Database2Sharp 代码生成工具

Python 开发介绍

V1.0

伍华聪

2025 年 01 月

目录

1 引言

1.1 前言

Database2Sharp 是一款代码生成工具和数据库文档生成工具，该工具从 2005 年开始至今，一直伴随着我们的客户和粉丝们经历着过各种各样的项目开发，在实际开发中能带来效率的提高及编程的快乐。

Database2Sharp 是一款最初主要用于 C#代码生成以及数据库文档生成的工具，软件支持 Oracle、SqlServer、MySql、PostgreSQL、Sqlite 以及国产达梦等数据库的代码生成。随着我们框架的扩展，从最初的.NET 体系也扩展到了纯前端的 Vue 项目，以及跨平台的 **Python 开发框架**等。

代码生成工具根据不同的数据库读取相关的表、视图、存储过程、字段等基本元数据和关系到模型中，可以生成各种架构后端处理的代码和各个前端的界面代码。如可以生成 Winform 界面代码、Web 界面代码（包括 EasyUI 和 BootstrapWeb 界面）、Entity Framework 实体框架代码、ABP/ABP VNext 框架代码生成、SqlSugar 框架代码生成、**Python 开发框架代码**，以及导出数据库文档、浏览数据库架构、查询数据、生成 Sql 脚本等，还整合自定义模板和数据库信息的引擎，方便编写自定义模板调试和开发。

由于我们设计的开发框架，一般都是和支持多种数据库使用的，因此生成的框架代码也就支持多种数据库，一种适应性非常强、弹性很好的应用框架。

利用 Database2Sharp 进行快速开发，一个简单点击几次鼠标就能完成一周代码量的代码生成工具，效率惊人、友好体贴，真正的开发好伴侣。

针对各种开发场景，如 Winform 开发框架、混合式开发框架、基于 ABP/ABP VNext 开发框架、SqlSugar 开发框架、**Python 开发框架**等,我们也针对项目实际的情况进行相关代的生成工作，并通过引入基类封装的方式，尽可能的提高开发效率，减少编写代码。

当然，开发的过程是一个繁复、精细的过程，因此 Database2Sharp 也吸收了来自我们自己的实际需求，以及很多同仁朋友的宝贵意见，一直在改进，一直努力做到更好，以求达到一个更加完美、更加易用的境界。

如果您有好的意见以及建议，如能不吝赐教，则感激不尽，可以通过邮件

wuhuacong@163.com 或者 QQ 6966254 与我们联系。

1.2 背景

我们一系列的开发框架目的是为了让用户快速开发特定的项目，从快速入门、工具辅助、重用模块、技术协助等多个维度为您及您的团队保驾护航，从而能够节省用户学习探索和开发常规模块的时间，并可以基于已有的基础模块快速开发项目。

随着大环境的跨平台需求越来越多，对与开发环境和实际运行环境都有跨平台的需求，Python 开发和部署上都是跨平台的，因此很好的满足这个大环境的需求。

由于我们多年深耕.NET 开发框架领域，形成有自己独到开发框架思路和丰富的经验，我们把它们拓展到 Python 开发的领域，形成我们的《Python 开发框架》。该开发框架利用 Python 开发的最新、最广泛的技术，为客户提供最直接、高效的开发帮助。

《Python 开发框架》是严格的前后端分离模式，后端是基于 Python + FastApi + SqlAlchemy + Pydantic 的 Web API 服务，前端可以是基于 Web API 接入的任何前端，目前纯 Python 的前端为基于 Python + WxPython 的开发前端，后面会继续扩展其他前端，如 Python+PyQt 等；基于 BS 的前端为 Vue3+ElementPlus+TypeScript 的前端。

1.3 编写目的

本篇内容主要介绍《Python 开发框架》中前后端中利用代码生成工具来进行开发的相关内容，包括后端 FastApi 项目的开发，前端 WxPython 的开发等内容。

2 FastApi 项目后端开发介绍

2.1 关于代码生成工具 Database2Sharp

代码生成工具可以到地址下载：https://www.iqidi.com/database2sharp.htm。

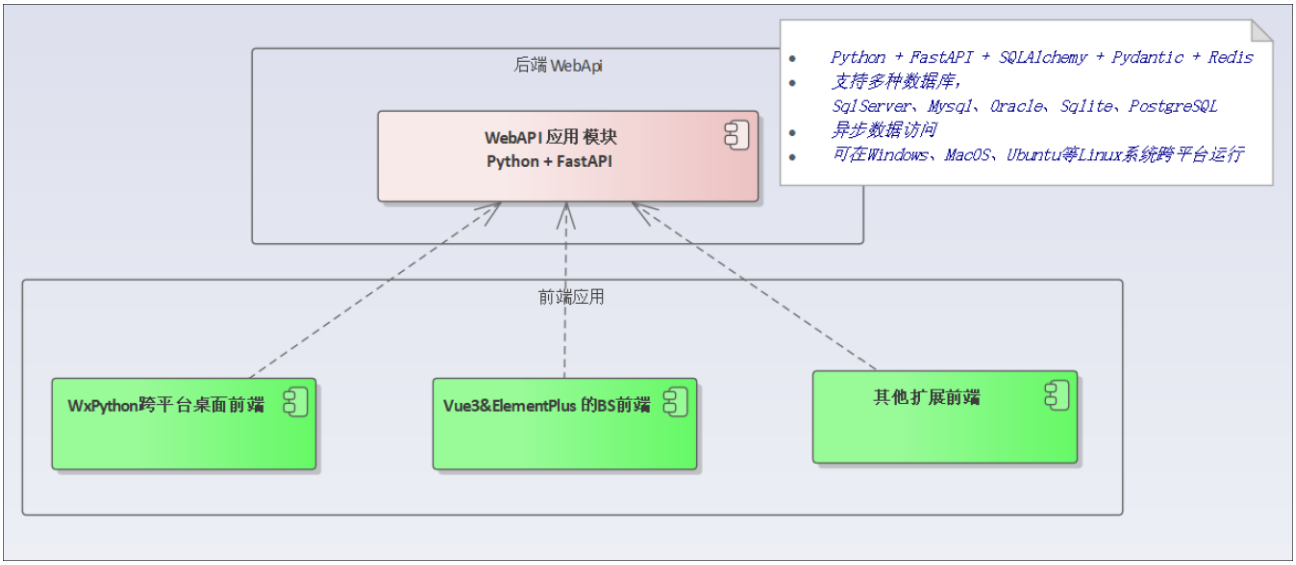
使用代码生成工具来快速开发项目代码，有很多好处。

- **减少重复工作**：自动生成常用代码（如数据模型、CRUD 操作、API 接口等），减少手动编写的时间。
- **专注于核心逻辑**：开发者可以将时间集中在业务逻辑和复杂问题的解决上，而非基础代码的编写。
- **模块化和规范化**：生成代码一般遵循既定的架构和风格，便于后续维护和扩展。
- **初学者友好**：新手开发者可以通过代码生成工具快速上手，了解项目结构和基础代码。

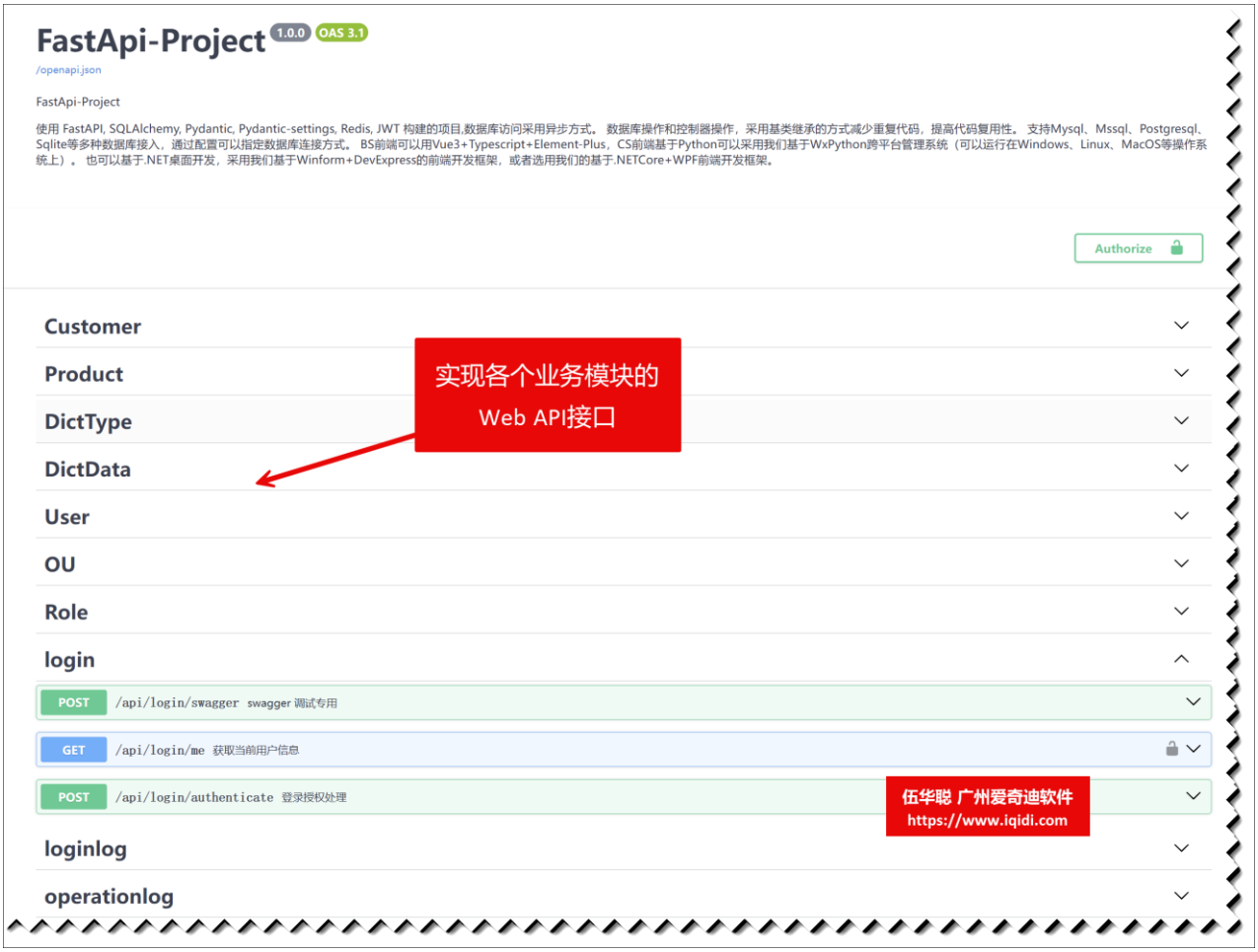
代码生成工具在提升开发效率、降低出错率、标准化代码方面具有显著优势，尤其在重复性工作较多或团队合作时尤为适用。

