return make_response_json(200, "上传图片成功", md5_str)
```

● 改进结果

首先把创建路径的函数提取到更外层的 utils 中

```
def createPath(path: str) -> None:
    # 当前路径不存在，直接建立文件夹
    if not os.path.exists(path):
        os.makedirs(path)
    # 路径存在但不是文件夹，将其删除后创建新的文件夹
    elif not os.path.isdir(path):
        os.remove(path)
        os.makedirs(path)
```

然后抽取出保存图片的方法 savePic

```
def savePic(data, curpath):
    path_name = os.path.join(curpath, data.filename)
    createPath(curpath)
    data.save(path_name)
    img = Image.open(path_name)
    w, h = img.size
    # 长宽中较大的/1920作为比率
    ratio = max(w, h) / 1920
    # 比率大于1，按照比率缩放，使得最大的到达1920
    if ratio > 1:
        img = img.resize((int(w / ratio), int(h / ratio)))
    ratio = 250 / min(w, h)
    # 按比例缩放使得图片长宽较小者大于等于250
    if ratio > 1:
        img = img.resize((int(w * ratio), int(h * ratio)))
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name)
    # 创建新的图片路径并保存
    path_name_new = os.path.join(curpath, f'{md5_str}')
    img.save(path_name_new, 'WEBP')
    img = Image.open(path_name_new)
    md5_str = md5(img.tobytes()).hexdigest()
    os.remove(path_name_new)

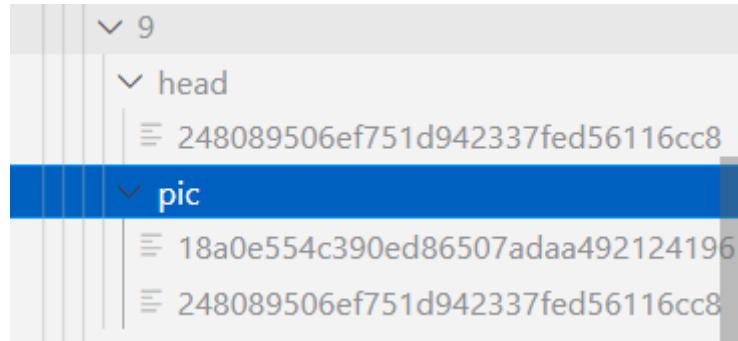
    path_name_new = os.path.join(curpath, f'{md5_str}')
    #if os.path.exists(path_name_new):
    #    return make_response_json(400, "上传图片失败：请勿重复上传图片")
    img.save(path_name_new, 'WEBP')
    return md5_str
```

由此 get_pillow_img_form_data_stream 和 save_pic 都能通过这样的统一调用来完成保存图片的操作

```
# 保存图片
md5_str=savePic(data,curpath)
```

● 测试结果

能够成功保存图片



改进 5.6 从 get_favor 和 get_history 中抽取方法

● 改进原因

获取历史和获取收藏有很类似的地方，除了部分属性存在差别以外，整体流程较为统一，因此可以将其提取出新的方法，并通过传参的方式实现其中细微的差别

```

@api_blue.route("/get_favor", methods=["GET"])
def get_favor():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    req = dict(request.args)
    if "range_min" in req or "range_max" in req:
        if "range_min" in req and "range_max" in req:
            try:
                range_min = int(req["range_min"])
                range_max = int(req["range_max"])
            except Exception as e:
                return make_response_json(400, "请求格式错误")
        else:
            return make_response_json("请求格式错误")
    else:
        range_min = 0
        range_max = 50
    fav = Favor.select().where(Favor.user_id == current_user.id).order_by(
        Favor.collect_time.desc())
    fav_data = []
    for i in fav:
        res = dict()
        res['id'] = i.id
        res['item_id'] = i.item_id.id
        res['collect_time'] = str(i.collect_time)
        fav_data.append(res)
    range_max = min(len(fav_data), range_max)
    data = {
        "total_count": len(fav_data),
        "favor_list": fav_data[range_min:range_max]
    }
    return make_response_json(200, "操作成功", data)

@api_blue.route("/get_history", methods=["GET"])
def get_history():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    req = dict(request.args)
    if "range_min" in req or "range_max" in req:
        if "range_min" in req and "range_max" in req:
            try:
                range_min = int(req["range_min"])
                range_max = int(req["range_max"])
            except Exception as e:
                return make_response_json(400, "请求格式错误")
        else:
            return make_response_json("请求格式错误")
    else:
        range_min = 0
        range_max = 50
    history = History.select().where(History.user_id == current_user.id).order_by(
        History.visit_time.desc())
    his_data = []
    for i in history:
        res = dict()
        res['id'] = i.id
        res['item_id'] = i.item_id.id
        res['visit_time'] = str(i.visit_time)
        his_data.append(res)
    range_max = min(len(his_data), range_max)
    data = {
        "total_count": len(his_data),
        "history_list": his_data[range_min:range_max]
    }
    return make_response_json(200, "操作成功", data)

```

● 改进结果

首先在方法内部，再分割抽取出两个小的方法用来获取列表的范围和列表本身

```

def getRange(req):
    if "range_min" in req or "range_max" in req:
        if "range_min" in req and "range_max" in req:
            try:
                range_min = int(req["range_min"])
                range_max = int(req["range_max"])
            except Exception as e:
                return (-1,400, "请求格式错误")
        else:
            return (-1,400, "请求格式错误")
    else:
        range_min = 0
        range_max = 50
    return (0,range_min,range_max)

```

