广州爱奇迪软件科技有限公司(http://www.iqidi.com) Mail:wuhuacong@163.com

PySide/PyQt 桌面端

系统架构设计说明书

版本： 1.0

编制人：伍华聪

目录

1. 引言

1.1. 背景

我们一直致力于各种应用开发框架的研究和开发，著有《Winform 开发框架》、《WCF 开发框架》和《混合式开发框架》，以及《微信开发框架》、《Bootstrap 开发框架》、《ABP/ABP VNext 快速开发框架》、《SqlSugar 开发框架》、《WPF 开发框架》等众多框架，主要为客户快速构建项目做基础框架研究，并结合我们的专用代码生成工具，基于数据库信息实现项目的快速开发，提高开发效率。

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。

以上众多框架多数是基 .NET 系的，由于随着大环境的变化，跨平台系统的需求越来越多，对于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。

《Python 开发框架》是严格的前后端分离模式，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。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》。

框架后端是基于 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 前端为基于 Python + WxPython 的桌面应用前端，基于 Python + PySide/PyQt 的桌面应用前端，以及基于 BS 的前端 Vue3+ElementPlus+TypeScript 前端。后面会继续完善和扩展其他前端等内容。

1.2. 设计目标

为了实现新增业务系统的快速开发，以及提高系统的健壮性，系统的设计目标满足下面要求。

1) 在完善的《Python 开发框架》基础上开发新的业务系统，减少重复劳动。
2) 权限管理系统、数据字典模块等基础模块，和业务系统分离，能为其他项目系统提供相应的基础功能服务。
3) 支持多种数据库业务系统应用的解决方案，配置即可自动切换。
4) 严格的前后端分离模式，前后端均能够实现跨平台开发和跨平台运行模式。
5) 使用专用定制化的代码生成工具辅助，快速搭建相关的业务框架系统。
6) 界面、模块、公用代码复用性高，降低重复代码，提高开发效率。

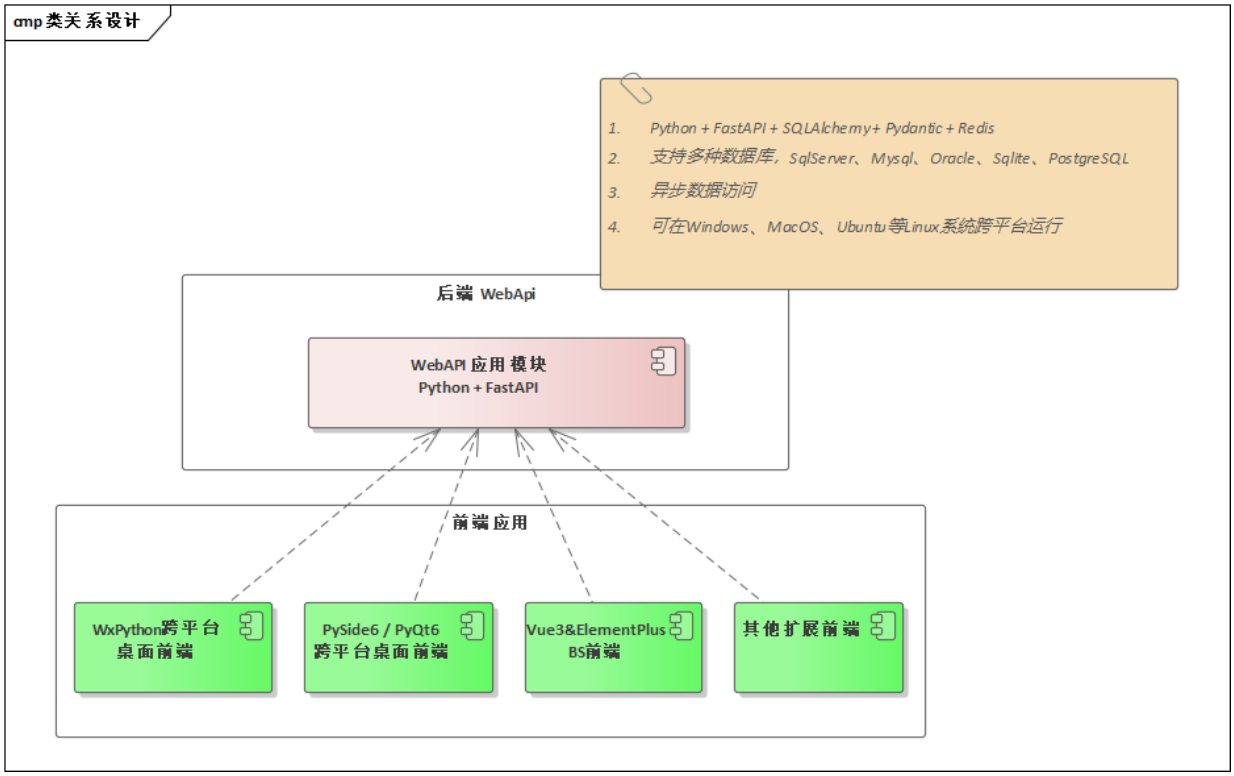
7) 只需关注业务系统的核心业务实现，框架、类库、独立模块、控件简化、封装常用操作。