2.2 理解后端项目的结构

Python + FastAPI 项目是一个 Web API 的项目，为各个前端提供接口的后端项目，也就是多个终端共用一个 WebAPI，统一数据中心的模式。

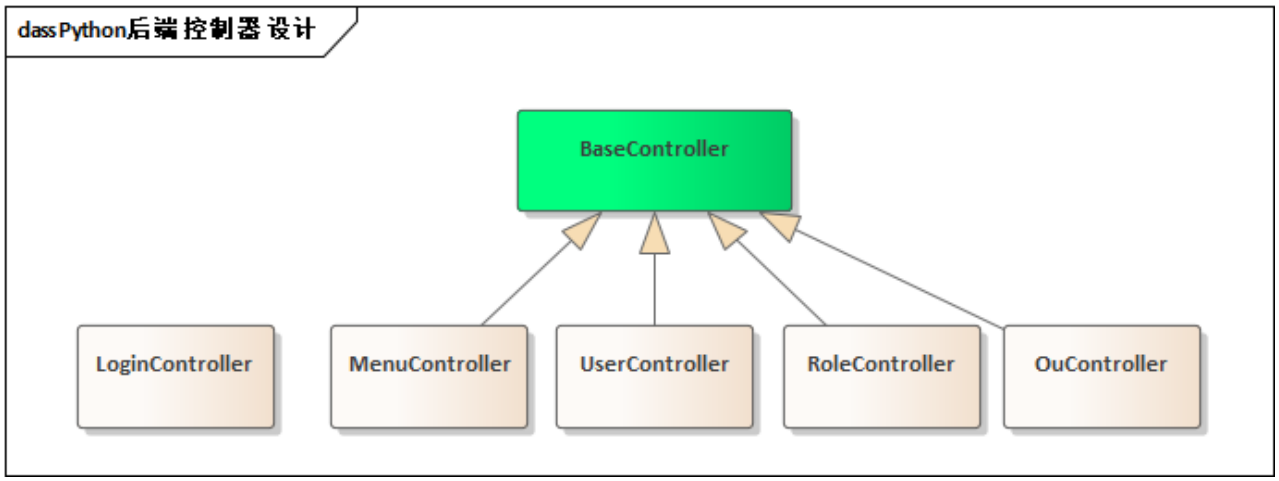


FastAPI 项目运行后，其界面自动整合 Swagger 的文档界面，如下所示。

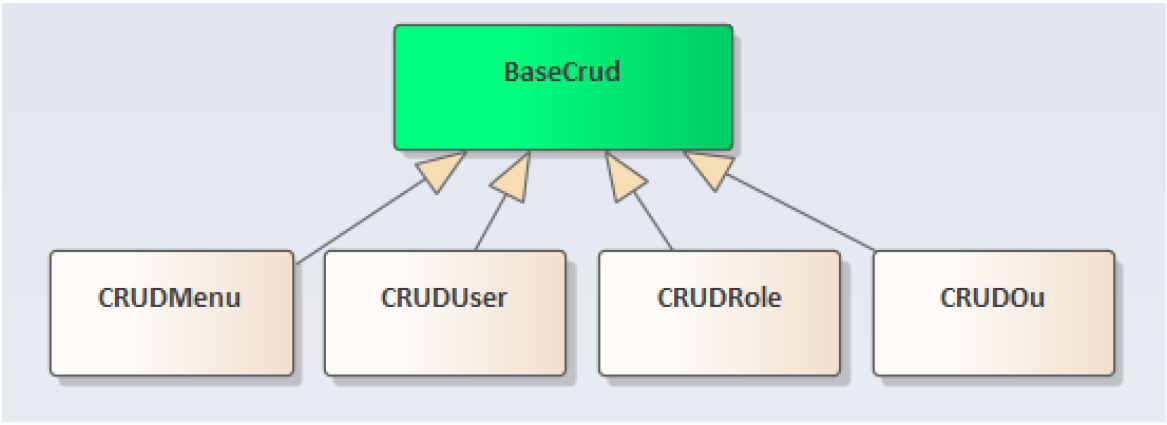


在 FastApi 的后端项目中，我们对各个层的基类做了一些封装，以便处理一些通用的逻辑，减少相关的代码。

后端的 Web API 控制器层，我们采用下面的继承方式来实现一些逻辑的剥离和抽象。



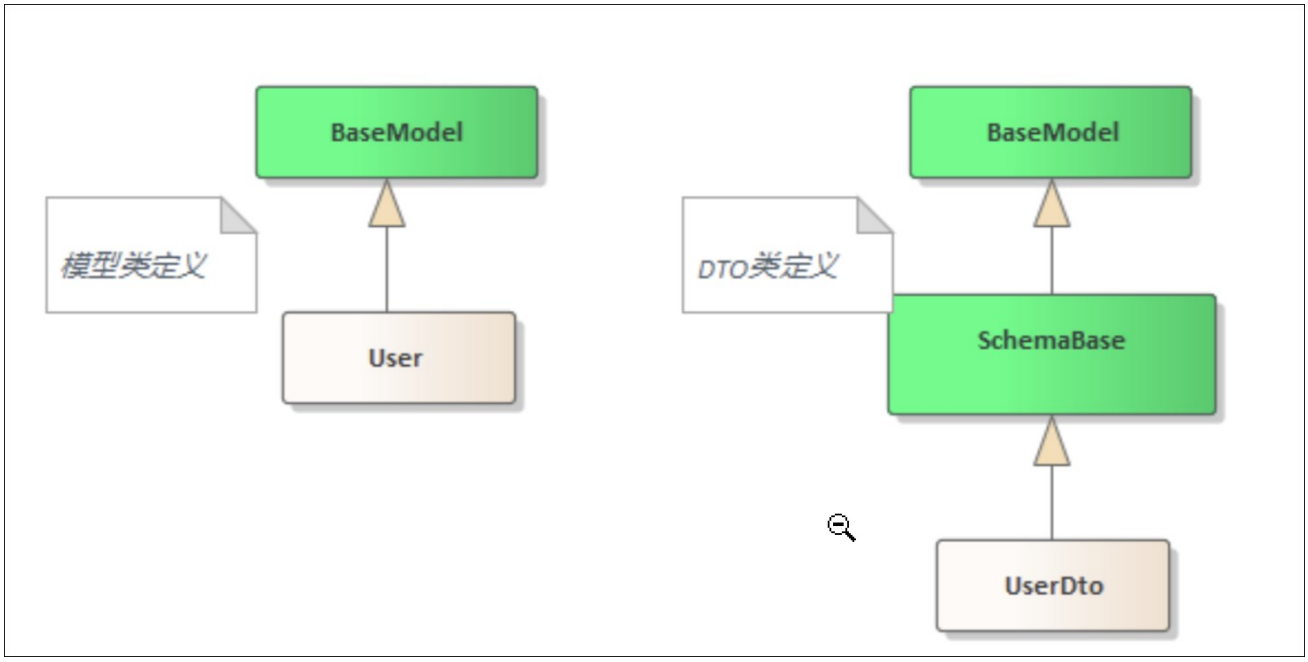
后端针对数据库访问的常规处理，我们也做了抽象的封装。



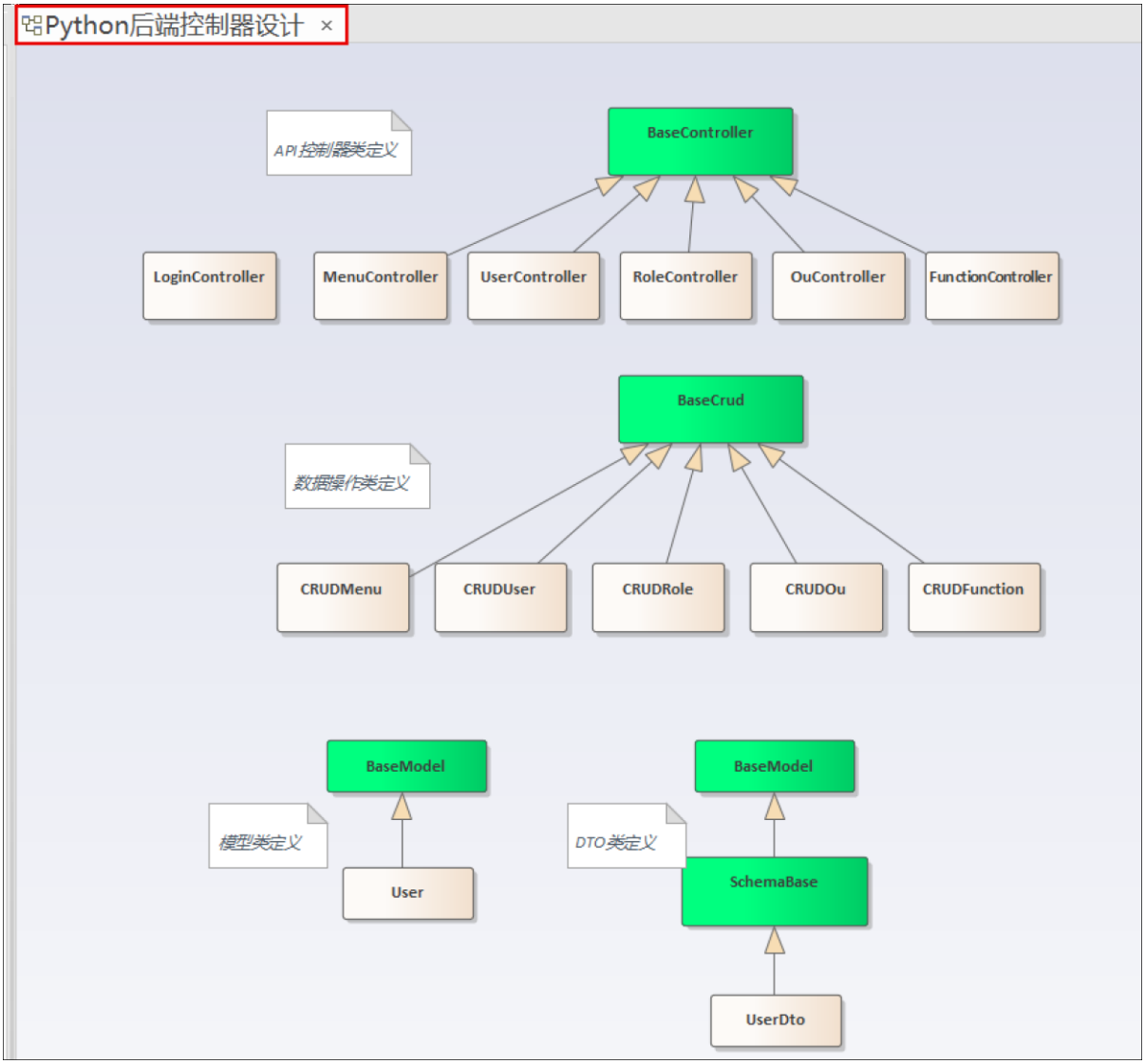
对于不同的业务表，我们继承 BaseCrud，并传递不同对象的参数，就可以具有非常强大、丰富的函数处理功能了。

其中我们在 BaseCrud 类中实现了常规查询条件的处理逻辑，以及排序规则，可以默认排序，或者根据查询对象的排序条件进行排序处理。以及相关的 CRUD 操作。

对于 DTO 对象，作为 UI 界面上的交换对象，我们也做了基类的定义，默认 BaseModel 是 pydantic 对象，pydantic 一般作为后端数据接入的处理类库，可以对数据格式进行校验和映射等处理。



我们可以在 SchemaBase 中进行了一些定制化的处理，这样可以让他满足我们实际的需要，最终各个分层的基类设计关系如下图所示。

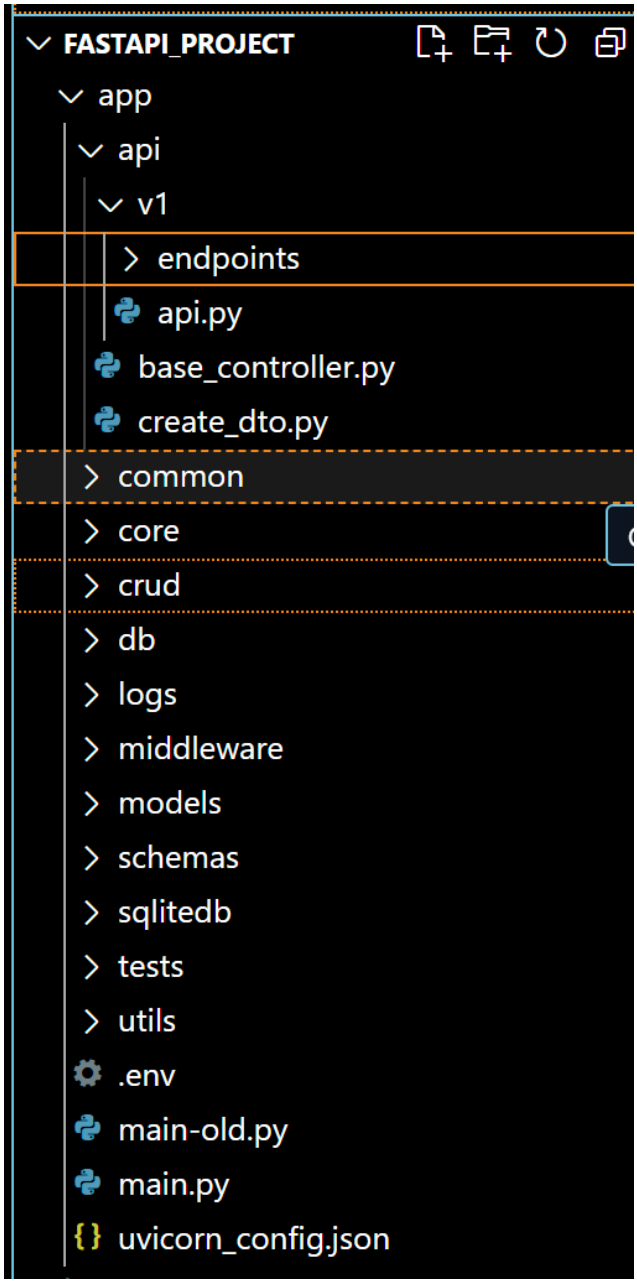


框架的分层沿用一些通用的做法，如下所示：

分层介绍	Java、C#开发	Python 开发框架
视图、控制器	Controller	api
数据传输	DTO	schema
业务逻辑	Service + Interface	service
数据访问	DAO / Mapper	crud
模型	Model / Entity	model