```

def getFSList(myclass,listname):
    if listname=="favor_list":
        tep = myclass.select().where(myclass.user_id == current_user.id).order_by(
            myclass.collect_time.desc())
    else:
        tep = myclass.select().where(myclass.user_id == current_user.id).order_by(
            myclass.visit_time.desc())
    fav_data = []
    for i in tep:
        res = dict()
        res['id'] = i.id
        res['item_id'] = i.item_id.id
        if listname=="favor_list":
            res['collect_time'] = str(i.collect_time)
        else:
            res['visit_time'] = str(i.visit_time)
        fav_data.append(res)
    return fav_data

```

然后再将这两个方法运用到新抽取出的方法中

```

def getFS(myclass,req,listname):
    #取范围
    result=getRange(req)
    if result[0]==-1:
        return (-1,result[1], result[2])
    range_min=result[1]
    range_max=result[2]
    #添加data
    fav_data=getFSList(myclass,listname)
    data = {
        "total_count": len(fav_data),
        listname: fav_data[range_min:range_max]
    }
    return (0,data)

```

在原函数中可以使用参数按需调用

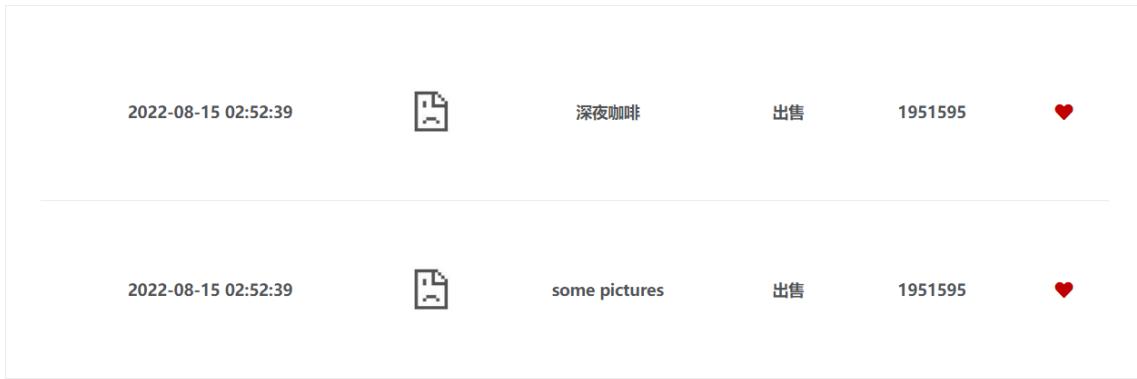
```

result=getFS(Favor,req,"favor_list")
result=getFS(History,req,"history_list")

```

- 测试结果

能够获取到收藏列表和历史记录列表



改进 5.7 从 del_favor 和 item_delete_history 中抽取方法

● 改进原因

删除历史和删除收藏有很类似的地方，除了部分属性存在差别以外，整体流程较为统一，因此可以将其提取出新的方法，并通过传参的方式实现其中细微的差别

● 改进结果

从中抽取出删除记录的统一方法

```
def delFS(myclass,req):
    try:
        NotFound = False
        for i in req:
            tep = myclass.select().where((myclass.user_id == current_user.id)
                                         & (myclass.item_id == i))

            if tep.count() <= 0:
                NotFound = True
            else:
                myclass.delete().where((myclass.user_id == current_user.id)
                                         & (myclass.item_id == i)).execute()

        if NotFound == True:
            if myclass==History:
                return (404, "不存在对应的历史")
            else:
                return (404, "不存在对应的收藏")
        return (200, "删除成功")
    except Exception as e:
        return(500, f"发生错误 {repr(e)}")
```

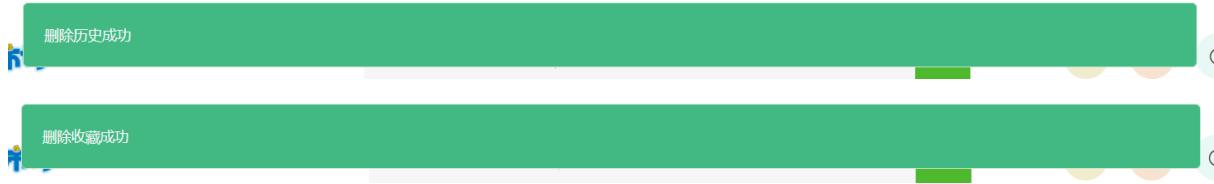
原来的方法中对其进行调用并传入适当的参数

```
#删除
retid,retinfo=delFS(Favor,req)

#删除
retid,retinfo=delFS(History,req)
```

● 测试结果

可以进行收藏/历史记录的删除



改进 5.8 分割抽取 change_order_state 方法

● 改进原因

`change_order_state` 函数是一个非常核心的操作订单的方法，其中流程较为复杂，且有多个角色都能够参与其中，导致了代码较为复杂，可读性和可维护性都比较差。经过仔细研究和分析，我决定站在实际应用中会调用到这个方法的不同身份用户（管理员、订单发起方、商品提供方）的角度对其进行分割。

● 改进结果

把管理员、订单发起方、商品提供方的操作分别提取成方法供原方法调用

```
elif current_user.state == User_state.Admin.value: #管理员,无限权力  
    retid,retinfo=change_order_administrator(req_state,order,_order_item,order_user_id,order_op_user_id)
```