8) 良好封装、开发高效、界面友好、强壮完善等特点。

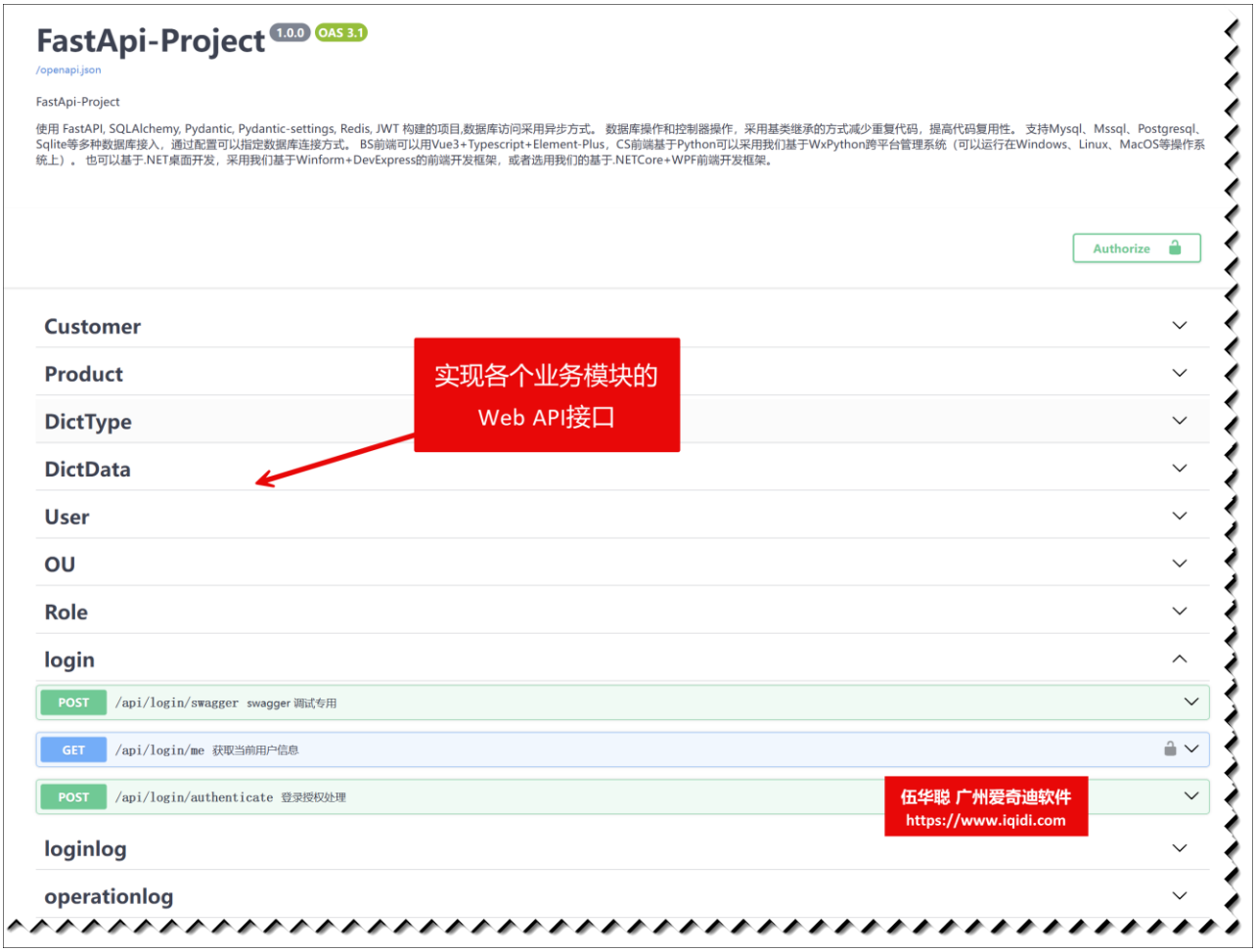
2. 架构视图

2.1. 开发框架中前后端之间的关系

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。

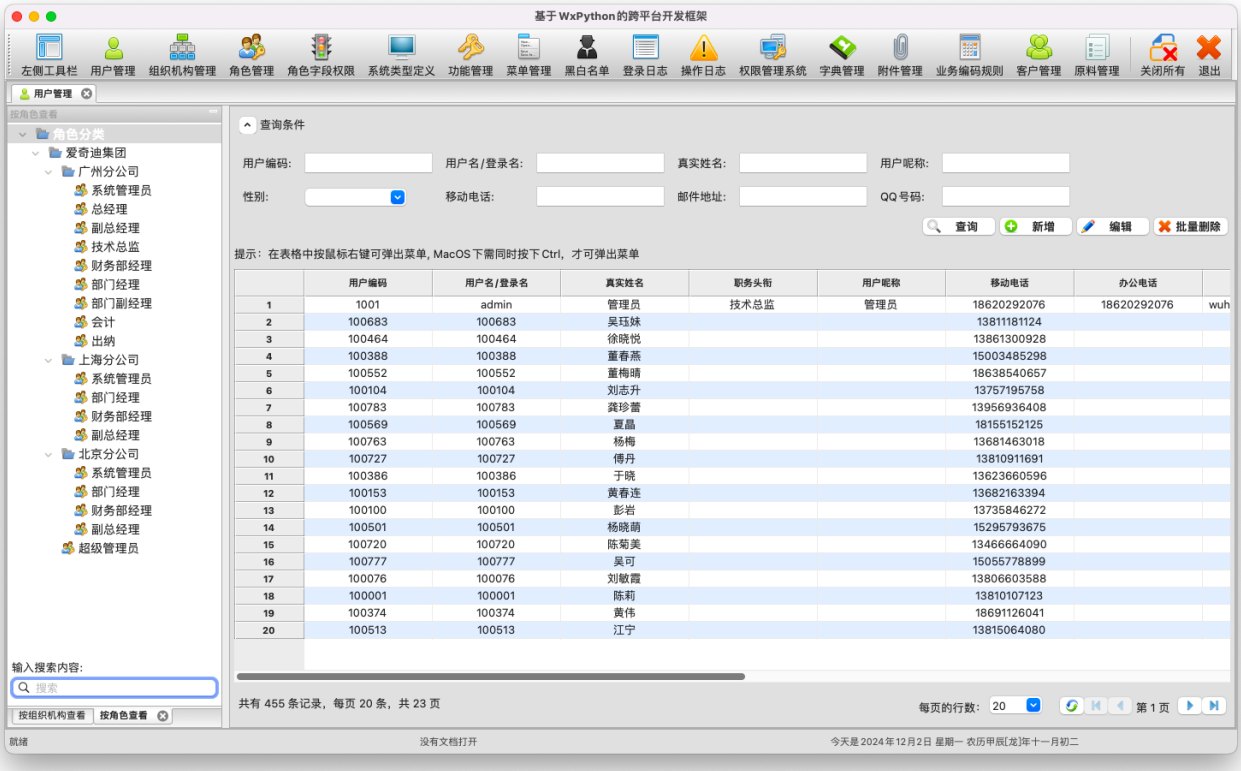


FastAPI 项目运行后，其界面自动整合 Swagger 的文档界面，如下所示。



WxPython 的桌面应用前端和 PySide/PyQt 的桌面应用前端， 都是一个多文档界面布局的应用程序，能够在 Windows、MacOS、Ubuntu 等 Linux 系统上运行。

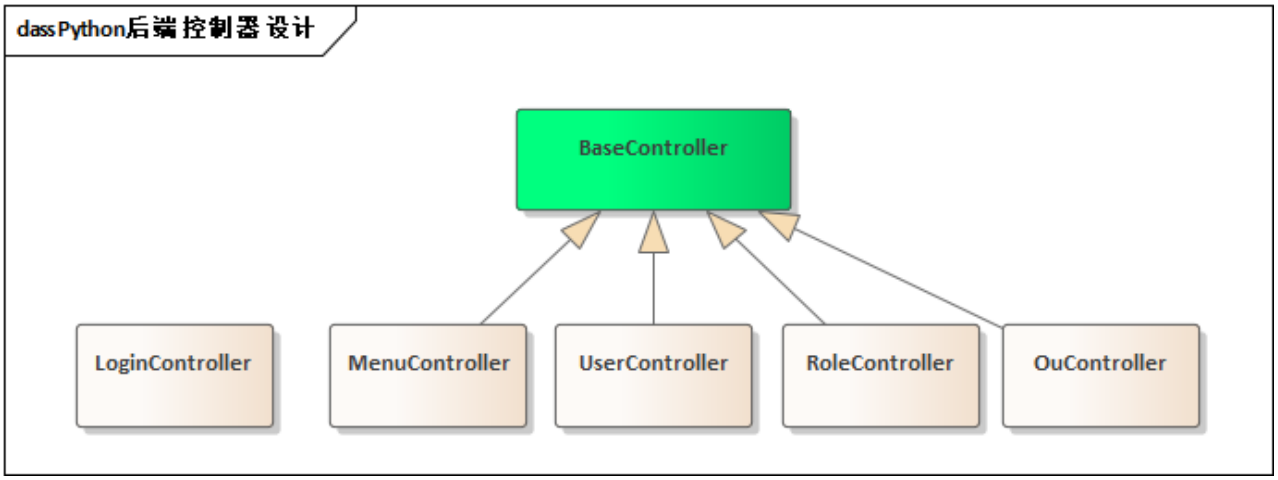
WxPython 的桌面应用前端，界面效果如下所示。



PySide/PyQt 的桌面应用前端，界面效果如下所示。

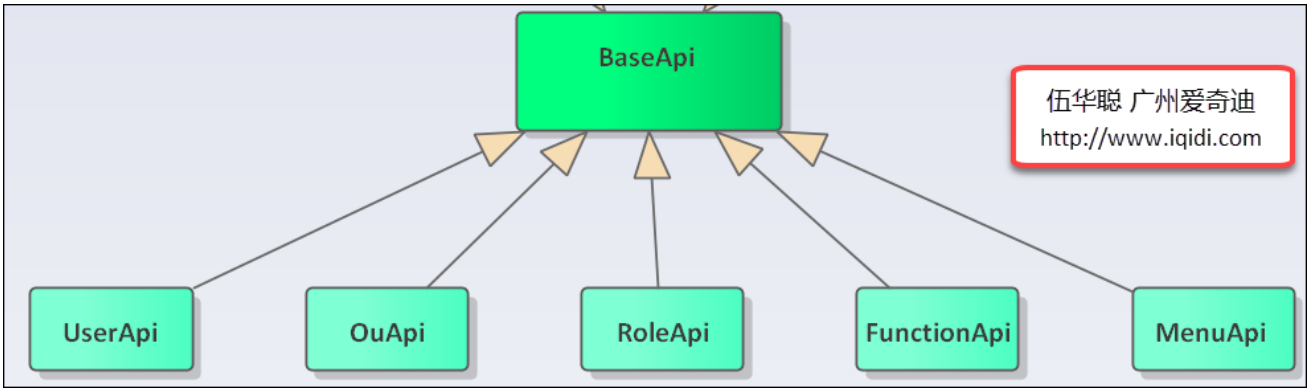
2.2. Web API 接口和桌面端的调用

在 FastApi 的后端项目中，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，这样在基类中，可以统一实现常规的增、删、改、查、排序等操作。



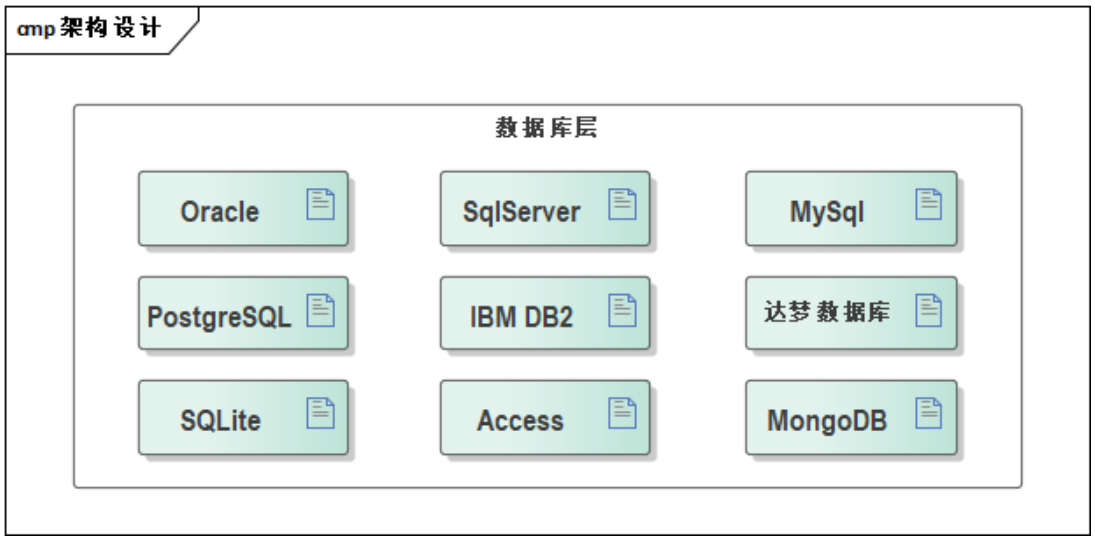
同样，在 WxPython 的桌面端和 PySide/PyQt 的桌面端中，我们也一样对 Web API 的调用类进行统一的封装处理，这样增、删、改、查、排序等处理的接口都可以抽象到 BaseApi 里面了。