按照分层逻辑的划分，我们对于每个分层中的对象，我们都应该尽量减少重复编码，因此使用 Python 的继承关系来抽象一些通用的属性或者接口及实现等，以便实现更加高效的开发，减少冗余代码。

最终项目的结构如下所示。



2.3 FastApi 项目后端开发

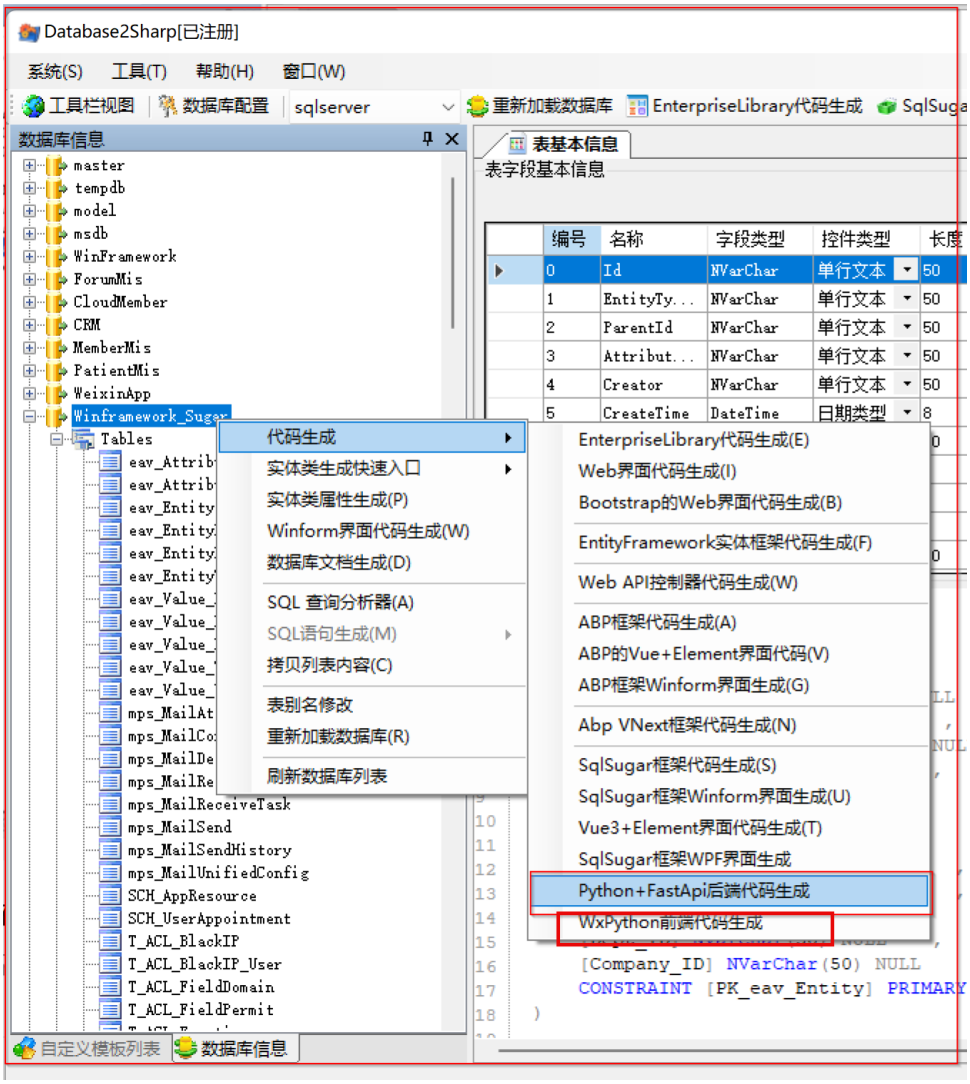
为方便快速开发项目代码，我们在代码生成工具中整合了基于 Python 跨平台方案项目的代码快速生成，包括基于 FastApi 的后端 Web API 项目，以及前端的 WxPython 前端界面

项目。

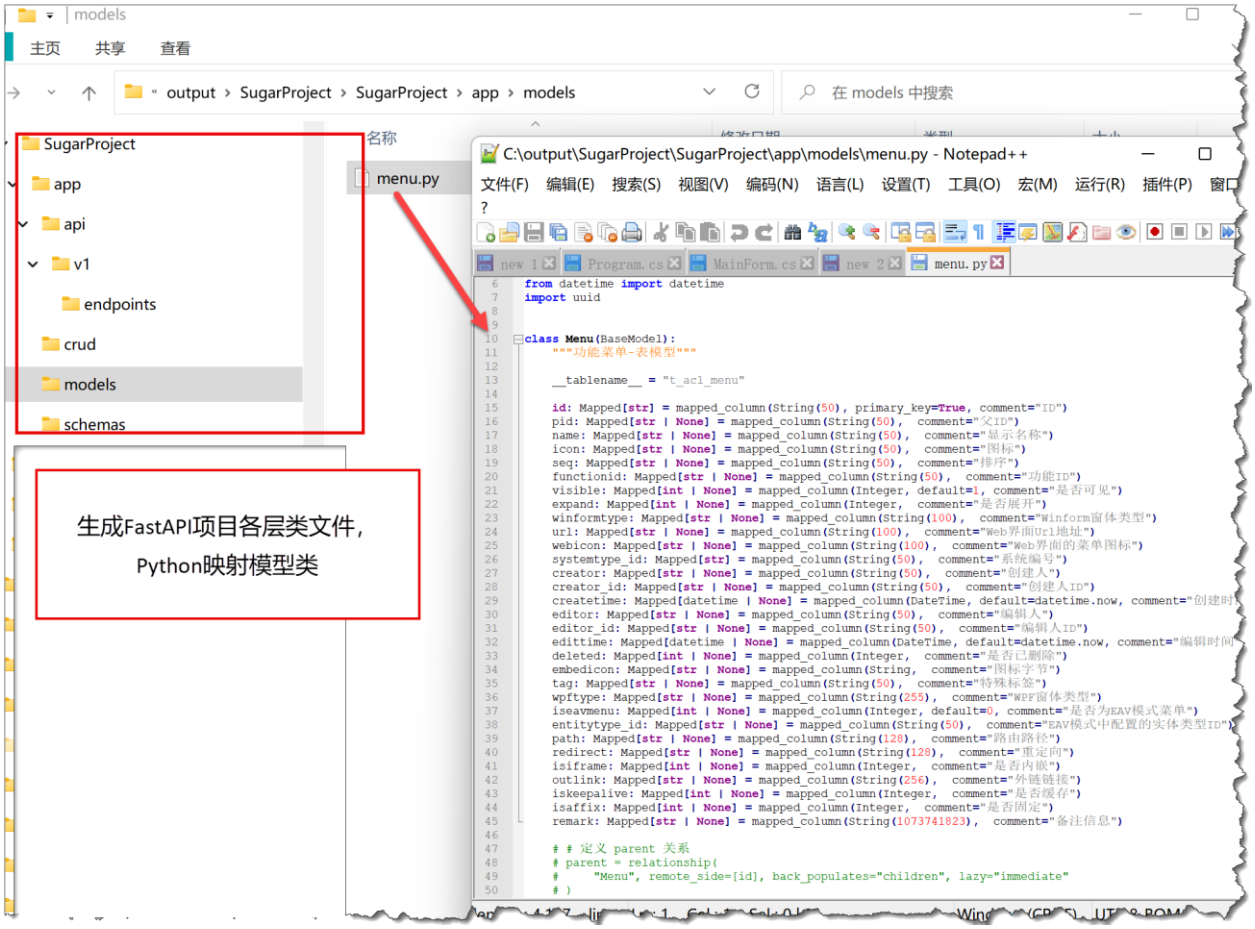
前面介绍了 FastApi 后端框架的相关基类抽象处理操作，虽然大多数规则已经进行了基类的抽象的处理，但是对于一个新增的业务表，我们还是需要在不同的分层目录中添加对应的子类，如控制器、CRUD 数据访问类、Model 模型类，DTO 对象类等，特别是对于模型类和数据库表的一一对应代码，手工编写肯定比较枯燥，因此这些问题，我们使用代码生成工具一次性解决它。

代码生成工具可以到地址下载：https://www.iqidi.com/database2sharp.htm。

在展开数据库信息后，可以再代码生成工具的主工具栏或者右键菜单上执行后端代码的生成或 WxPython 前端代码生成的操作，如下界面所示。



我们在代码生成工具中加入 Python 的后端代码的生成，选中相关的表后，一键可以生成各层的类文件，其中包括最为繁琐的 Model 映射类信息。



生成的代码各个分层的目录结构和实际的开发框架项目一致，直接复制到项目目录上即可，在 VSCode 上可以马上看到更新的内容。

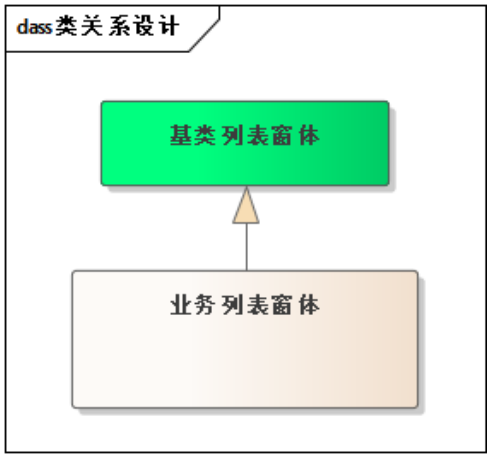
对于已有的项目文件，我们可以使用类似 Beyond Compare 的对比工具，进行一些代码替换即可。