```

    elif current_user.id == order_user_id: #当前用户是订单发起者
        retid,retinfo=change_order_a(req_state,order,_order_item,order_op_user_id)
        return make_response_json(retid, retinfo)

    elif current_user.id == order_op_user_id: #当前用户是物品发布者（非订单发起者）
        retid,retinfo=change_order_b(req_state,order,_order_item,order_user_id,order_op_user_id)
        return make_response_json(retid, retinfo)

```

● 测试结果

能够成功更改订单状态



改进 5.9 从 get_item_pic 与 get_item_head_pic 中提取方法

● 改进原因

源代码中 get_item_pic 函数与 get_item_head_pic 函数有大量重复部分，经过分析，两函数中区别仅在于获取图片的位置以及提取图片的数量。

故使用参数化函数的重构方法，通过使用不同参数来调用两种方法，减少大量代码重复。

<pre> def get_item_head_pic(): data = dict(request.args) try: item_id = int(data["item_id"]) except Exception as e: return make_response_json(400, f"请求格式错误 {repr(e)}") try: item = Item.get(Item.id == item_id) except Exception as e: return make_response_json(400, "此物品不存在") pic_path = os.path.join(item_blue.static_folder, f'resource/item_pic/{item_id}/head') default_pic = os.path.join(item_blue.static_folder, 'resource/default_pic/test.jpg') if not os.path.exists(pic_path): createPath(pic_path) if len(os.listdir(pic_path)) == 0: shutil.copy(default_pic, pic_path) pic_list = os.listdir(pic_path) pic = url_for('item.static', filename=f'resource/item_pic/{item_id}/head/{pic_list[0]}') return make_response_json(200, "图片查找成功", data={"url": pic}) </pre>	<pre> def get_item_pics(): data = dict(request.args) try: item_id = int(data["item_id"]) except Exception as e: return make_response_json(400, f"请求格式错误 {repr(e)}") try: item = Item.get(Item.id == item_id) except Exception as e: return make_response_json(400, "此物品不存在") pic_path = os.path.join(item_blue.static_folder, f'resource/item_pic/{item_id}/pic') default_pic = os.path.join(item_blue.static_folder, 'resource/default_pic/test.jpg') if not os.path.exists(pic_path): createPath(pic_path) if len(os.listdir(pic_path)) == 0: shutil.copy(default_pic, pic_path) pic_list = os.listdir(pic_path) pics = list() for pic_name in pic_list: pics.append(</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

● 改进结果

将两个函数提取为一个函数，并使用不同参数来执行不同功能，具体如下图所示：

```
# 接口/get_item_pics与/get_item_head_pic的公共函数
def get_pic(data, select_pics):
    try:
        item_id = int(data["item_id"])
    except Exception as e:
        return make_response_json(400, f"请求格式错误 {repr(e)}")
    try:
        item = Item.get(Item.id == item_id)
    except Exception as e:
        return make_response_json(400, "此物品不存在")
    pic_path = ''
    if select_pics:
        pic_path = os.path.join(item_blue.static_folder,
                               f'resource/item_pic/{item_id}/pic')
    else:
        pic_path = os.path.join(item_blue.static_folder,
                               f'resource/item_pic/{item_id}/head')
    default_pic = os.path.join(item_blue.static_folder,
                               'resource/default_pic/test.jpg')
    if not os.path.exists(pic_path):
        createPath(pic_path)
    if len(os.listdir(pic_path)) == 0:
        shutil.copy(default_pic, pic_path)
    pic_list = os.listdir(pic_path)

    if select_pics:
        pics = list()
        for pic_name in pic_list:
            pics.append(
                url_for(
                    'item.static',
                    filename=f'resource/item_pic/{item_id}/pic/{pic_name}'))
    return make_response_json(200, "图片查找成功", data={"url": pics})
    else:
        pic = url_for(
            'item.static',
            filename=f'resource/item_pic/{item_id}/head/{pic_list[0]}')
    return make_response_json(200, "图片查找成功", data={"url": pic})
```

两个接口调用函数时如下图所示：

```
@api_blue.route("/get_item_pics", methods=['GET'])
def get_item_pics():
    data = dict(request.args)
    return get_pic(data, True)

@api_blue.route("/get_item_head_pic", methods=['GET'])
def get_item_head_pic():
    data = dict(request.args)
    return get_pic(data, False)
```

● 测试结果

经过测试，获取商品图片功能正确，用户可以正确地获取商品的头图与所有图片。



some pictures

卖家: 1951595 | 物品ID: 6

¥ 5 - 剩余500件

很美

- 发布类型: 售卖
- 发布时间: 2022-08-15 00:07:16

[修改商品信息](#)

[下架商品](#)

[加入收藏](#)

TAG: 图书/文娱/教育/电子书

商品状态: 正常发布

改进 5.10 分拆 get_search 方法

● 改进原因

源代码中 get_search 函数过长，代码可读性较差，功能杂乱且不突出。

细读源代码后，该函数主要可以分为三个小功能：请求数据检查、搜索依据添加、进行搜索并排序输出。

故使用提取函数的重构方法，提取三个独立功能的小函数，使 get_search 方法调用三个函数。