如对于权限模块我们涉及到的用户管理、机构管理、角色管理、菜单管理、功能管理、操作日志、登录日志等业务类，有如下所示继承关系。



2.3. 多数据库支持的设计

一个良好的产品，可能往往需要支持多种数据库的接入，根据实际业务的需要进行调整，有时候可能需要 2 到 3 种数据库的支持。



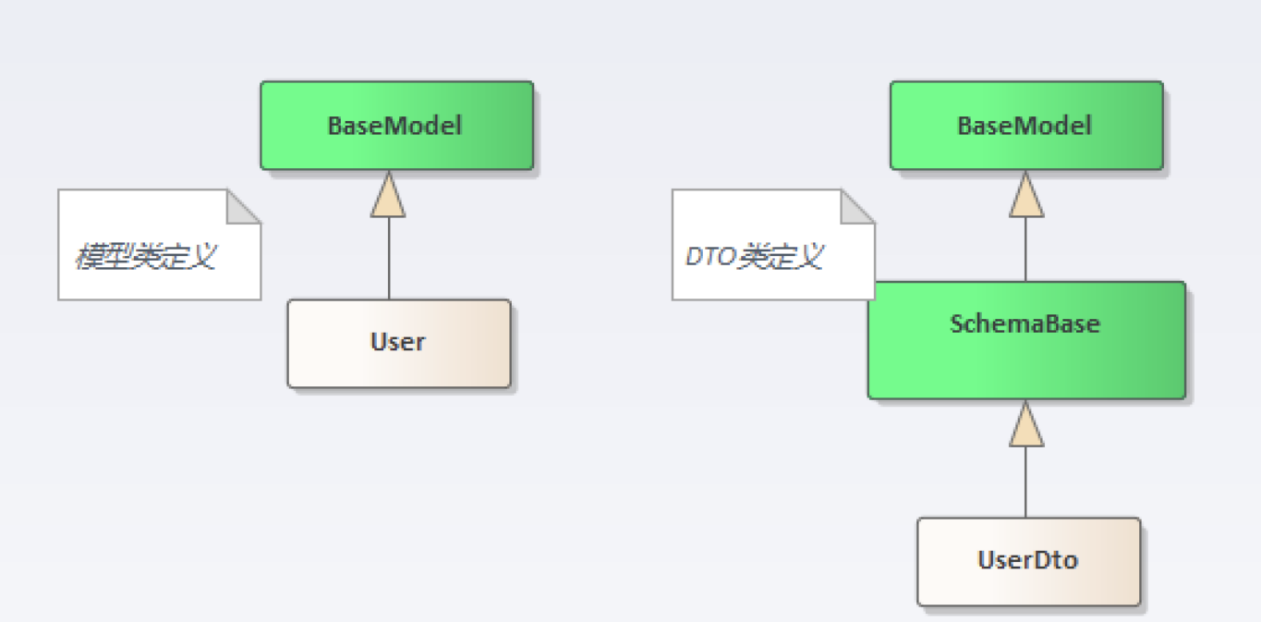
在我们使用 Python 来和数据库打交道中，**SQLAlchemy** 是一个非常不错的 ORM 工具，通过它我们可以很好的实现多种数据库的统一模型接入，而且它提供了非常多的特性，通过结合不同的数据库驱动，我们可以实现同步或者异步的处理封装。

我们在后端框架中，使用 **SQLAlchemy** 2.X 来实现多数据库的接入以及异步的数据库操作处理。

SQLAlchemy 是一个功能强大且灵活的 Python SQL 工具包和对象关系映射（ORM）库。它被广泛用于在 Python 项目中处理关系型数据库的场景，既提供了高级的 ORM 功能，又保留了对底层 SQL 语句的强大控制力。SQLAlchemy 允许开发者通过 Python 代码与数据库进行交互，而无需直接编写 SQL 语句，同时也支持直接使用原生 SQL 进行复杂查询。下面是 **SQLAlchemy** 和我们常规数据库对象的对应关系说明。

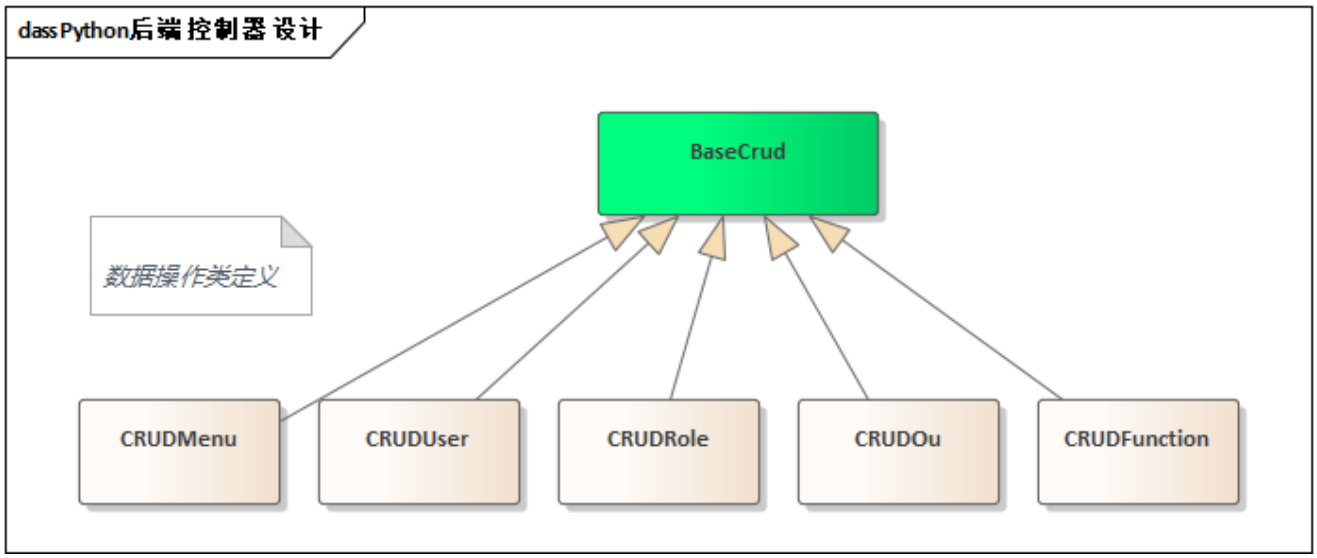
Engine	连接对象	驱动引擎
Session	连接池	事务 由此开始查询
Model	表	类定义
Column	列	
Query	若干行	可以链式添加多个条件

利用 **SQLAlchemy** 其模型的定义，我们很好的实现多数据库的接入。



在配置文件.env 文件中配置指定的数据库信息即可。

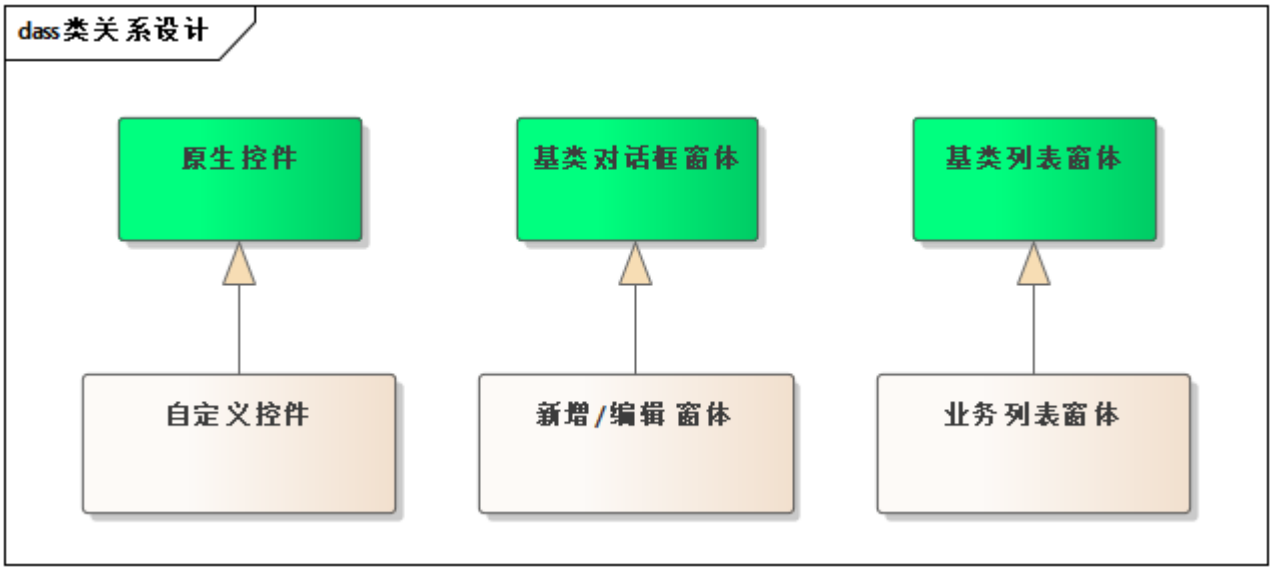
同时，对于数据库的常规增删改查等通用性的数据表的处理，我们往往把它们的功能，利用泛型的特性来抽象到基类上进行重用。这样降低子类代码的编写，有利于前端后端的相同接口处理。