3 WxPython 前端开发介绍

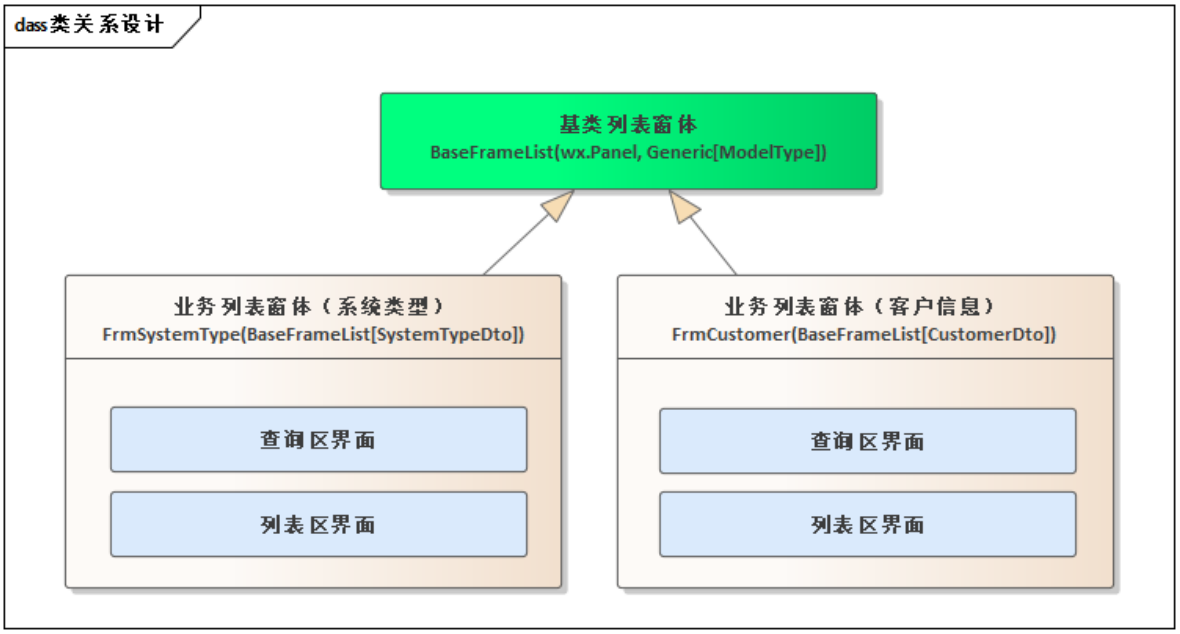
3.1 理解前端项目结构

1）列表界面和继承关系

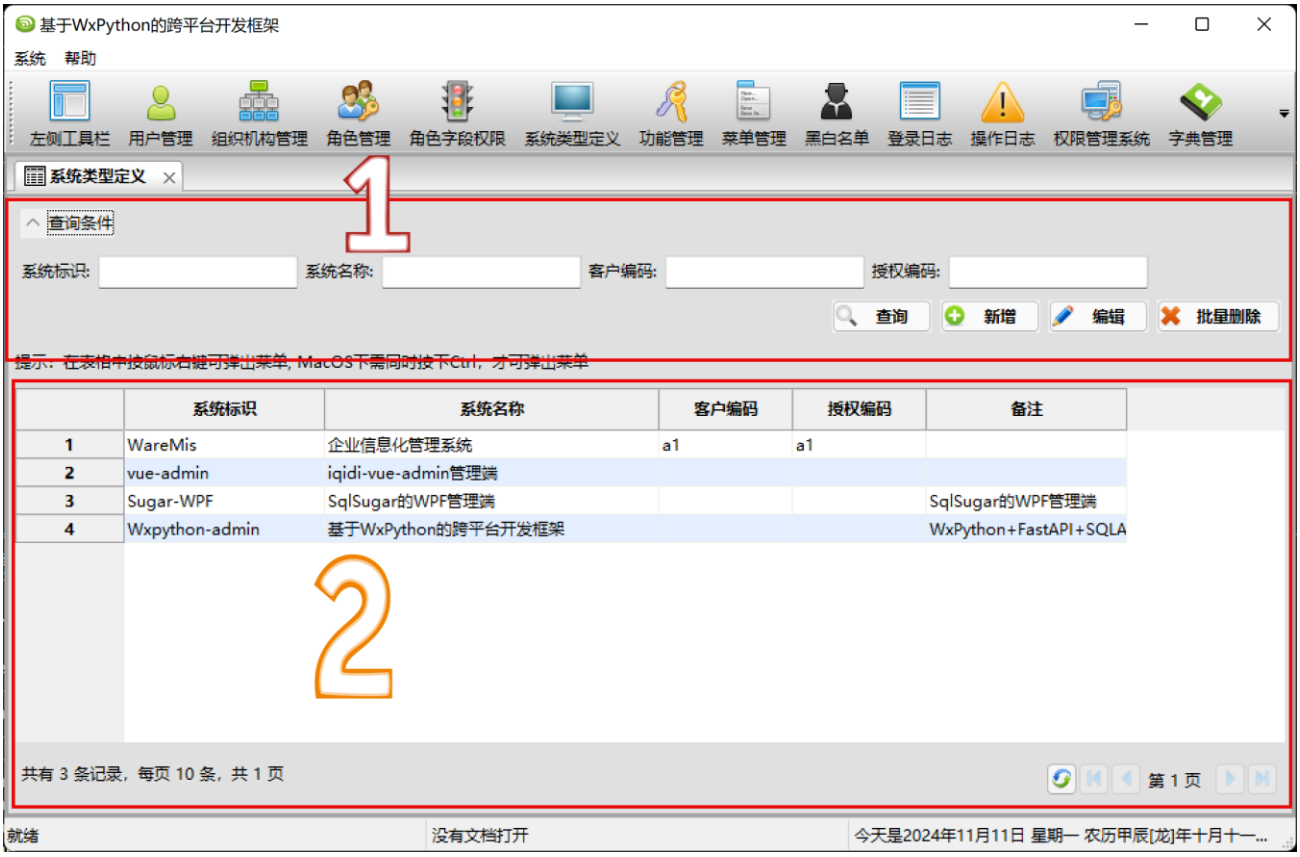
列表界面继承基类，从而可以大幅度的利用相应的规则和实现。



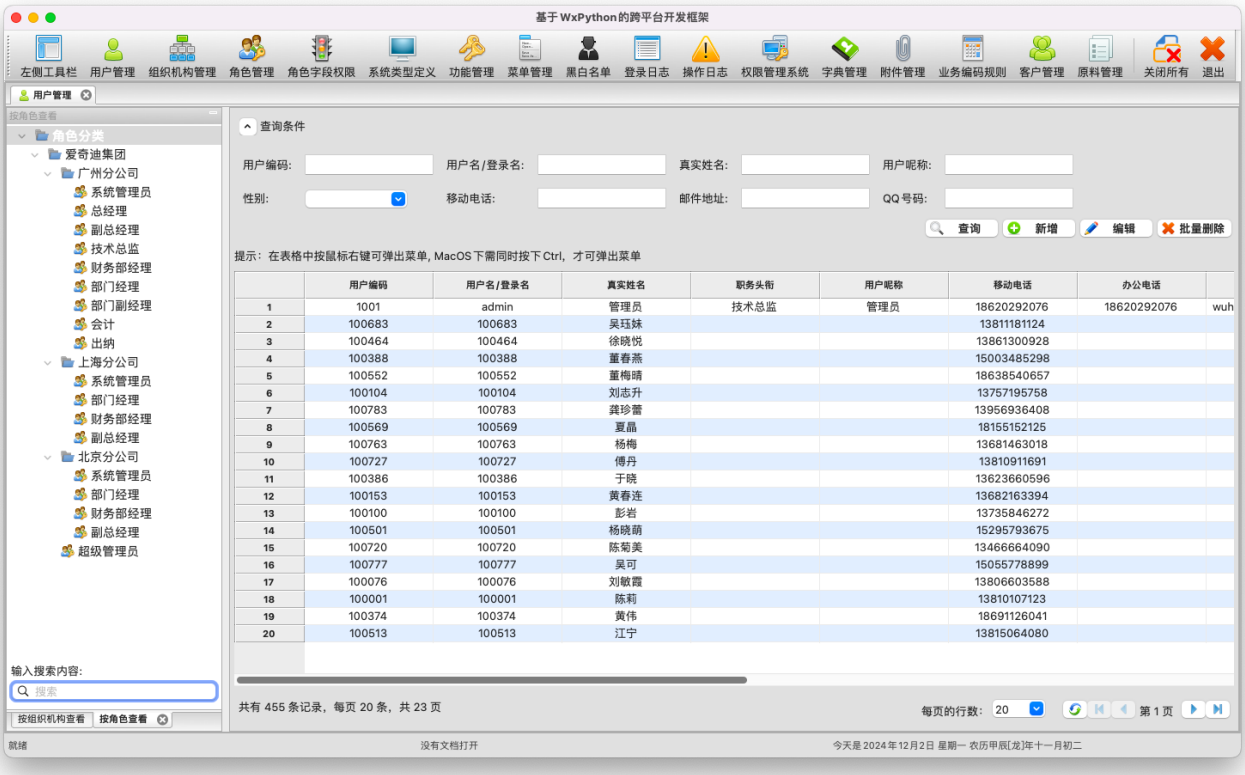
如对于两个例子窗体：系统类型定义，客户信息，其中传如对应的 DTO 信息和参数即可。



因为常规的列表界面一般分为查询区、列表界面展示区和分页信息区，我们把它分为两个主要的部分，如下界面所示。



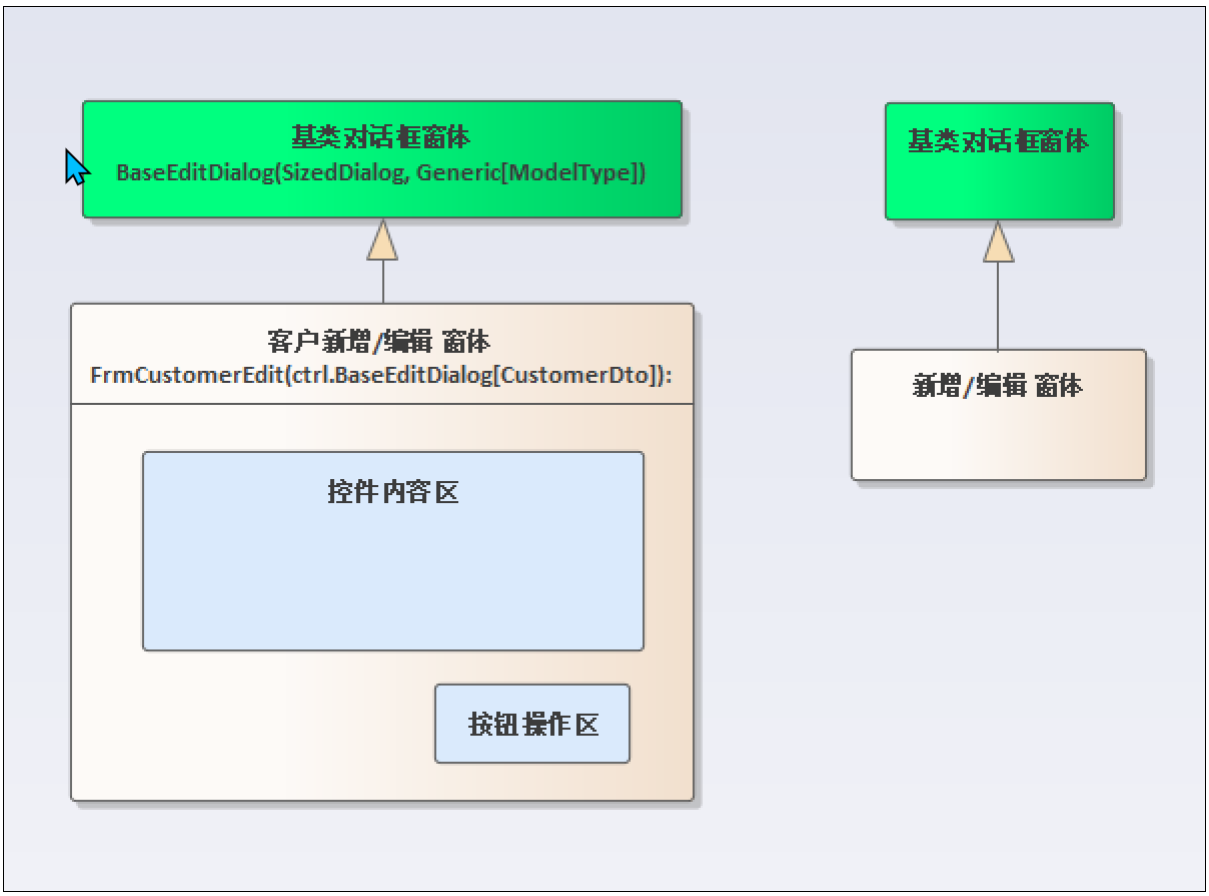
当然如果有树形列表的，也整合在基类窗体中实现控制逻辑，具体实现放在子类处理即可。