```

def get_search():
    data = request.get_json()
    if "range_min" in data or "range_max" in data:
        if "range_min" in data and "range_max" in data:
            try:
                range_min = int(data["range_min"])
                range_max = int(data["range_max"])
            except Exception as e:
                return make_response_json(400, "请求格式错误")
        else:
            return make_response_json(400, "请求格式错误")
    else:
        range_min = 0
        range_max = 50
    try:
        search_type = int(data["search_type"])
    except Exception as e:
        return make_response_json(400, "请求格式错误")
    if "key_word" in data:
        key_word = data["key_word"]
    else:
        return make_response_json(400, "请输入关键词")
    if "order_type" in data:
        order_type = data["order_type"]
    else:
        order_type = "name"
    #get_data = Item.select().where("select_need").execute()
    need = (Item.id, Item.name, Item.user_id, Item.publish_time, Item.price,
            Item.tag)
    select_need = [Item.name.contains(key_word)]
    if search_type in Item_type._value2member_map_:
        select_need.append(Item.type == search_type)
    if "tag" in data:
        if data["tag"] not in Item_tag_type._value2member_map_:
            return make_response_json(400, "请求格式错误")
        else:
            select_need.append(Item.tag == data["tag"])
    try:
        if "start_time" in data:
            start_time = data["start_time"]
            if start_time != "" and start_time is not None:
                start_time = datetime.strptime(start_time, "%Y-%m-%d")
                select_need.append(Item.publish_time >= start_time)
        if "end_time" in data:
            end_time = data["end_time"]
            if end_time != "" and end_time is not None:
                end_time = datetime.strptime(end_time, "%Y-%m-%d")
                select_need.append(Item.publish_time <= end_time)
    except Exception as e:
        print(e)
    return make_response_json(400, "时间格式错误,应为年-月-日格式")
    get_data = Item.select(*need).where(*select_need).execute()
    datas = [i._data_ for i in get_data]
    new_data = list()
    if order_type == "time":
        datas.sort(key=lambda x: x["publish_time"], reverse=True)
    elif order_type == "price":
        datas.sort(key=lambda x: x["price"], reverse=False)
    else:
        #orderWay = (Item.publish_time.desc(),) # 误: 默认为倒叙
        datas.sort(
            key=lambda x: SequenceMatcher(a=key_word, b=x["name"]).ratio(),
            reverse=True)
    for i in datas:
        if 'price' != float(i['price']):
            if 'publish_time' != str(i['publish_time']):
                new_data.append(i)
    range_max = min(len(new_data), range_max)
    final_data = {
        "total_count": len(new_data),
        "item_list": new_data[range_min:range_max]
    }
    return make_response_json(200, "搜索结果如下", data=final_data)

```

● 改进结果

经过函数分拆，实现了三个子方法：

1. 请求数据检查

```

def data_check(data): #请求数据检查
    if "range_min" in data or "range_max" in data:
        if "range_min" in data and "range_max" in data:
            range_min = int(data["range_min"])
            range_max = int(data["range_max"])
        else:
            raise Exception('请求格式错误')
    else:
        range_min = 0
        range_max = 50

    search_type = int(data["search_type"])

    if "key_word" in data:
        key_word = data["key_word"]
    else:
        raise Exception('请输入关键词')

    if "order_type" in data:
        order_type = data["order_type"]
    else:
        order_type = "name"

    return range_min, range_max, search_type, key_word, order_type

```

2. 搜索依据添加

```

def select_need_append(key_word, search_type, data): #搜索依据添加
    select_need = [Item.name.contains(key_word)]
    if search_type in Item_type._value2member_map_:
        select_need.append(Item.type == search_type)
    if "tag" in data:
        if data["tag"] not in Item_tag_type._value2member_map_:
            raise Exception('请求格式错误')
        else:
            select_need.append(Item.tag == data["tag"])

    if "start_time" in data:
        start_time = data["start_time"]
        if start_time != "" and start_time is not None:
            start_time = datetime.strptime(start_time, "%Y-%m-%d")
            select_need.append(Item.publish_time >= start_time)
    if "end_time" in data:
        end_time = data["end_time"]
        if end_time != "" and end_time is not None:
            end_time = datetime.strptime(end_time, "%Y-%m-%d")
            select_need.append(Item.publish_time <= end_time)

```

3. 搜索并输出

```

def search(need, select_need, order_type, key_word, range_min, range_max): #实现搜索结果
    get_data = Item.select(*need).where(*select_need).execute()
    datas = [i._data_ for i in get_data]
    new_data = list()
    if order_type == "time":
        datas.sort(key=lambda x: x["publish_time"], reverse=True)
    elif order_type == "price":
        datas.sort(key=lambda x: x["price"], reverse=False)
    else:
        #orderWay = (Item.publish_time.desc(), ) # 改: 默认其实为相似度
        datas.sort(
            key=lambda x: SequenceMatcher(a=key_word, b=x["name"]).ratio(),
            reverse=True)
    for i in datas:
        i['price'] = float(i['price'])
        i['publish_time'] = str(i['publish_time'])
        new_data.append(i)
    range_max = min(len(new_data), range_max)
    final_data = {
        "total_count": len(new_data),
        "item_list": new_data[range_min:range_max]
    }
    return make_response_json(200, "搜索结果如下", data=final_data)

```

最后调用如下：

```

@api_blue.route('/search', methods=['POST'])
def get_search():
    data = request.get_json()
    try:
        range_min, range_max, search_type, key_word, order_type = data_check(data)
        need = (Item.id, Item.name, Item.user_id, Item.publish_time, Item.price,
                Item.tag)
        select_need = select_need_append(key_word, search_type, data)
    except:
        return make_response_json(400, "请求格式错误")
    else:
        return search(need, select_need, order_type, key_word, range_min, range_max)