2.4. 桌面端的界面组件设计

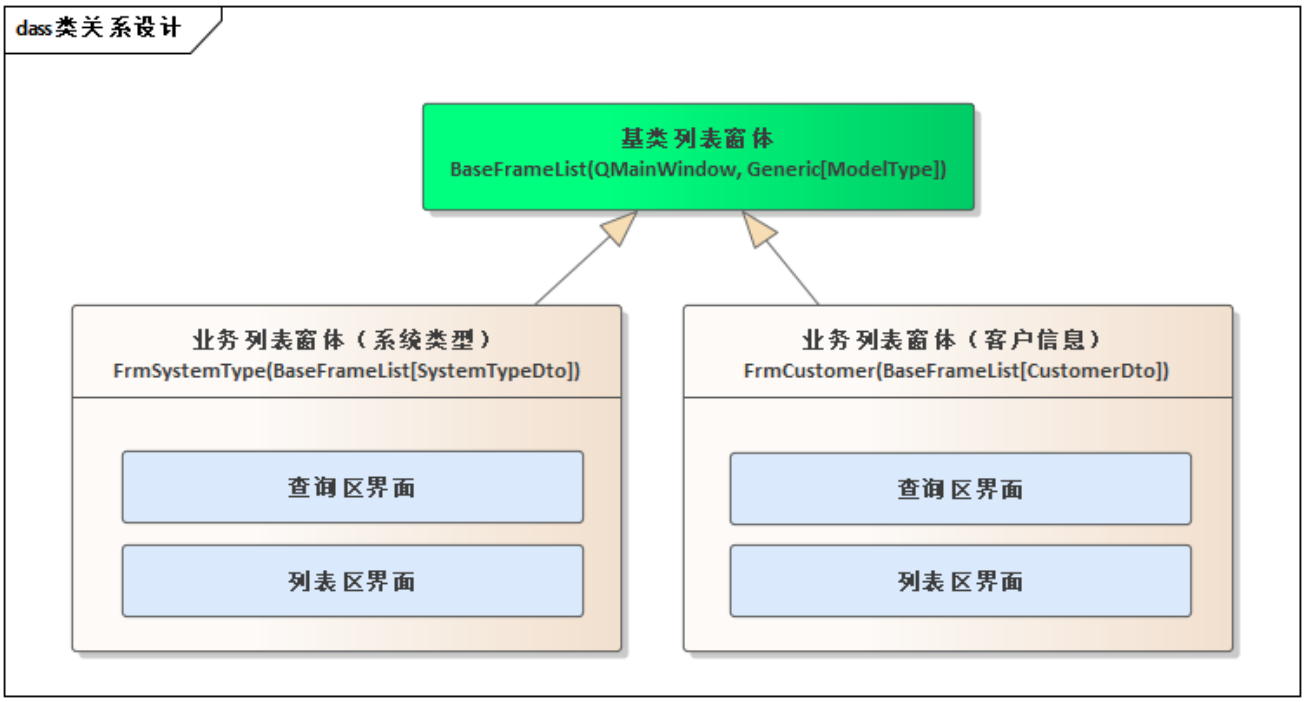
要降低一个程序的编码数量，提高开发效率，很大程度上就是提高基类的通用性和封装逻辑，通过重用具体的实现或者重用一定的逻辑，可以大大降低重复代码的编写，提高代码的可读性和可维护性。

在前端界面中，一般我们把不同的视图窗口归类，有些是列表查询展示的，有些是单条记录展示的，有些则是自定义用户控件元素的，因此他们可以分为几类进行不同的封装，以便统一界面处理演示，以及重用共同的处理逻辑等。

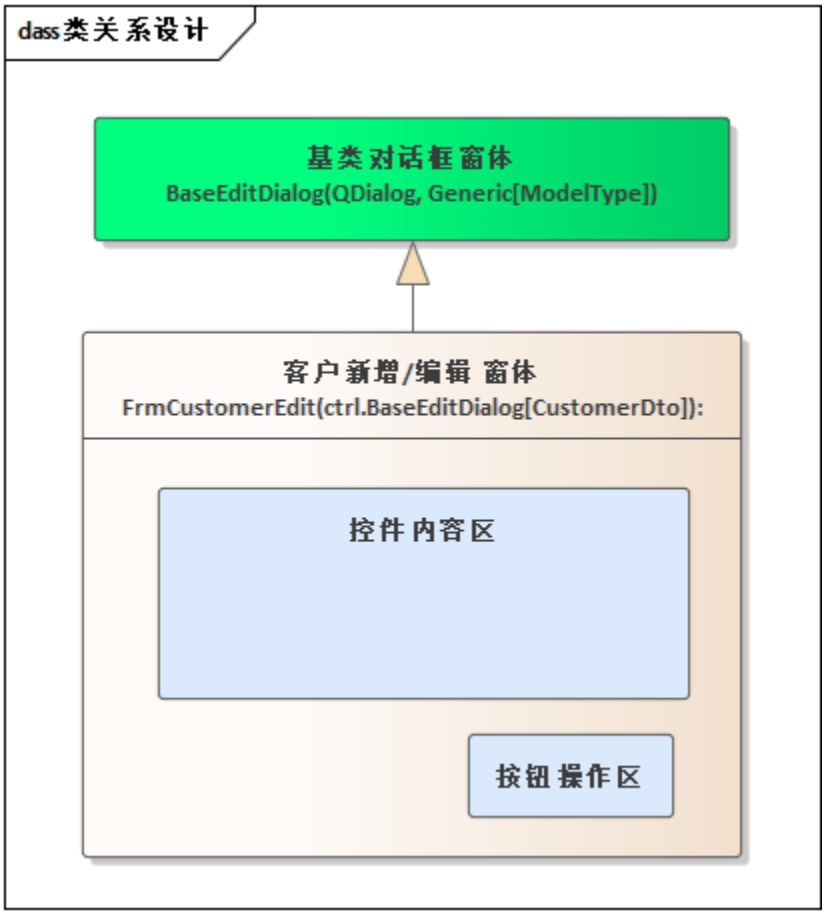


我们把常规的列表界面，新增、编辑、查看、导入等界面放在一起，除了列表页面，其他内容以弹出层对话框的方式进行处理，如下界面示意所示。

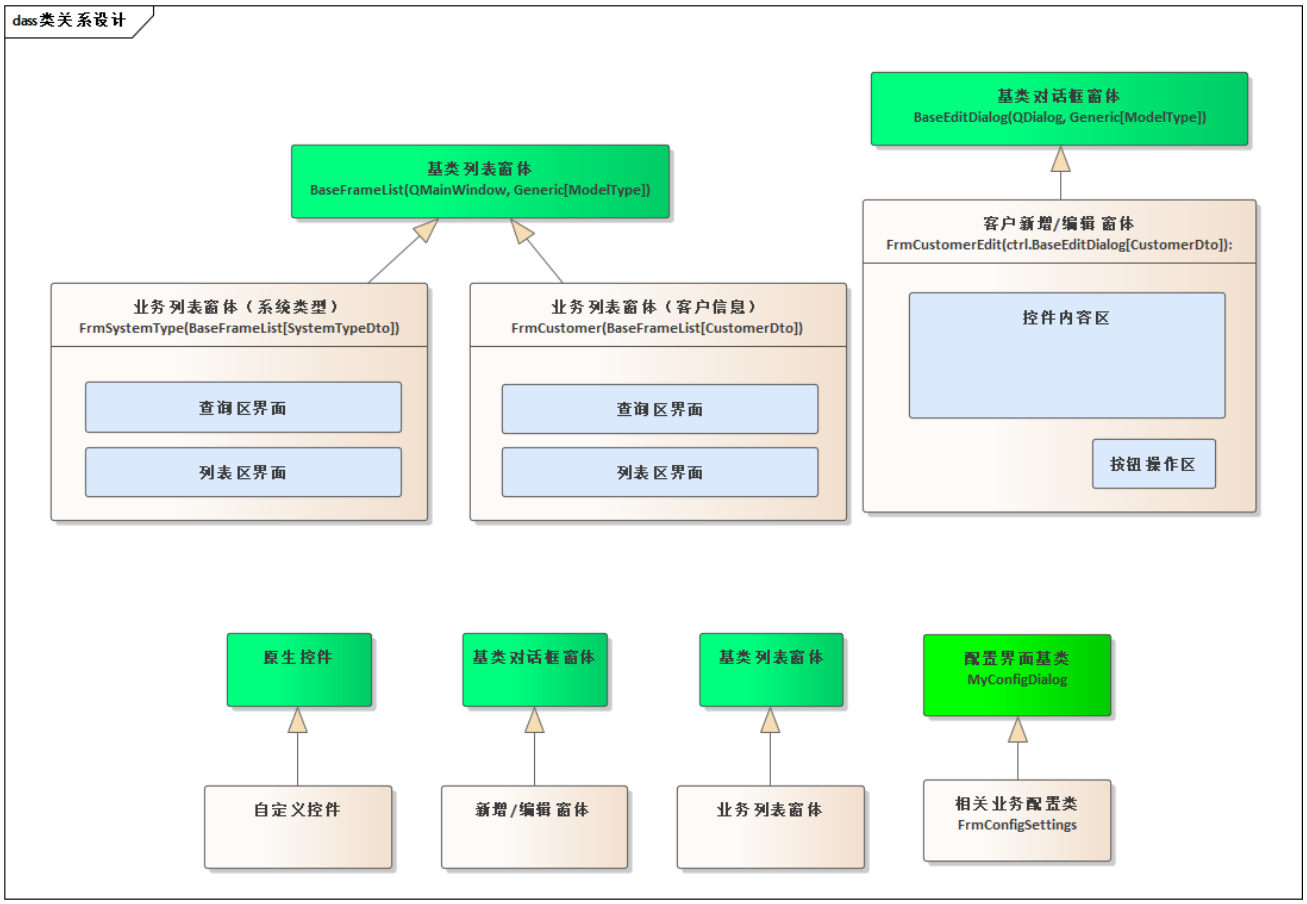
如对于两个例子窗体：系统类型定义，客户信息，其中传如对应的 DTO 信息和参数即可。