同时，在树列表或者表格数据控件支持右键弹出菜单处理，包括常规的新增、编辑、删除、

复制、刷新等常规功能，如果需要更多业务模块的功能，整合在右键菜单中，在窗体子类中重写某些只定义函数即可实现。

2）编辑/新增界面继承关系



继承基类编辑对话框（通常用于创建模态对话框或自定义窗口的基类）有以下优点：

- **共享通用功能**：将所有对话框的共同行为（如按钮布局、事件处理、数据校验逻辑等）

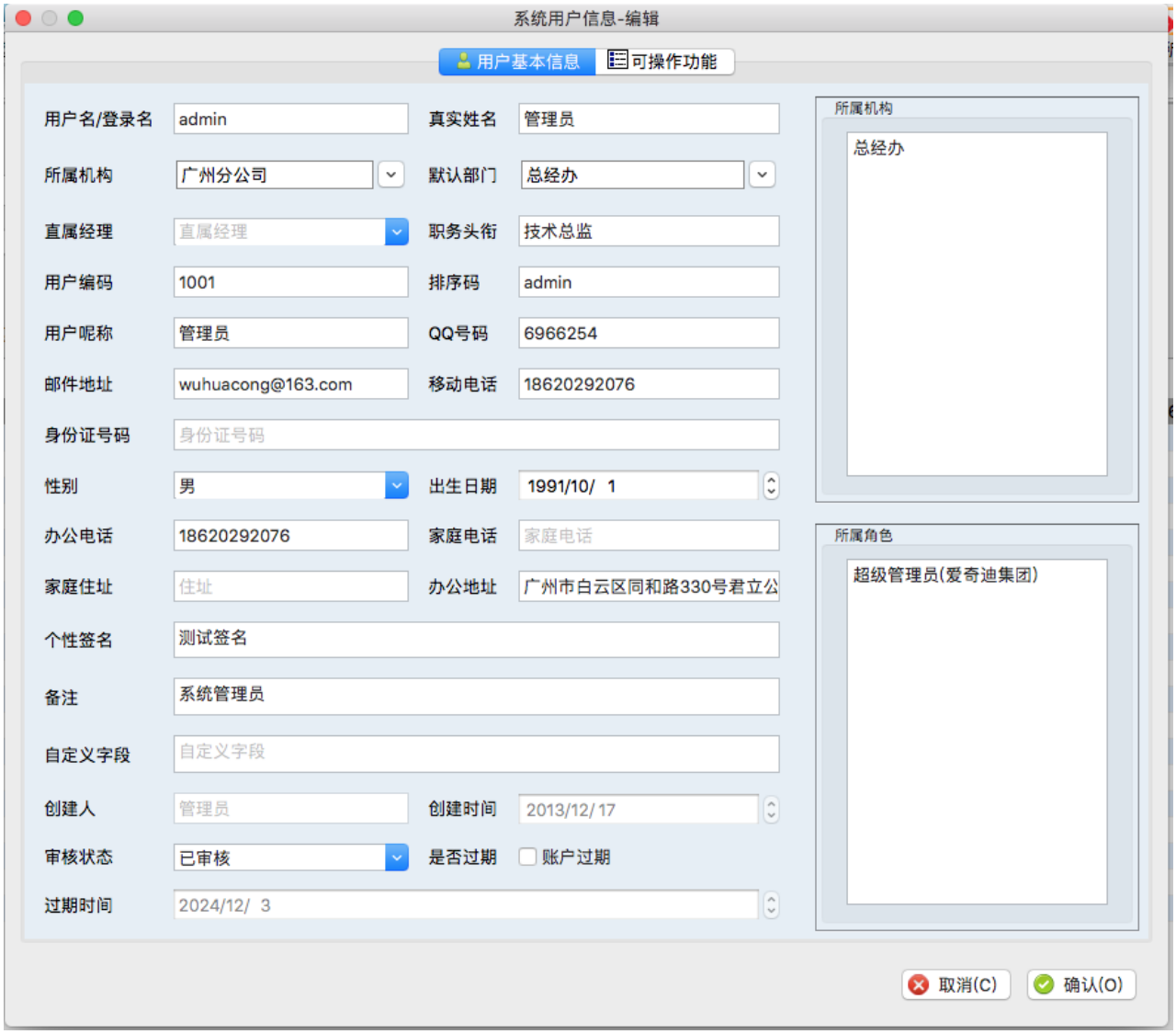
封装在基类中，子类可以直接继承使用，无需重复实现。

- **减少冗余代码**：对话框的通用结构只需在基类中实现一次，后续的功能扩展只需通过继承来实现，减少代码重复。
- **统一界面风格**：基类可以预定义窗口的样式和布局，确保项目中所有对话框的界面一致。
- **快速定制功能**：子类仅需实现或覆盖特定方法，即可快速实现自定义对话框的功能。

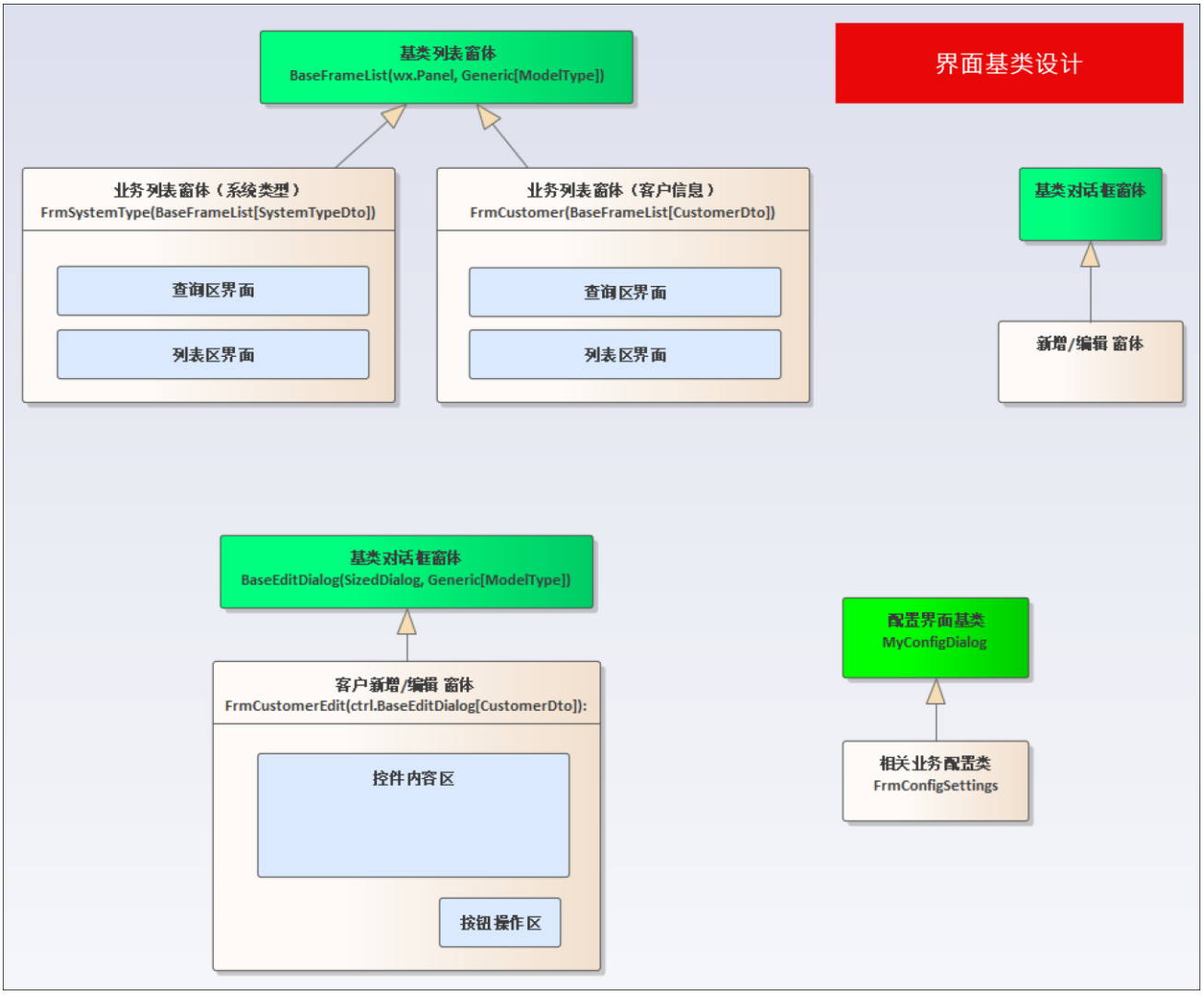
继承基类编辑对话框能够提高代码复用性、开发效率和维护性，特别适合在复杂系统中管理多个相似对话框。通过合理设计基类，开发者可以显著减少重复代码，实现更灵活的功能扩展。

常规的对话框中，业务表编码规则的新增、编辑界面如下所示。

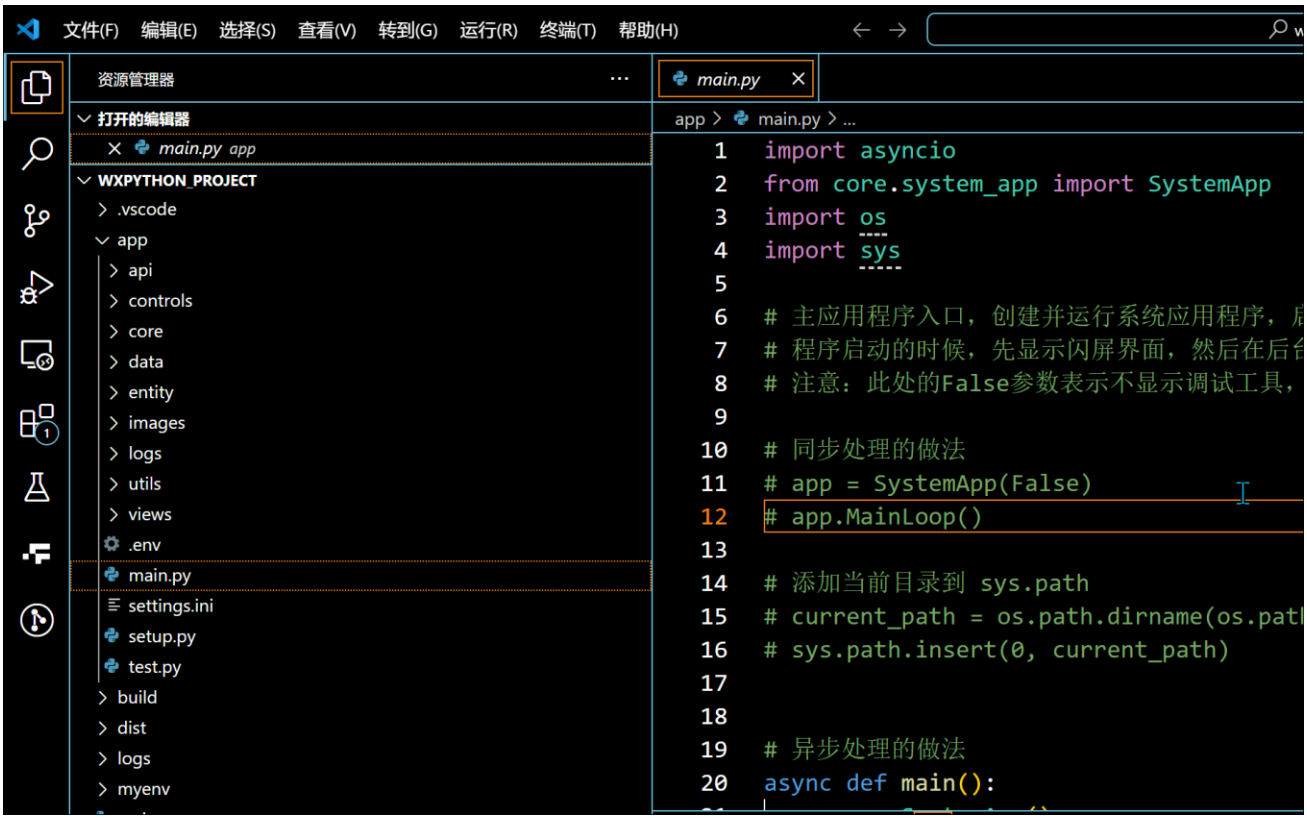
当然，我们也可以增加更多的定制功能，稍作调整可以增加多页的功能。



最终我们可以看到列表或者编辑界面的基类设计关系如下所示。


基类列表窗体
BaseFrameList(wx.Panel, Generic[ModelType])
界面基类设计
业务列表窗体（系统类型）
FrmSystemType(BaseFrameList[SystemTypeDto])
查询区界面
列表区界面
业务列表窗体（客户信息）
FrmCustomer(BaseFrameList[CustomerDto])
查询区界面
列表区界面
基类对话框窗体
新增/编辑 窗体
基类对话框窗体
BaseEditDialog(SizedDialog, Generic[ModelType])
客户新增/编辑 窗体
FrmCustomerEdit(ctrl.BaseEditDialog[CustomerDto]):
控件内容区
按钮操作区
配置界面基类
MyConfigDialog
相关业务配置类
FrmConfigSettings

最终项目结构如下所示：



主要的目录结构介绍如下所示。

app/: 项目的主目录，包含所有应用相关代码。

main.py: 项目的入口文件。

.env: 环境变量文件，用于存储敏感信息，如数据库连接字符串。

README.md: 项目说明文件。

requirements.txt: 项目依赖列表。

api/: 用于处理客户端 API 请求, 用于管理用户、角色、机构、权限等。

user.py : 用户 API，用于管理用户信息。

role.py : 角色 API，用于管理角色信息。

ou.py : 机构 API，用于管理机构信息。

menu.py : 菜单 API，用于管理菜单信息。

...

controls/: 自定义控件库， 用于扩展系统功能，包括文本框、数值框、日期选择、下拉框、树形控件、表格控件等。

my_textctrl.py : 文本框控件。

my_comobox.py : 下拉框控件。

my_dialog.py： 对话框控件。

core/: 核心功能，如配置、安全等。

config.py: 项目的配置信息，包括数据库连接信息、日志配置等。

system_app.py： 系统应用，用于管理系统登录，以及获取用户相关信息，如用户、角色、机构、权限等。

system_splash_screen.py： 系统启动画面，用于显示启动信息。

system_taskbar_icon.py： 系统任务栏图标，用于显示系统状态。

core_images.py： 系统图片资源，用于显示系统图标等。

...

entity/: 数据模型，用于请求和响应的数据模型。

user.py： 用户实体。

role.py： 角色实体。

ou.py： 机构实体。

...

utils/: 工具函数和公用模块。

message_util.py： 用于处理系统消息。

filedialog_util.py： 用于处理文件选择对话框。

...

views/: 视图层，用于展现各个模块的界面，包括列表界面和编辑界面等。

frm_user.py： 用户视图。

frm_user_edit.py： 用户编辑视图。

...