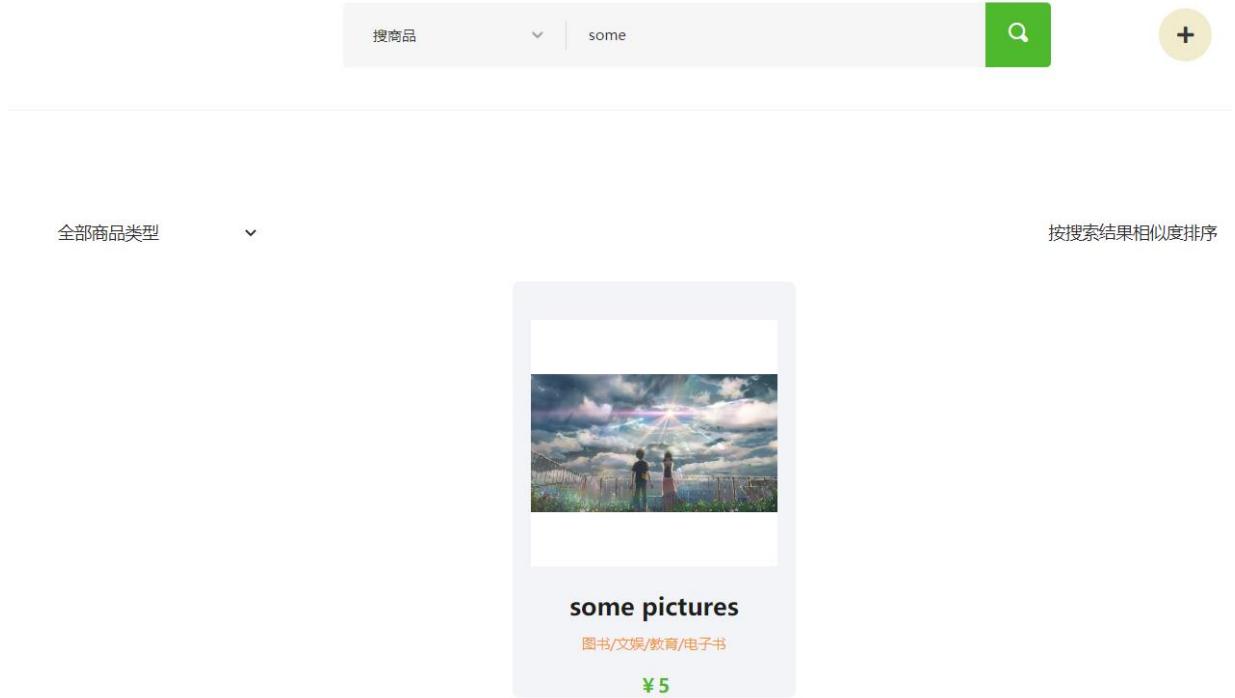
```

可以看到，经过改进后的方法更加清晰明了。

● 测试结果

经过多种搜索测试，搜索功能完好，后端正确地调用了接口，前端正确地返回了搜索结果。

```
(9140) accepted ('127.0.0.1', 61288)
127.0.0.1 - - [19/Aug/2022 10:57:22] "POST /api/search HTTP/1.1" 200 509 0.152950
127.0.0.1 - - [19/Aug/2022 10:57:22] "GET /api/get_class HTTP/1.1" 200 897 0.118176
127.0.0.1 - - [19/Aug/2022 10:57:23] "GET /api/get_item_head_pic?item_id=6 HTTP/1.1" 200 329 0.166526
127.0.0.1 - - [19/Aug/2022 10:57:23] "GET /item/static/resource/item_pic/6/head/d444fc08387700a53f2de445287ad607 HTTP/1.1" 304 234 0
127.0.0.1 - - [19/Aug/2022 10:57:26] "GET /api/get_message_cnt HTTP/1.1" 200 267 0.162637
```



改进 5.11 从 order_post 中提取方法

- 改进原因

源代码中 order_post 函数中有一小部分代码重复出现，执行多次。

故使用提取函数的重构方法，提出重复多次的代码为单独函数，减少大量代码重复。

```

if call_back[0] != 200:
    for t in range(start_num):
        item_list[t].shelved_num += data["item_info"][t]["num"]
        item_list[t].locked_num -= data["item_info"][t]["num"]
        item_list[t].save()
    return make_response_json(call_back[0], call_back[1])
try:
    contact = Contact.get(Contact.id == contact_id)
except Exception as e:
    for t in range(len(data["item_info"])):
        item_list[t].shelved_num += data["item_info"][t]["num"]
        item_list[t].locked_num -= data["item_info"][t]["num"]
        item_list[t].save()

```

```

except Exception as e:
    for t in range(len(data["item_info"])):
        item_list[t].shelved_num += data["item_info"][t]["num"]
        item_list[t].locked_num -= data["item_info"][t]["num"]
        item_list[t].save()
    for j in od_it_list:
        j.delete_instance()
    od_st_it.delete_instance()
    od.delete_instance()
    return make_response_json(500, f"订单详情存储时出错 {repr(e)}")
od_it_list.append(od_it)
od_it.item_id.type == Item_type.Goods.value:
send_message(SYS_ADMIN_NO, od_it.item_id.user_id.id,
f"已有用户购买你的商品<{od_it.item_id.name}>, 请前往个人中心确认或取消订单")

```

- 改进结果

将重复部分提取为函数，具体如下图所示：

```

#接口order_post函数公共部分
def item_list_save(data, item_list, len):
    for t in range(len):
        item_list[t].shelved_num += data["item_info"][t]["num"]
        item_list[t].locked_num -= data["item_info"][t]["num"]
        item_list[t].save()

```

提取 item_list_save() 函数，然后在原方法中进行以下调用：