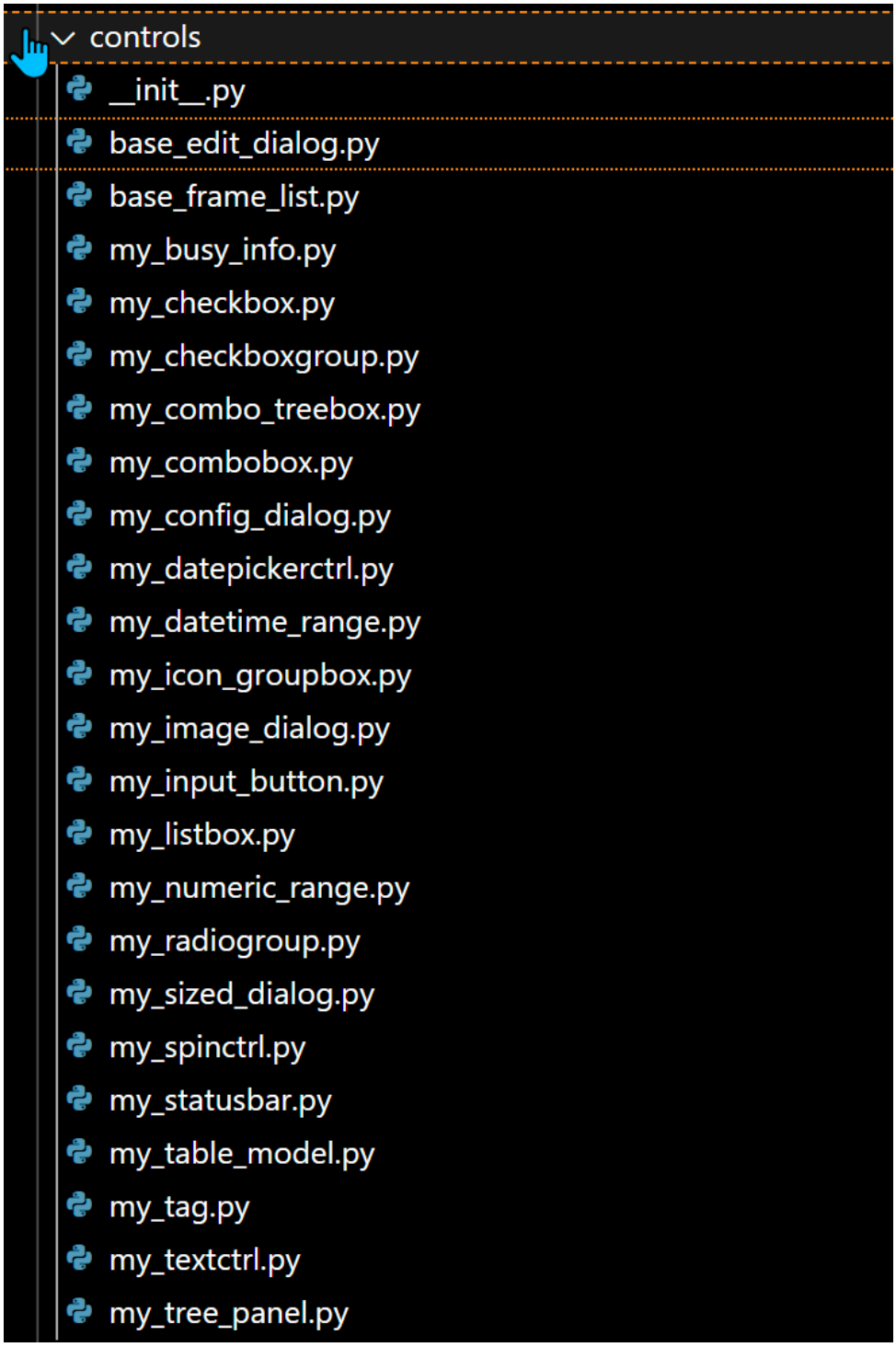
而一般的新增/编辑界面，继承基类编辑对话框，如下所示。



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。



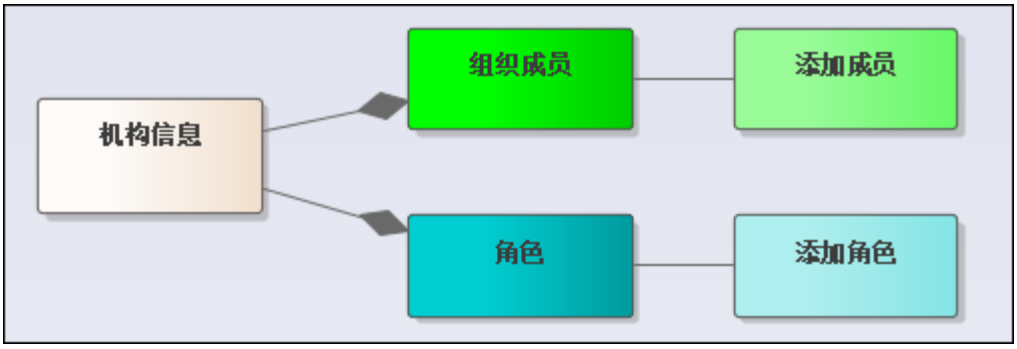
对于自定义控件，我们对其封装，使之能够在开发使用习惯上更一致，下面是我们根据需要对常见的原生控件进行一些自定义控件的封装列表。