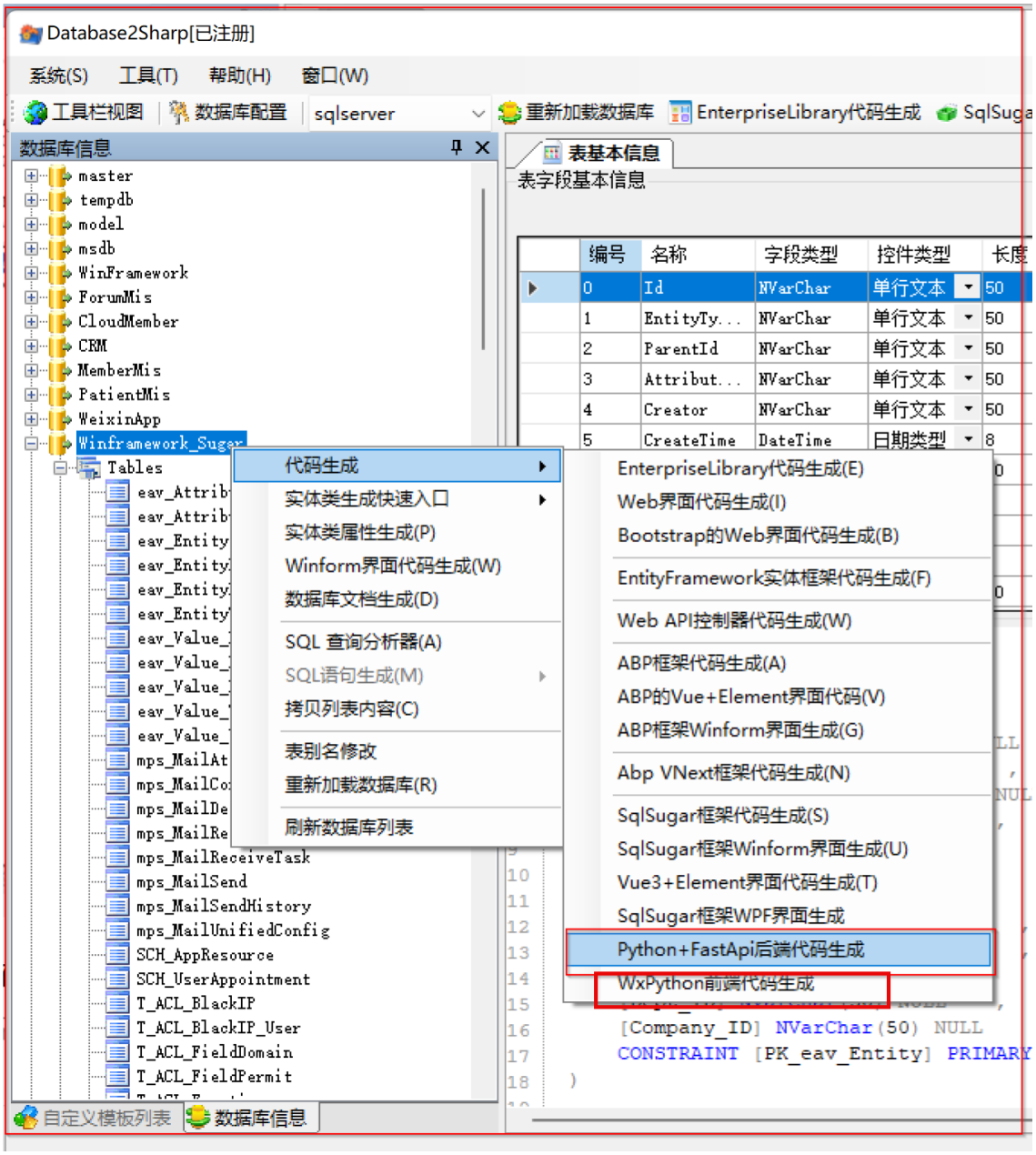
images/: 图片资源。

myenv/: 虚拟环境目录，用于隔离项目依赖库。

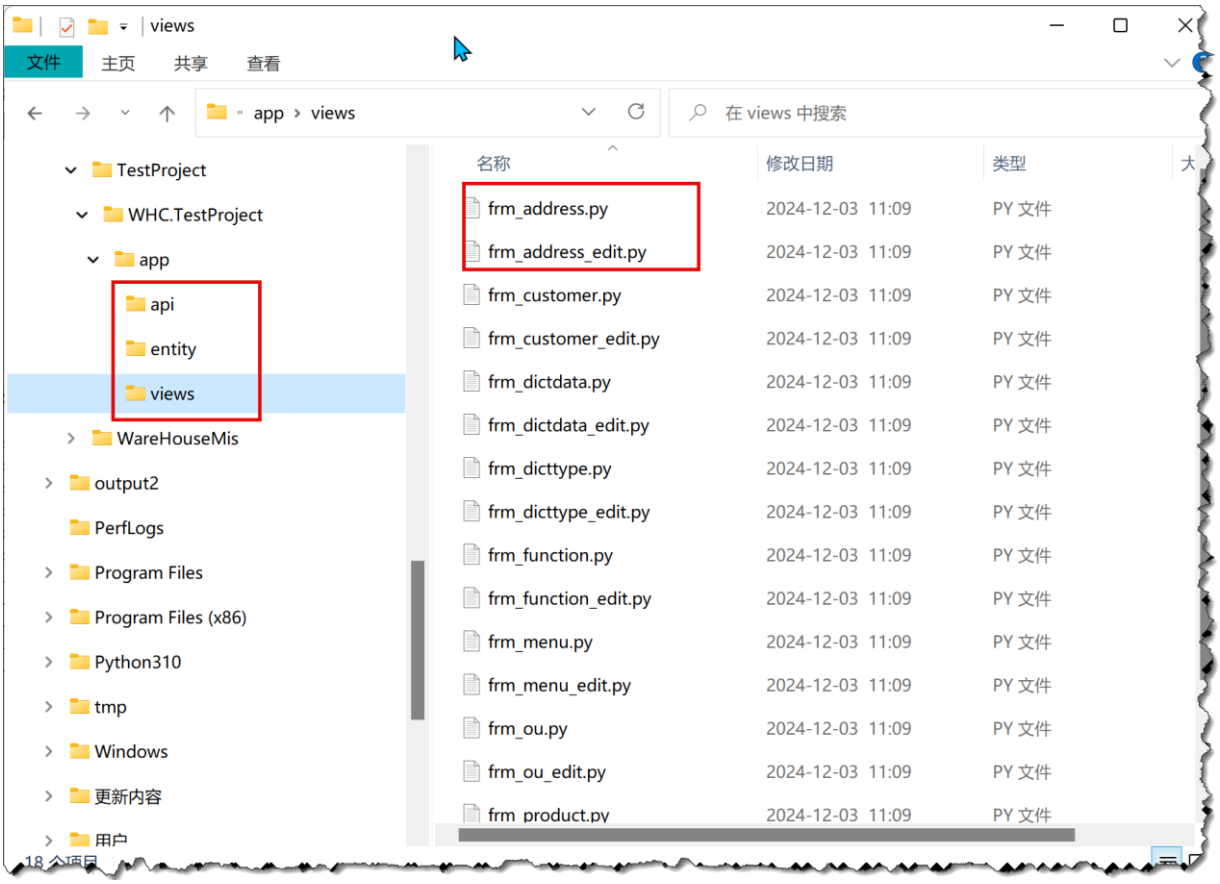
logs/:生成日志文件。

3.2 WxPython 前端开发介绍

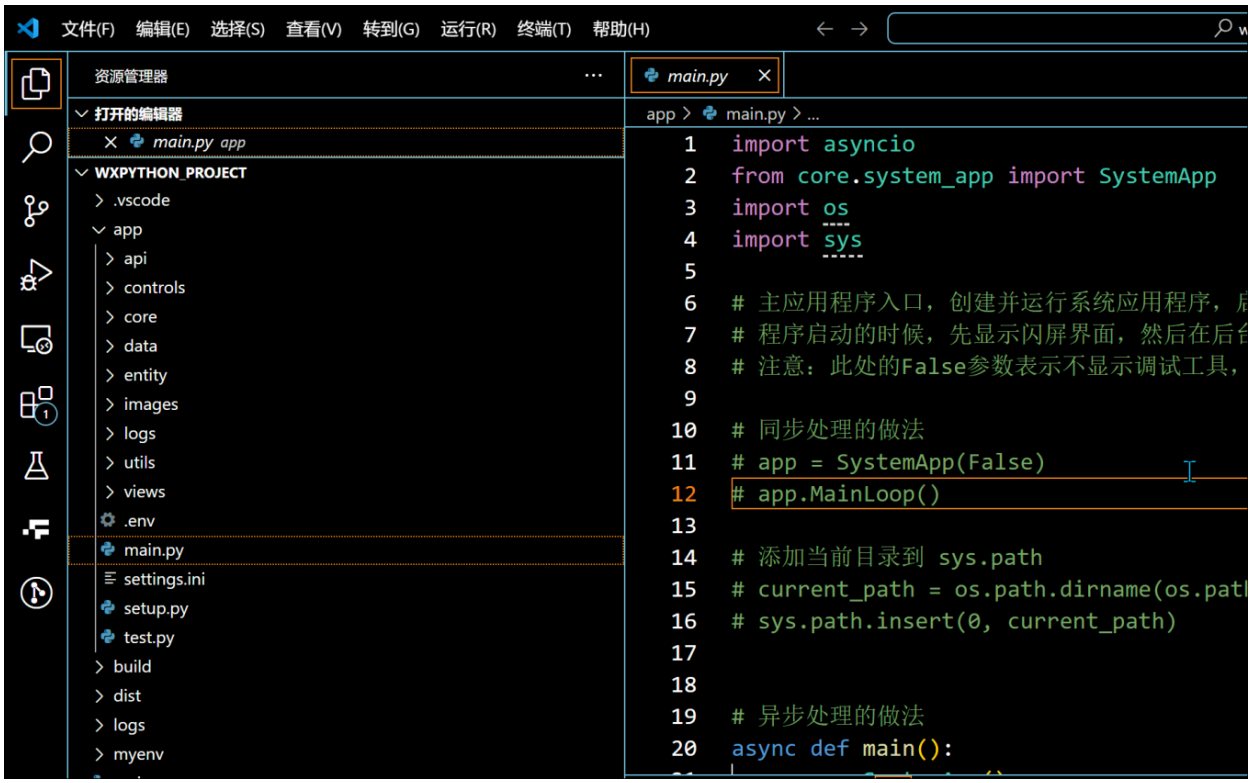
在展开数据库信息后，可以再代码生成工具的主工具栏或者右键菜单上执行 WxPython 前端代码生成的操作，如下界面所示。



选择相关的数据表后，一键生成相关的代码，如下所示。



最后把相关的增量生成的目录文件，复制到项目目录 app 上即可，然后在 VSCode 上做相关的调整修改。



4 Vue3+ElementPlus 前端开发介绍

4.1 理解 Vue3+ElementPlus 前端项目结构

该前端项目也是基于 VSCode 进行开发的，基于最新的 Vue3 +TypeScript + ElementPlus 技术栈开发的 Vue 前端，整合了众多前端流行组件，以 ElementPlus 界面组件为基础，拓展很多相关组件，界面设计模块化处理