```

except Exception as e:
    item_list_save(data, item_list, len(data["item_info"]))
    ...
for t in range(len(data["item_info"])):
    item_list[t].shelved_num += data["item_info"][t]["num"]
    item_list[t].locked_num -= data["item_info"][t]["num"]
    item_list[t].save()
...
return make_response_json(500, f"Order存储错误 {repr(e)}")
try:
    od_st_it = Order_State_Item.create(order_id=od.id)
except Exception as e:
    item_list_save(data, item_list, len(data["item_info"]))
    ...
for t in range(len(data["item_info"])):
    item_list[t].shelved_num += data["item_info"][t]["num"]
    item_list[t].locked_num -= data["item_info"][t]["num"]
    item_list[t].save()
...
od.delete_instance()
return make_response_json(500, f"Order_State_Item 存储错误 {repr(e)}")
od_it_list = list()
for i in range(len(item_list)):
    try:
        od_it = Order_Item.create(order_id=od.id,
                                   quantity=num,
                                   price=item_list[i].price,
                                   item_id=item_list[i].id)
    except Exception as e:
        item_list_save(data, item_list, len(data["item_info"]))
if call_back[0] != 200:
    item_list_save(data, item_list, start_num)
    ...
for t in range(start_num):
    item_list[t].shelved_num += data["item_info"][t]["num"]
    item_list[t].locked_num -= data["item_info"][t]["num"]
    item_list[t].save()
    ...
return make_response_json(call_back[0], call_back[1])
try:
    contact = Contact.get(Contact.id == contact_id)
except Exception as e:
    item_list_save(data, item_list, len(data["item_info"]))
    ...
for t in range(len(data["item_info"])):
    item_list[t].shelved_num += data["item_info"][t]["num"]
    item_list[t].locked_num -= data["item_info"][t]["num"]
    item_list[t].save()
    ...
return make_response_json(404, "不存在的地址信息")

```

● 测试结果

经过测试，提交订单的功能正确，可以正确提交订单，对于出错等一系列情况也没有问题。

The screenshot shows a user interface for generating an order. At the top, there is a red error message box containing the text '地址未填写' (Address not filled). Below this, there is a navigation bar with links '返回商品页' (Return to商品页) and '生成订单' (Generate Order). The main area contains two tables: one for product details and one for confirming the order.

商品/悬赏	单价	数量	总价
	¥100.0	5	¥500.00

确认订单

确认订单	
地址	
点击前往地址管理	
备注	请输入你的备注...
总价	¥500.00
提交	

改进 5.12 分拆 address 方法

● 改进原因

源代码中 address 方法通过 if-else 语句集成了在三种不同请求方法下的执行过程，由于每种请求方式对应的函数代码不同，故使得函数较长。

现通过三种不同的请求方式，独立完成三个方法，使得三个方法公用同一个接口，并带上不同请求方式。

```

@api_blue.route("/address", methods=["POST", "PUT", "DELETE"])
def address():
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    data = request.get_json()
    temp = [None for i in range(len(data))]
    if request.method == "DELETE":
        data = list(map(lambda x: x["contact_id"], data))
        for i, j in enumerate(data):
            try:
                temp[i] = Contact.get(Contact.id == int(j))
            except Exception as e:
                return make_response_json(401, "不存在的联络地址")
            else:
                if temp[i].user_id.id != current_user.id:
                    return make_response_json(401, "不可删除其他用户的联络地址")
        delete_default = False
        for i, j in enumerate(temp):
            try:
                j.delete_instance()
            except Exception as e:
                for t in range(5):
                    temp[t].save()
            return make_response_json(500, f"发生错误 {repr(e)}")
        else:
            if j.default:
                delete_default = True
    if delete_default:
        try:
            datas = Contact.select().where(
                Contact.user_id == current_user.id).order_by(
                    Contact.id.desc()).execute()
        except Exception as e:
            print(repr(e))
        else:
            if len(datas) > 0:
                datas[0].default = True
                datas[0].save()
    elif request.method == "PUT":
        for i, j in enumerate(data):
            if j["id"] == None:
                j.pop("contact_id")
            else:
                if isinstance(j[1], str) and len(j[1]) == 0:
                    return make_response_json(400, "不允许提交空参数")
                if "campus_branch" in j:
                    if User_Campus_state._value2member_map_[j["campus_branch"]] not in User_Campus_state._value2member_map_:
                        return make_response_json(400, "校园填写错误")
                try:
                    if j[1] == "Contact.default":
                        j.pop("contact_id")
                    else:
                        Contact.create(**j)
                except Exception as e:
                    return make_response_json(400, "发生错误 {repr(e)}")
                else:
                    if temp[1].user_id.id != current_user.id:
                        return make_response_json(400, "不可修改其他用户的联络地址")
                    has_default, num = False, 0
                    old_default = None
                    for k in j:
                        if isinstance(j[k], str) and len(j[k]) == 0:
                            return make_response_json(400, "不允许提交空参数")
                        if "campus_branch" in j:
                            if j["campus_branch"] not in User_Campus_state._value2member_map_:
                                return make_response_json(400, "校园填写错误")
                        if "user_id" in j and j["user_id"] != current_user.id:
                            if "default" in j and j["default"]:
                                if not has_default:
                                    has_default, num = True, 1
                                else:
                                    data[num]["default"] = False
                                    num += 1
                            old_default = None
                            try:
                                old_default = Contact.get(Contact.default == True,
                                Contact.user_id == current_user.id)
                            except Exception as e:
                                if not has_default:
                                    old_default.default = True
                                else:
                                    data[num]["default"] = False
                                    num = i
                            try:
                                old_default = Contact.get(Contact.default == True,
                                Contact.user_id == current_user.id)
                            except Exception as e:
                                if not has_default:
                                    old_default.default = True
                                else:
                                    data[-1]["default"] = True
                            else:
                                if has_default:
                                    old_default.default = False
                                    old_default.save()
                                else:
                                    if old_default in temp:
                                        return make_response_json(400, "至少需要一个默认地址")
                            for l in range(len(data)):
                                for m in data[l]:
                                    if m == "data":
                                        exec(f"if update_data[{l}][{m}] != data[{l}][{m}]: update_data[{l}][{m}] = data[{l}][{m}]")
                            update_data[1][1] = Contact(**update_data[1][1])
                            update_data[1][1].create()
                            except Exception as e:
                                return make_response_json(500, f"存储错误 {repr(e)}")
                            else:
                                if old_default is not None:
                                    old_default.default = True
                                    old_default.save()
                return make_response_json(500, f"存储错误 {repr(e)}")
            else:
                if old_default is not None:
                    old_default.default = False
                    old_default.save()
    return make_response_json(500, f"存储错误 {repr(e)}")

```

● 改进结果

1. post 方法