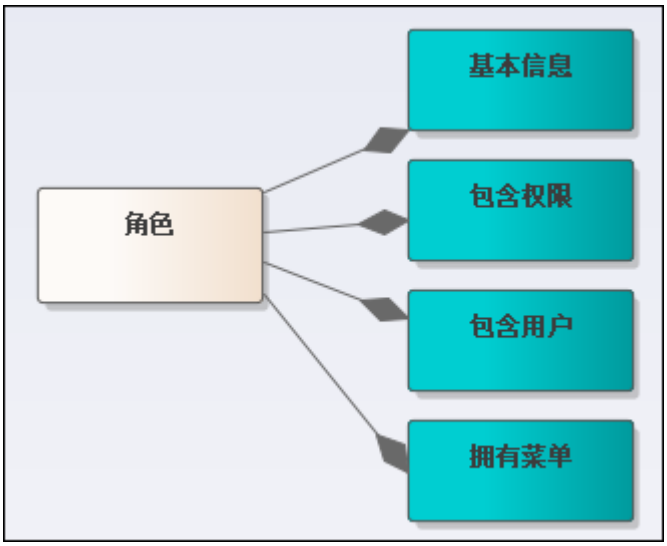
2.5. 界面模块的分拆

对于一个复杂的界面处理，我们还可以逐步细分他们的一些重复性元素，可以重用，也方便减少模块的代码，降低关注点。

如对于一个机构信息展示内容来说，可能包括了组织成员的信息，角色信息等，我们可以把它们逐一细化，细化的尺度以最大程度的重用代码为核心。



如一个角色的信息，可能包括多个方面，我们把它们按照一定的边界进行划分不同的用户控件，实现重用或者减少一个复杂模块的代码量。



3. 界面设计

3.1. 基于 PySide/PyQt 的界面设计

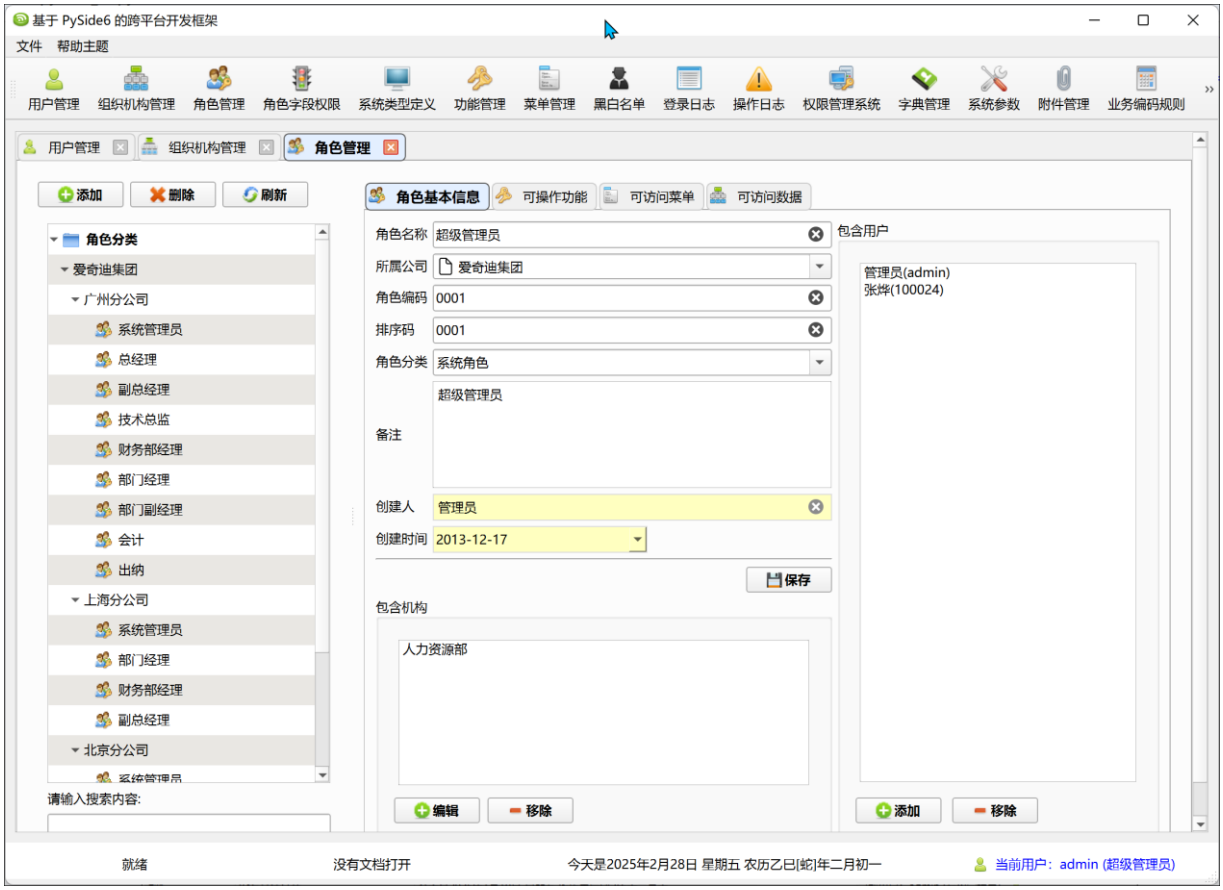
PySide/PyQt 桌面端是基于多文档界面的，可以一次性打开多个不同业务模块界面，列表界面效果如下所示。

工具栏和多文档界面是现代桌面应用程序中常见的界面元素，它们为用户提供了直观的操作和高效的工作流。工具栏通过将常用功能放置在一个可见的位置，让用户可以迅速访问和操作，无需通过多层菜单或子界面进行选择。工具栏的按钮通常使用图标（而非文字），这种视觉方式能减少用户的认知负担，让操作更加直观和高效。

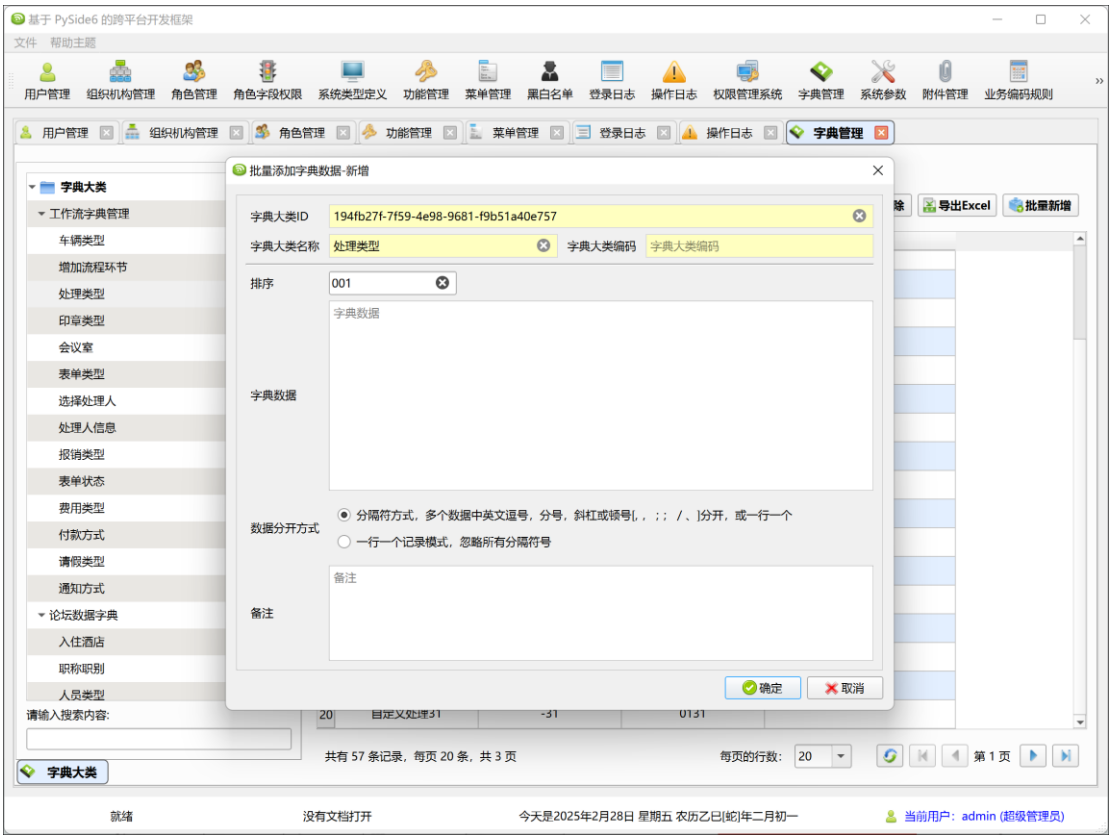
多文档界面（MDI）是一种设计模式，允许用户在同一个应用程序窗口中同时打开多个文档或视图，方便用户在不同任务之间进行快速切换。