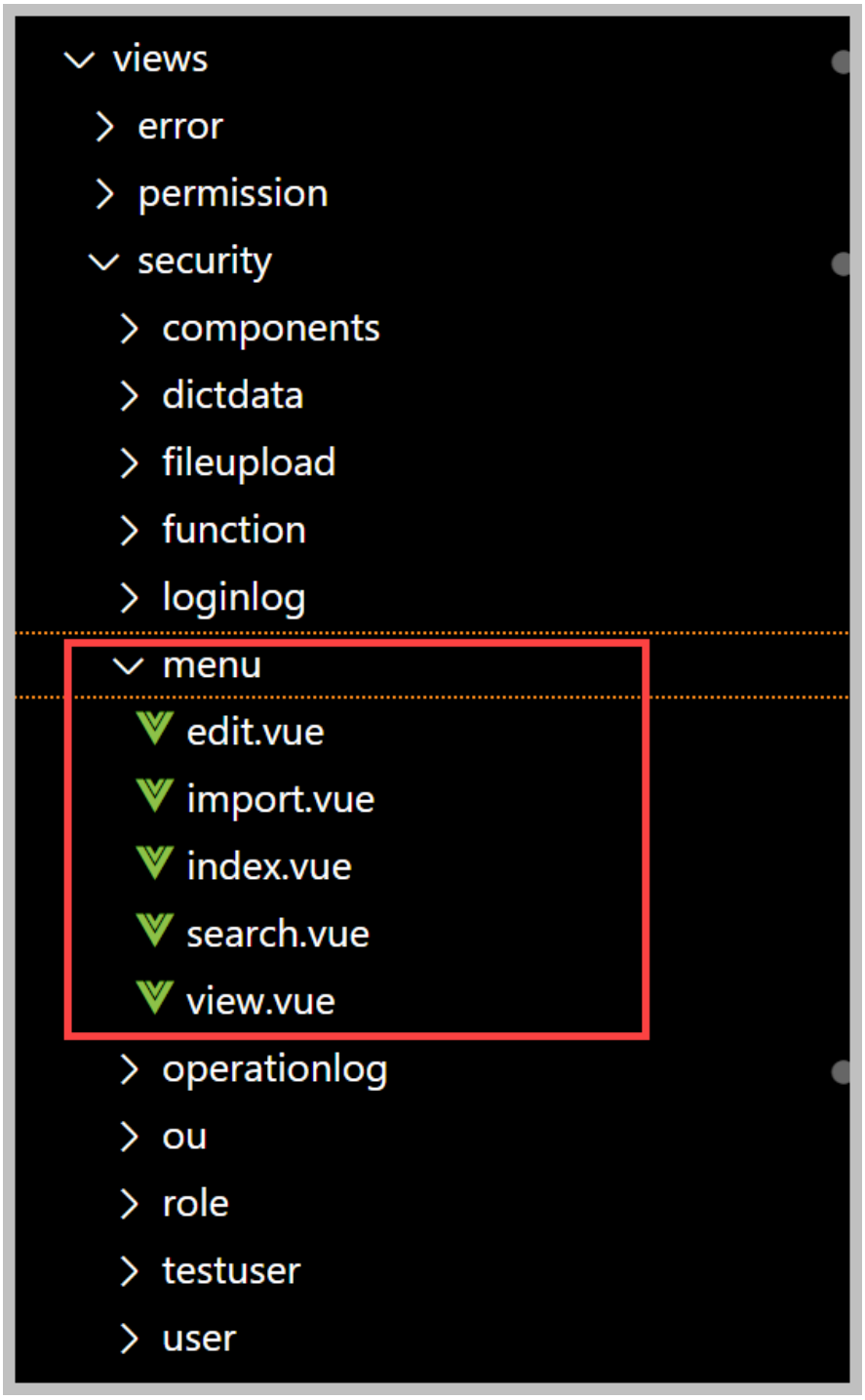
严格的前后端分离，前端通过 API 访问获得数据进行展示和增删改查处理。前端整合模块包括用户管理、组织机构管理、角色管理、菜单管理、功能管理及权限分配，日志管理、字典管理、附件管理等管理功能。

Vue3+TypeScript+ElementPlus 前端界面，常规页面的内容的整体界面布局，它包含常规的列表界面，新增、编辑、查看、导入等界面，除了列表页面，其他内容以弹出层对话框的方式进行处理，如下界面示意图所示。

Vue3+TypeScrip 最大的特点就是组件模块的便利性。

我们通过抽取共共性的内容组成组件，从而不同的页面内容，只需要维护不同的文件即可，从而隔离变化，提高代码的可读性和可维护性。

根据以上的页面划分，我们把一个页面分为 search.vue、edit.vue、import.vue、view.vue、index.vue，其中 index.vue 为整合各个组件的主页面，在视图中如下所示。我们每个业务模块都是如此统一划分，因此比较统一，同时也是为后续的代码生成工具批量生成做好准备。



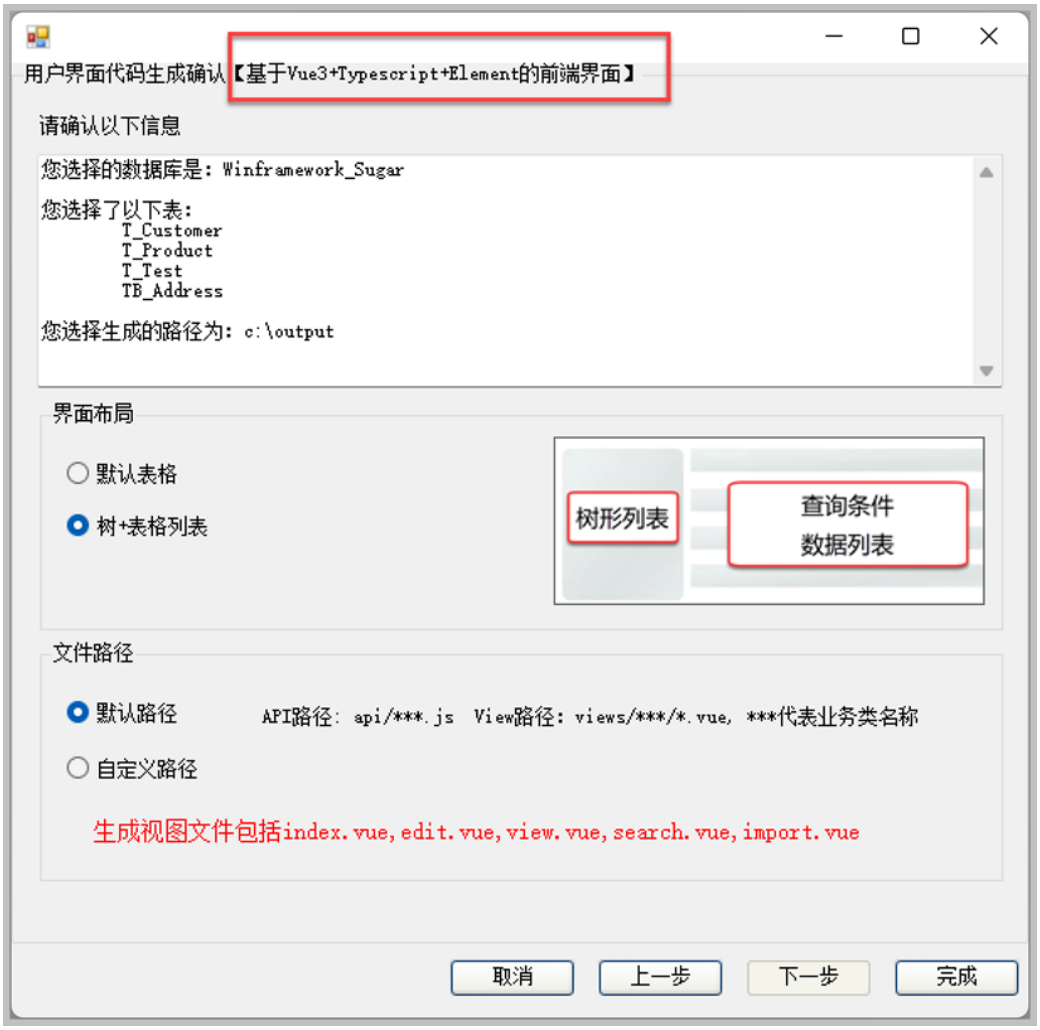
因此在 index.vue 页面中，我们整合了几个组件页面即可。

```
<template>
  <div class="main">
    <!--条件及列表展示-->
    <Search ref="searchRef" @show-import="showImport" @show-add="showAdd" @show-view="showView" @show-edit="showEdit" />

    <!--查看详细组件界面-->
    <view-data ref="viewRef" />
    <!--新增、编辑组件界面-->
    <edit-data ref="editRef" @submit="refreshData" />
    <!--模板导入信息-->
    <import-data ref="importRef" @finish="finishImport" />
  </div>
</template>

<script setup lang="ts">
import { reactive, ref, onMounted } from 'vue';

import Search from './search.vue';
import ViewData from './view.vue';
import EditData from './edit.vue';
import ImportData from './import.vue';
```

这样我们只需要维护查询页面、编辑页面、查看页面、导入页面的内容就可以了，而这些主要反映在相关的字段信息上，大的方向上我们已经根据页面的布局设置好了。

常规的列表页面内容，包含一些查询条件，以及相关的入口按钮事件的处理，如下界面所示

而如果是需要显示树列表的内容，我们在代码生成的时候，选择树列表界面生成即可，大致效果如下所示。



4.2 Vue3+ElementPlus 前端界面开发

代码生成工具 Database2sharp 配套工具实现前端界面和 API 代理 ES6 类的快速生成，由于页面模块化，开发和维护非常方便。

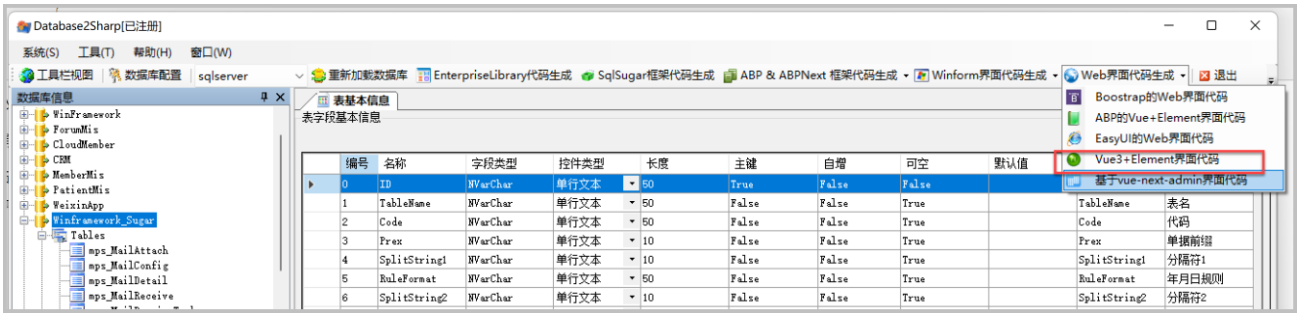
在开发前，我们需要根据业务需求设计好相关的数据库表，如设计一个通用的编码规则，对这些元素进行组合配置，数据库设计如下所示。

	Name	Code	Comment	Data Type
➡	ID	ID		nvarchar(50)
2	TableName	TableName	表名	nvarchar(50)
3	Code	Code	代码	nvarchar(50)
4	Prex	Prex	单据前缀	nvarchar(10)
5	SplitString1	SplitString1	分隔符1	nvarchar(10)
6	RuleFormat	RuleFormat	年月日规则	nvarchar(50)
7	SplitString2	SplitString2	分隔符2	nvarchar(10)
8	ValueLength	ValueLength	流水号长度	int
9	CurrentValue	CurrentValue	当前流水号	int
10	Suffix	Suffix	后缀	nvarchar(10)
11	CurrentNumberString	CurrentNumberString	完整序列号字符串	nvarchar(100)
12	Seq	Seq	排序	nvarchar(50)
13	Note	Note	备注	nvarchar(255)
14	Creator	Creator	创建人	nvarchar(50)
15	CreateTime	CreateTime	创建时间	datetime
16	LastGenerateTime	LastGenerateTime	最后生成日期	datetime