```

@api_blue.route("/address", methods=["POST"])
def address_post():
    res = check_user(current_user)
    if res[0] == -1:
        return res[1]
    ...
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    ...
    data = request.get_json()
    temp = [None for i in range(len(data))]
    has_default, num = False, 0
    old_default = None
    for i, j in enumerate(data):
        for k in j:
            if isinstance(j[k], str) and len(j[k]) == 0:
                return make_response_json(400, "不允许提交空参数")
        if "campus_branch" in j:
            if j["campus_branch"] not in User_Campus_state._value2member_map_:
                return make_response_json(400, "校区填写错误")
            j["user_id"] = current_user.id
        if "default" in j and j["default"]:
            if not has_default:
                has_default, num = True, i
            else:
                data[num]["default"] = False
                num = i
    try:
        old_default = Contact.get(Contact.default == True,
                                   Contact.user_id == current_user.id)
    except Exception as e:
        if not has_default:
            data[-1]["default"] = True
        else:
            if has_default:
                old_default.default = False
                old_default.save()
    for i, j in enumerate(data):
        try:
            temp[i] = Contact.create(**j)
        except Exception as e:
            for t in range(i):
                temp[t].delete_instance()
            if old_default is not None:
                old_default.default = True
                old_default.save()
            return make_response_json(500, f"存储时出现错误 {repr(e)}")
    return make_response_json(200, "完成")

```

2. put 方法

<pre> @api_blue.route("/address", methods=["PUT"]) def address_put(): res = check_user(current_user) if res[0] == -1: return res[1] ... if not current_user.is_authenticated: return make_response_json(401, "当前用户未登录") ... data = request.get_json() temp = [None for i in range(len(data))] for i, j in enumerate(data): # j["id"] = j["contact_id"] # j.pop("contact_id") for k in j: if isinstance(j[k], str) and len(j[k]) == 0: return make_response_json(400, "不允许提交空参数") if "campus_branch" in j: if j["campus_branch"] not in User_Campus_state._value2member_map_: return make_response_json(400, "校区填写错误") try: # j["id"] = j["contact_id"] # j.pop("contact_id") temp[i] = Contact.get(Contact.id == int(j["id"])) except Exception as e: return make_response_json(401, "不存在的联络地址") else: if temp[i].user_id.id != current_user.id: return make_response_json(401, "不可修改其他用户的联络地址") # update_data = list(range(len(data))) update_data = copy.deepcopy(temp) has_default, num = False, 0 for i, j in enumerate(data): if "default" in j and j["default"]: if not has_default: has_default, num = True, i else: data[num]["default"] = False num = i old_default = None </pre>	<pre> try: old_default = Contact.get(Contact.default == True, Contact.user_id == current_user.id) except Exception as e: if not has_default: temp[-1].default = True else: if has_default: if old_default not in temp: old_default.default = False old_default.save() else: if old_default in temp: return make_response_json(400, "至少要保留一个默认地址") for i in range(len(temp)): try: for j in data[i]: if j in update_data[i].__data__: exec(f'''if update_data[{i}].{j} != data[{i}][{j}]: update_data[{i}].{j} = data[{i}][{j}]''') new_data[j] = data[i][j] update_data[i] = Contact(**new_data) update_data[i].save() except Exception as e: for t in range(i): temp[i].save() if old_default is not None: old_default.default = True old_default.save() return make_response_json(500, f"存储错误 {repr(e)}") return make_response_json(200, "完成") </pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

3. delete 方法

```

@api_blue.route("/address", methods=["DELETE"])
def address_delete():
    res = check_user(current_user)
    if res[0] == -1:
        return res[1]
    ...
    if not current_user.is_authenticated:
        return make_response_json(401, "当前用户未登录")
    ...
    data = request.get_json()
    temp = [None for i in range(len(data))]
    data = list(set(map(lambda x: x["contact_id"], data)))
    for i, j in enumerate(data):
        try:
            temp[i] = Contact.get(Contact.id == int(j))
        except Exception as e:
            return make_response_json(401, "不存在的联络地址")
        else:
            if temp[i].user_id.id != current_user.id:
                return make_response_json(401, "不可删除其他用户的联络地址")
    delete_default = False
    for i, j in enumerate(temp):
        try:
            j.delete_instance()
        except Exception as e:
            for t in range(i):
                temp[t].save()
            return make_response_json(500, f"发生错误 {repr(e)}")
        else:
            if j.default:
                delete_default = True
    if delete_default:
        try:
            datas = Contact.select().where(
                Contact.user_id == current_user.id).order_by(
                Contact.id.desc()).execute()
        except Exception as e:
            print(repr(e))
        else:
            if len(datas) > 0:
                datas[0].default = True
                datas[0].save()
    return make_response_json(200, "完成")

```

● 测试结果

经过对创建地址、修改地址、删除地址的测试，系统均运行正确。

请双击表单内容进行修改				
收件人	电话	详细地址	所在校区	设为默认
mc	15088893266	7-237	嘉定校区	<input checked="" type="checkbox"/> <button>删除</button>
mc	15088893266	7-237	沪西校区	<input type="checkbox"/> <button>删除</button>

The image consists of two vertically stacked screenshots of a web-based user management system, likely for a university or organization.

Screenshot 1: Address Modification

A green header bar at the top says "修改保存成功". Below it is a circular logo with Chinese characters. The main content area has a light gray background. On the left, there's a sidebar with sections like "订单管理", "我的订单", "我的发布", etc. A central table lists addresses:

收件人	电话	详细地址	所在校区	设为默认
mc	15088893266	7-237	嘉定校区	<input checked="" type="checkbox"/> <button>删除</button>
mc	15088893266	7-237	四平路校区	<input type="checkbox"/> <button>删除</button>

A red box highlights the second row. At the bottom right is a green button labeled "添加新地址".

Screenshot 2: Address Deletion Confirmation

A green header bar at the top says "删除成功". Below it is a circular logo with Chinese characters. The main content area has a light gray background. The sidebar and table are identical to the first screenshot. A red box highlights the second row in the table. At the bottom right is a green button labeled "添加新地址".

改进 5.13 从用户管理相关中提取方法

● 改进原因

源代码中用户管理部分的一系列方法最前面都有对用户登录与否、用户权限、用户是否被封禁的状态的判断，各个方法其判断各不相同，但基本一致。

而源代码中并没有提取统一判断函数，导致每个方法都存在具体的判断，这样不仅使得重复代码增多，其判断方式与结果也不统一。

故使用提取函数的重构方法，提取这一部分对用户的判断。