树列表或者表格数据控件支持右键弹出菜单处理。

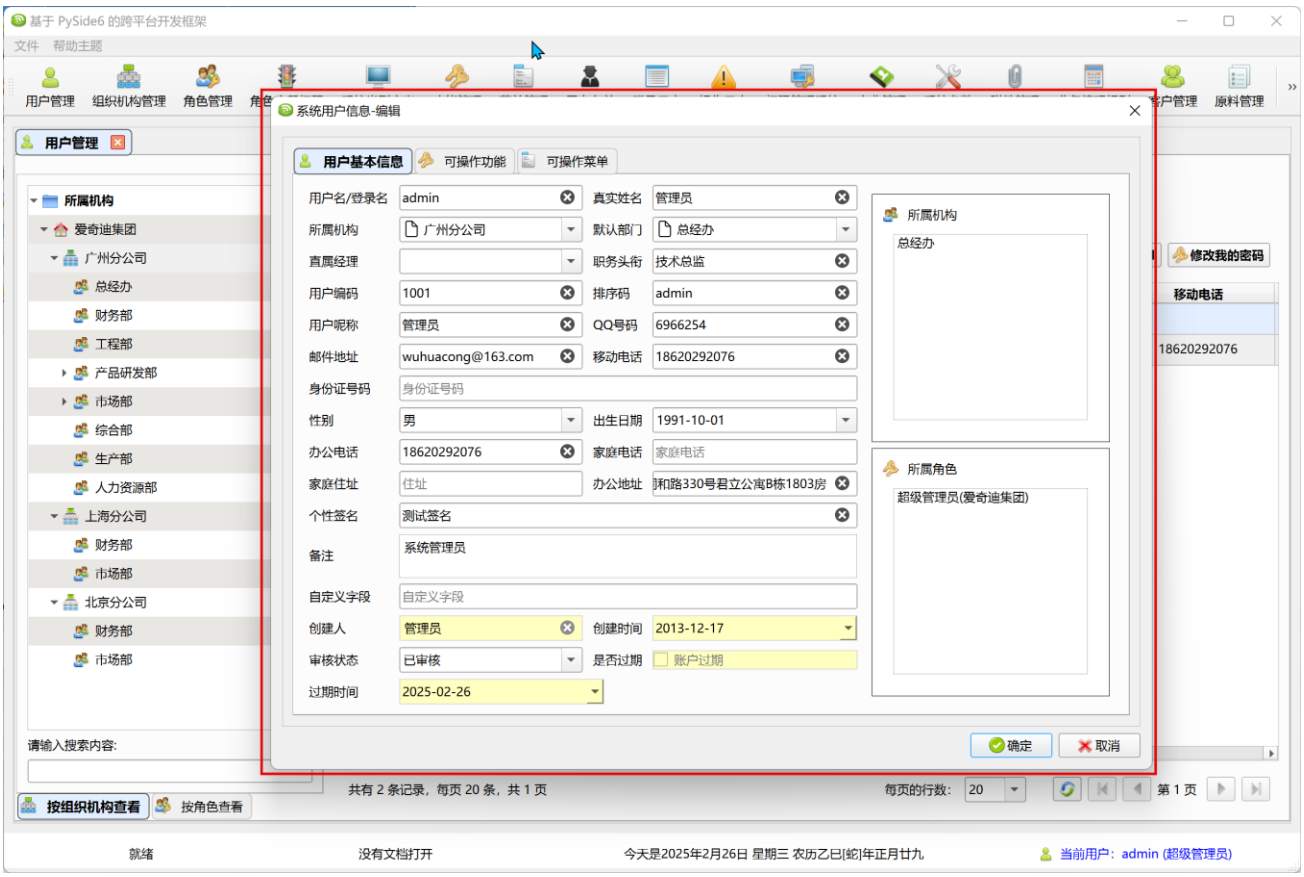
多个文档界面以选项卡方式展示在文档区域，如下界面所示。



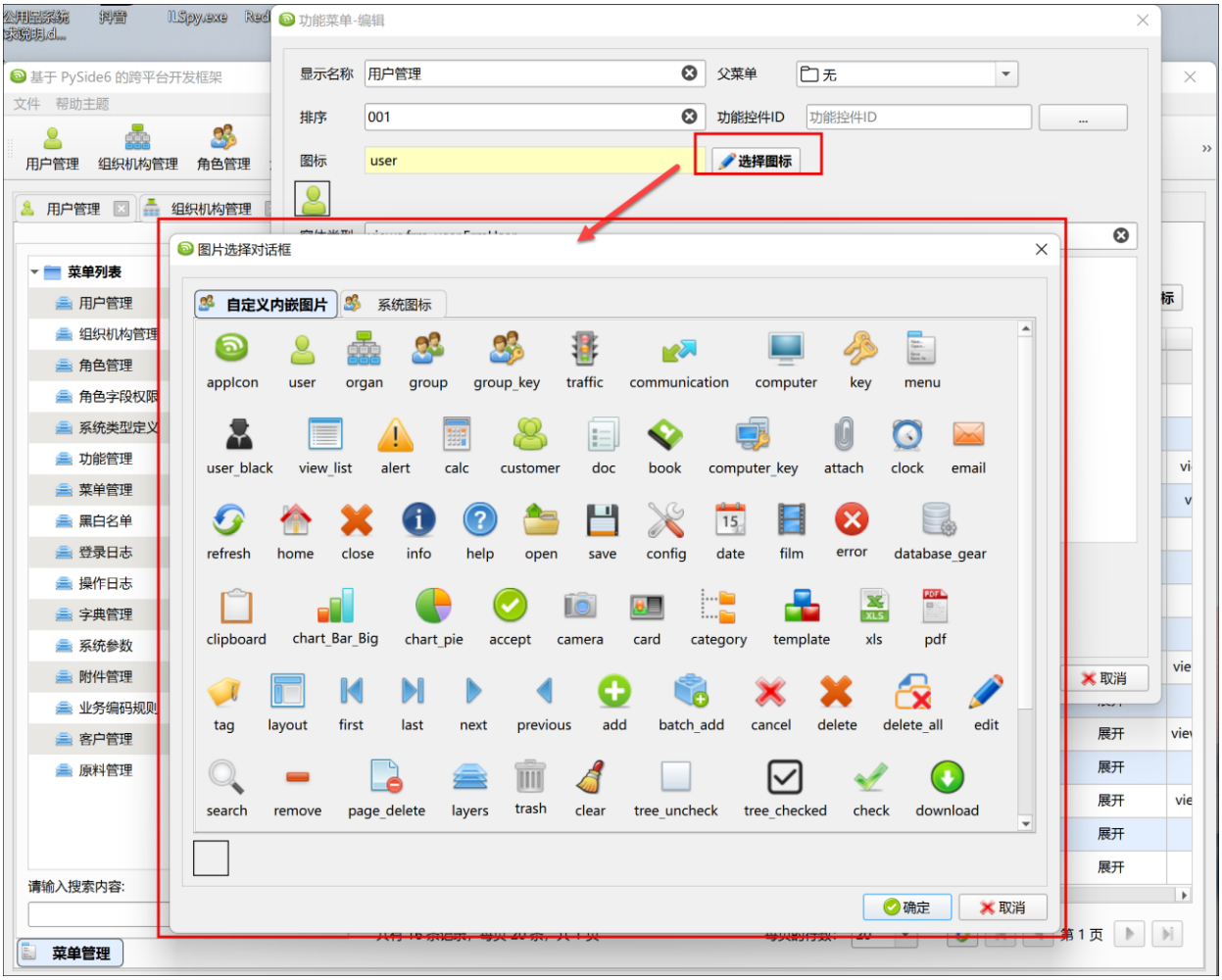
新增/编辑界面是弹出式、模态对话框的方式进行展示的。



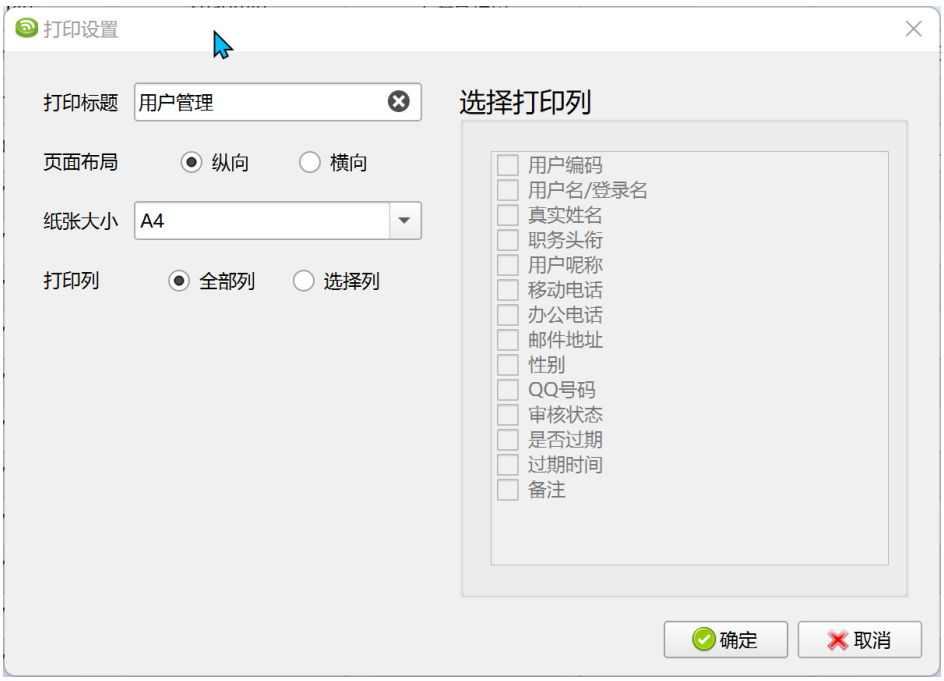
复杂一点的编辑界面，可以以多个选项卡的方式呈现复杂的内容。



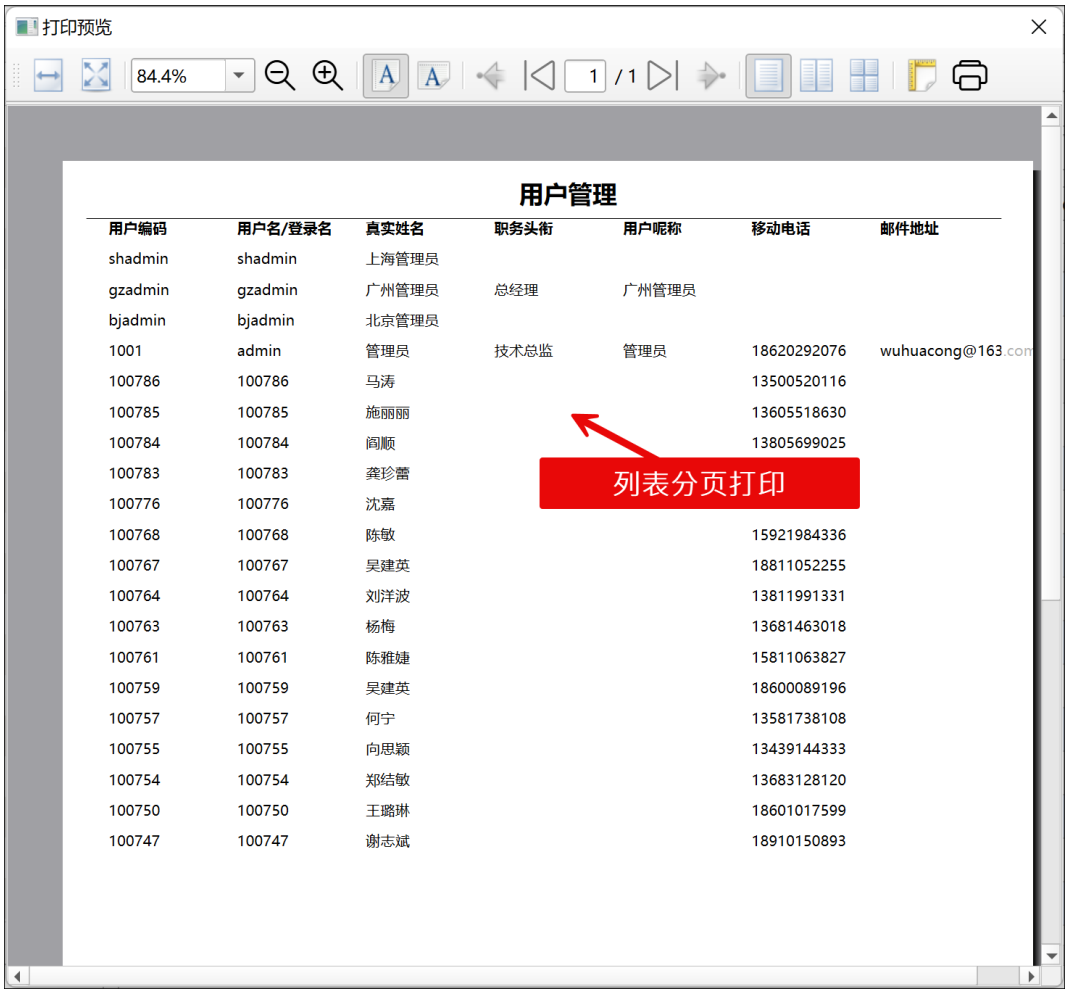
图标选择界面如下所示。


功能菜单-编辑
显示名称
用户管理
父菜单
无
排序
001
功能控件ID
图标
user
选择图标
图片选择对话框
自定义内嵌图片
系统图标
applcon
user
organ
group
group_key
traffic
communication
computer
key
menu
user_black
view_list
alert
calc
customer
doc
book
computer_key
attach
clock
email
refresh
home
close
info
help
open
save
config
date
film
error
database_gear
clipboard
chart_Bar_Big
chart_pie
accept
camera
card
category
template
xls
pdf
tag
layout
first
last
next
previous
add
batch_add
cancel
delete
delete_all
edit
search
remove
page_delete
layers
trash
clear
tree_uncheck
tree_checked
check
download
确定
取消
菜单列表
用户管理
组织机构管理
角色管理
角色字段权限
系统类型定义
功能管理
菜单管理
黑白名单
登录日志
操作日志
字典管理
系统参数
附件管理
业务编码规则
客户管理
原料管理
请输入搜索内容:
菜单管理

打印界面如下所示。


打印设置
打印标题
用户管理
页面布局
纵向
横向
纸张大小
A4
打印列
全部列
选择列
选择打印列
用户编码
用户名/登录名
真实姓名
职务头衔
用户昵称
移动电话
办公电话
邮件地址
性别
QQ号码
审核状态
是否过期
过期时间
备注
确定
取消

通用列表的打印提供预览界面，打印的内容，根据实际的列表数据进行分页预览。兼容在 MacOS 和 Windows 系统不同系统的效果。



4. 模块详细设计

4.1. 模块列表

我们把 PySide/PyQt 的桌面端程序内容，按照分层概念，大致可以分为下面几个模块。