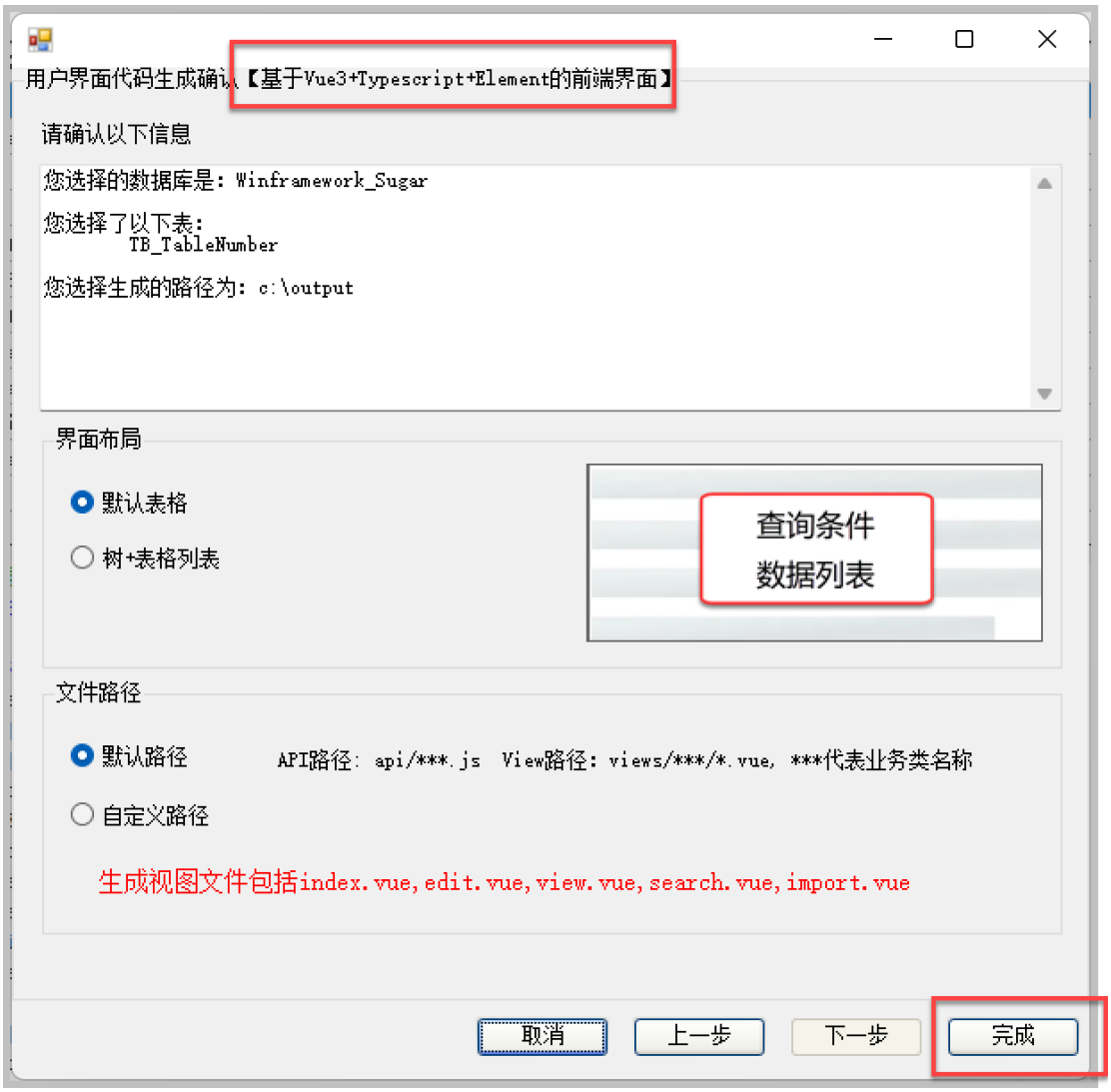
然后通过上面 PowerDesigner 工具生成 SQL 脚本后，在实际的数据库中创建对应的数据库表，从代码生成工具中展开数据库信息。

在代码生成工具的数据库列表右键上找到上面的功能入口（或者在工具栏的 Web 界面代码生成中选择）。

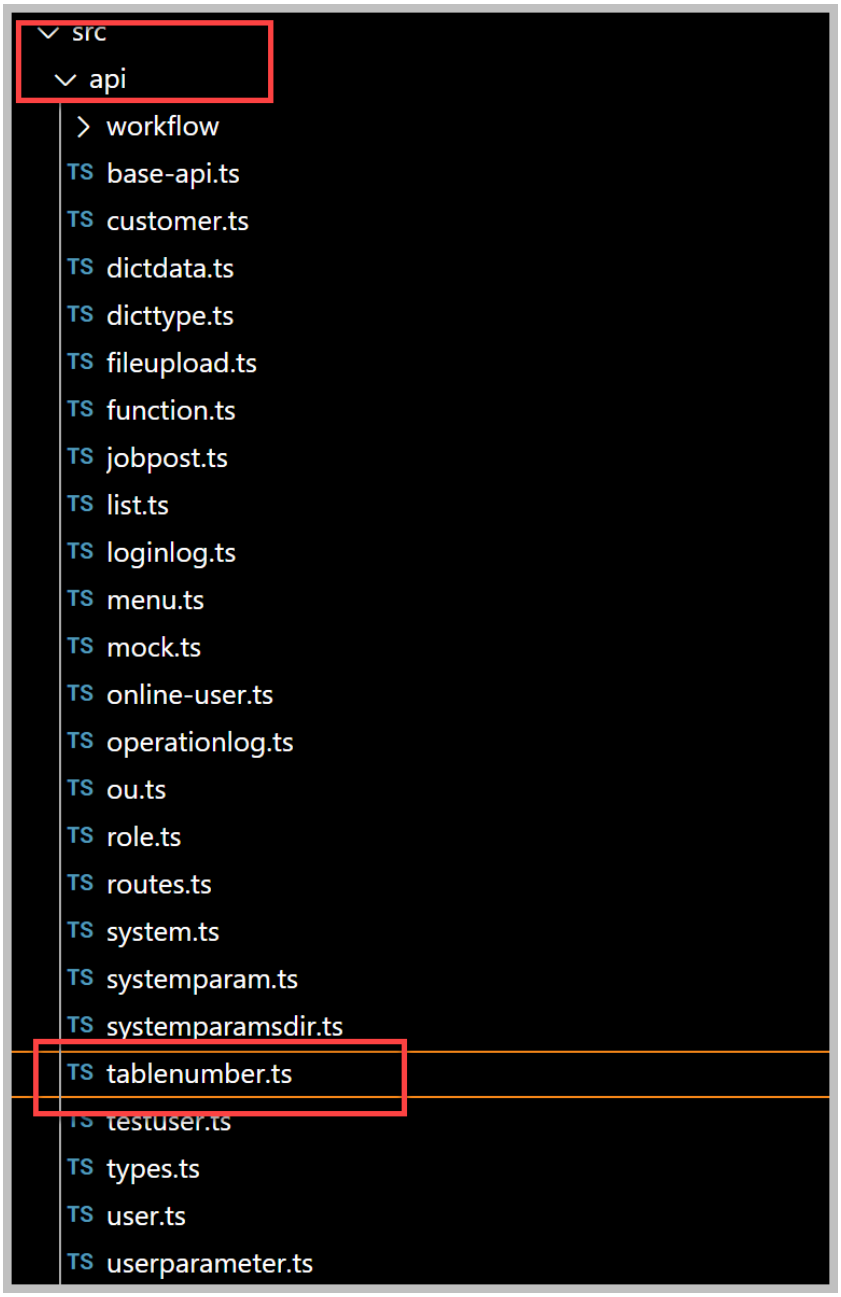
或者从工具栏中选择【Vue3+Element 界面代码生成】



最后选择该业务处理的表，生成相关的界面代码，其中包括了对 WebAPI 的远程调用封装的 API 客户端类，以及 View 视图界面。

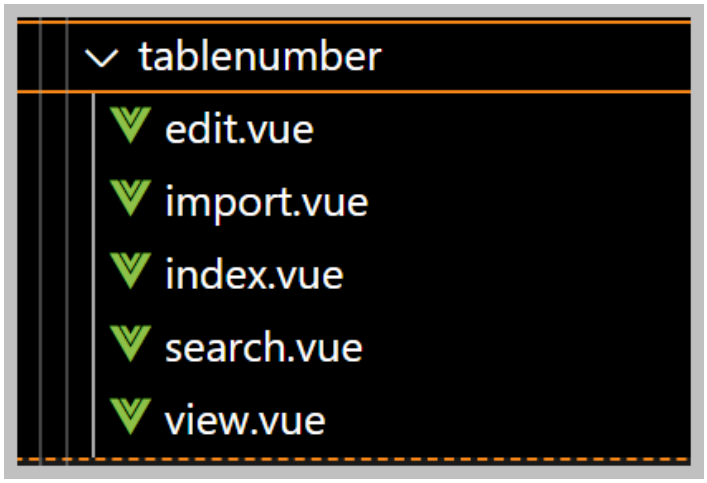


其中 Vue3+Element 前端的 API 类如下位置复制过去，放在 Src/api 目录下，这个是统一放置相关 Web API 调用的 JS 的 ES6 类。



而视图代码，我们复制到对应的 views 目录上即可，具体位置可以根据实际的需要移动目录处理。

使用代码生成工具，直接生成的视图包含了几个文件，这些文件就是各个模块的组件定义，如下视图代码所示。



它们最终是整合呈现在 index.vue 的视图入口中，我们可以适当的调整一下相关的界面代码。

在我们测试界面前，我们需要把静态路由添加到系统中去，我们找到对应模块的路由定义信息，如下所示。

添加上刚才的页面路由地址，如下所示。

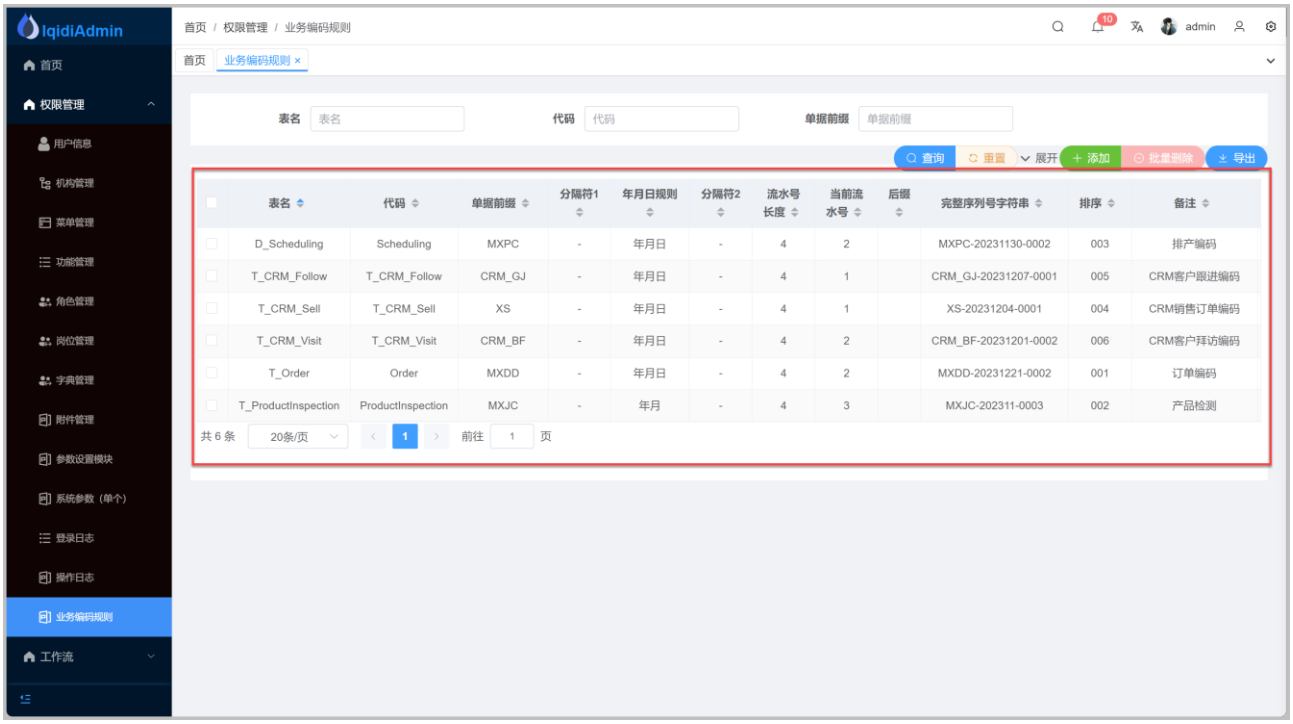
```
TS security.ts
src > router > modules > TS security.ts > securityRouter > children
114       }
115     },
116     {
117       path: '/operationlog/index',
118       name: 'Operationlog',
119       component: () => import('/@/views/security/operationlog/index.vue'),
120       meta: {
121         title: $t(操作日志),
122         icon: 'ppt'
123       }
124     },
125     {
126       path: '/tablenumber/index',
127       name: 'Tablenumber',
128       component: () => import('/@/views/security/tablenumber/index.vue'),
129       meta: {
130         title: $t(业务编码规则),
131         icon: 'ppt'
132       }
133     }
134   ]
135 };
136
137 export default securityRouter;
138
```

至此，我们构建了一个完整的界面和入口了，可以在 VSCode 的控制台中运行前端界面测试了，测试需要结合其中 Web API 的后端。

```
问题 5   输出   调试控制台   终端   端口   注释

  VITE v4.3.4  ready in 7620 ms

  ➜  Local:   http://localhost:8848/
  ➜  Network: http://192.168.1.103:8848/
  ➜  Network: http://192.168.102.1:8848/
  ➜  Network: http://192.168.109.1:8848/
  ➜  Network: http://172.25.48.1:8848/
  ➜  press h to show help
```

运行界面，登录后进入界面，可以查看相关的菜单，然后查看《业务编码规则》界面信

息了。



查看界面和编辑界面分属不同的视图界面，查看界面效果如下所示。

编辑界面效果如下所示，微调了界面效果，并增加了一个测试生成的按钮。