```

#管理员封号
@api_blue.route('/change_user_state', methods=['PUT'])
def change_user_state():
    if not current_user.is_authenticated:
        return make_response_json(401, "该用户未通过验证")

    if current_user.state < User_state.Admin.value:
        return make_response_json(401, "权限不足")

#在APIFOX测试运行时current_user未认证，需要先在apifox上登录
req = request.get_json()
try:
    user_id = int(req['user_id'])
    user_state = int(req['user_state'])
except:
    return make_response_json(400, "请求格式不对")

@api_blue.route('/get_user_info', methods=['GET'])
def get_user_info():
    if not current_user.is_authenticated:
        return make_response_json(401, "该用户未通过验证")
    data = dict(request.args)
    if "user_id" in data:
        try:
            user_id = int(data['user_id'])
        except:
            return make_response_json(400, "请求格式错误")
    else:
        user_id = current_user.id
    #user_id = int(request.get_json()["user_id"])
    try:
        tep = User.get(User.id == user_id)
    
```

```

@api_blue.route('/get_all_user', methods=['GET'])
def get_all_user():
    if not current_user.is_authenticated:
        return make_response_json(401, "该用户未通过验证")

    if current_user.state < User_state.Admin.value:
        return make_response_json(401, "权限不足")

users = User.select().where(User.state != User_state.Admin.value)
data_list = []
for i in users:
    user_dic = GetUserDict(i, True)
    data_list.append(user_dic)

if len(data_list) == 0:
    return make_response_json(404, "无用户信息")
return make_response_json(200, "所有用户信息获取成功", data_list)

@api_blue.route('/change_user_info', methods=["PUT"])
def change_user_info():
    if not current_user.is_authenticated:
        return make_response_json(401, "该用户未通过验证")
    req = request.get_json()
    #print(req)
    try:
        tep = User.get(User.id == current_user.id)
    except:
        return make_response_json(404, "未找到用户")
    else:
        try:
            if 'username' in req:
                if len(req["username"])>User.username.max_length:
                    return make_response_json(400,f"用户名长度过长,应{User.username.max_length}字节")
                tep.username = req['username']
        
```

● 改进结果

提取结果如下：

```

# 对用户的检查
def check_user(user, admin_power_check = False, ban_check = False):
    if not user.is_authenticated:
        return -1, make_response_json(401, "该用户未通过验证或未登录")
    if admin_power_check and user.state != User_state.Admin.value:
        return -1, make_response_json(401, "权限不足")
    if ban_check and user.state == User_state.Under_ban.value:
        return -1, make_response_json(401, "当前用户已被封禁")

    return 0, 0

```

通过不同的标志位来决定是否进行某个用户特征的判断，以此来统一对用户的判断。

调用时如下：

```

@api_blue.route('/get_all_user', methods=['GET'])
def get_all_user():
    res = check_user(current_user, True)
    if res[0] == -1:
        return res[1]

```

● 测试结果

经过测试，所有相关功能均能正确执行。

四、 重构后软件系统测试报告

*见附件中《同济跳蚤市场重构报告》

五、 参考文献

- [1] TracingOrigins. 《开发规范》代码命名规范[EB/OL].(2022-06-18)[2022-08-19]<https://blog.csdn.net/TracingOrigins/article/details/124388901>
- [2] Download.幼稚的数据类 (Data Class)[EB/OL].(2021-10-07)[2022-08-19]<https://www.jianshu.com/p/cb1eaea7feec>
- [3] yasoob. 《Intermediate Python》 装饰器 - Python 进阶 [EB/OL].(2020-09-21) [2022-08-19]<https://eastlakeside.gitbook.io/interpy-zh/decorators/>
- [4] 码农架构.MVC 架构解析 [EB/OL].(2020-11-12) [2022-08-19]<https://www.jianshu.com/p/499b5e62c08c>
- [5] Candyabc.[python] [flask] 使用技巧(一) 使用 before_request 来做权限和用户检查 [EB/OL].(2018-01-08)[2022-08-19] <https://www.jianshu.com/p/499b5e62c08c>
- [6] ldc_1257309054.python 框架 Django 中的 MTV 架构 [EB/OL].(2021-07-17)[2022-08-19]<https://www.jianshu.com/p/499b5e62c08c>
- [7] 捡田螺的小男孩.25 种代码坏味道总结+优化示例 [EB/OL].(2021-05-16)[2022-08-19]<https://juejin.cn/post/6962812178537644063>