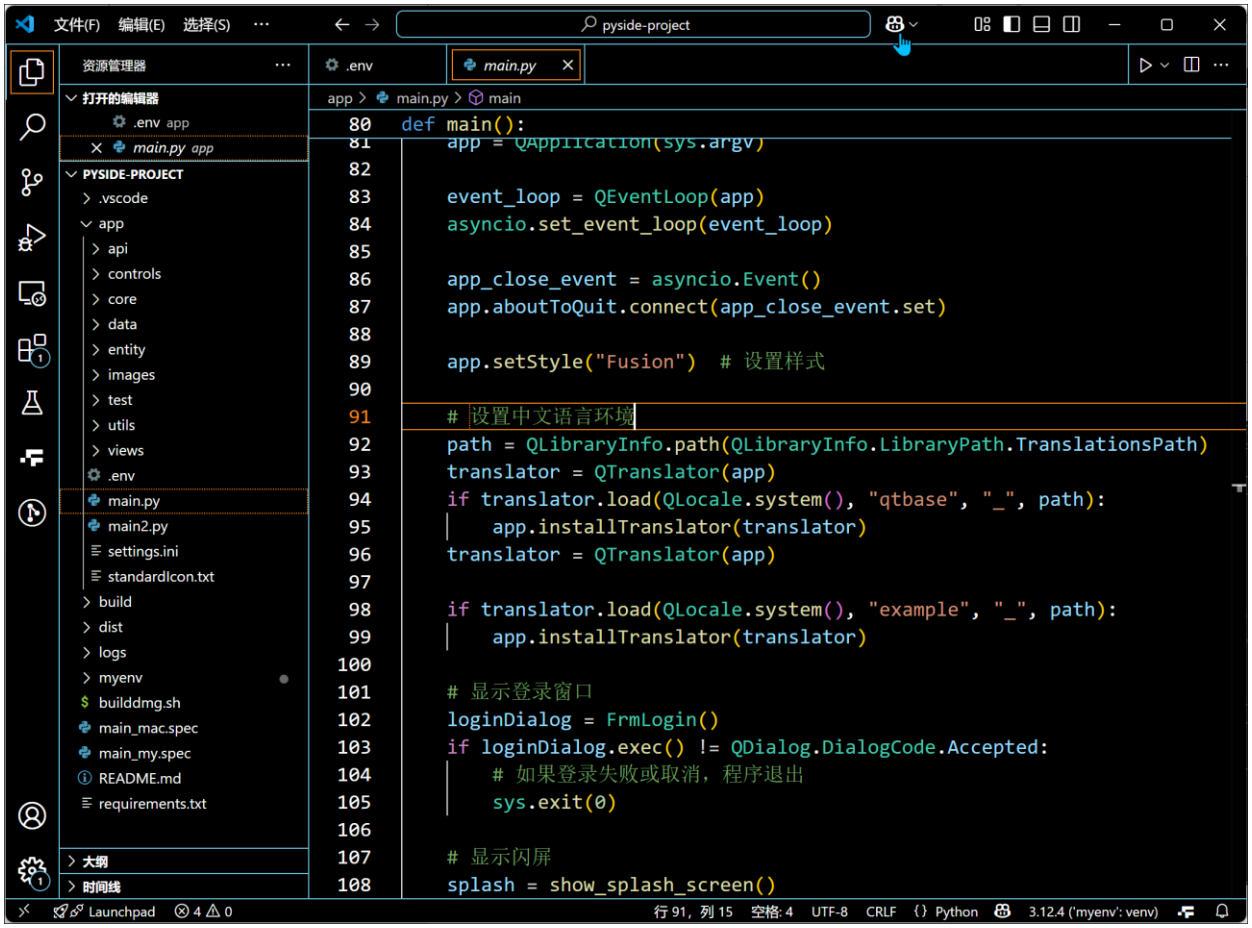
编号	名称	说明
MD001	框架业务及数据处理基础模块（后端）	提供各个分层的基类封装，包括 WebApi 控制器层、底层数据处理层、ORM 模型实体层、DTO 传输层等基础类的封装。 紧密结合 Database2Sharp 强大代码生成工具生成的代码、各层高度抽象继承及使用泛型支持的 Python 开发框架。 系统所有数据处理模块，底层均能很好支持多种数据库，包

		括 Oracle、SqlServer、SQLite、MySql、Postgresql 等常规数据库，以及扩展其他数据库等，所有模块开发模型及分层思路相同，实现统一化、高效化，减少学习成本。
MD002	API 调用封装层	针对后端的 API 接口，增加对常规操作的 API 调用的基类封装，以及各个模块的 API 类的封装处理。
MD003	框架界面处理基础模块	提供对系统界面的抽象封装的基础模块。包括常规的列表分页展示、编辑/新增对话框、参数配置对话框等界面的封装基类，以及图标选择、通用列表打印预览等界面基础类。 同时还包括对常规控件的二次封装内容，如文本输入、数值输入、下拉类别、树列表等控件，以便增加一些额外的自定义处理功能和统一接口处理。
MD004	公用类库模块	提供日常各种开发操作的辅助类库。如提示对话框封装、文件对话框封装等等。
MD005	权限管理模块	提供对业务系统功能控制以及用户角色管理的系统功能。权限管理模块既相对业务系统独立，又可以与业务系统紧密整合。权限管理模块负责管理用户信息、机构信息、角色管理、功能点管理和分配、菜单信息和分配、字段权限、黑白名单、操作日志和登录日志等内容。负责管理角色和用户关系、角色和菜单关系、角色和功能关系等内容。
MD006	数据字典模块	提供给业务系统常用字典管理功能，提供界面对字典进行维护等功能。字典模块在很多系统是必须的，如人员的职称、身份、客户类型、活动分类等等，有了这些维护处理，界面处理只需一行代码就可以绑定，非常方便。
MD007	附件管理	该模块其实是很通用的一个模块，例如我们的一些日常记录，可能会伴随着有图片、文档等的附件管理。
MD008	业务编码规则	业务编码规则是一个通用的编码规则设置，方便在系统相关业务表单中引用，生成指定格式的编码，并可以进行递增管理。
MD009	系统参数维护	提供对系统参数维护管理功能，以及对参数设置界面的模块

	和配置管理	化管理，界面按照不同的功能模块或参数类别划分为多个部分，常见形式包括选项卡。

4.2. 项目目录规划

最终项目结构如下所示：



主要的目录结构说明如下所示。

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。

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。

5. 代码生成工具的使用

5.1. 代码工具介绍

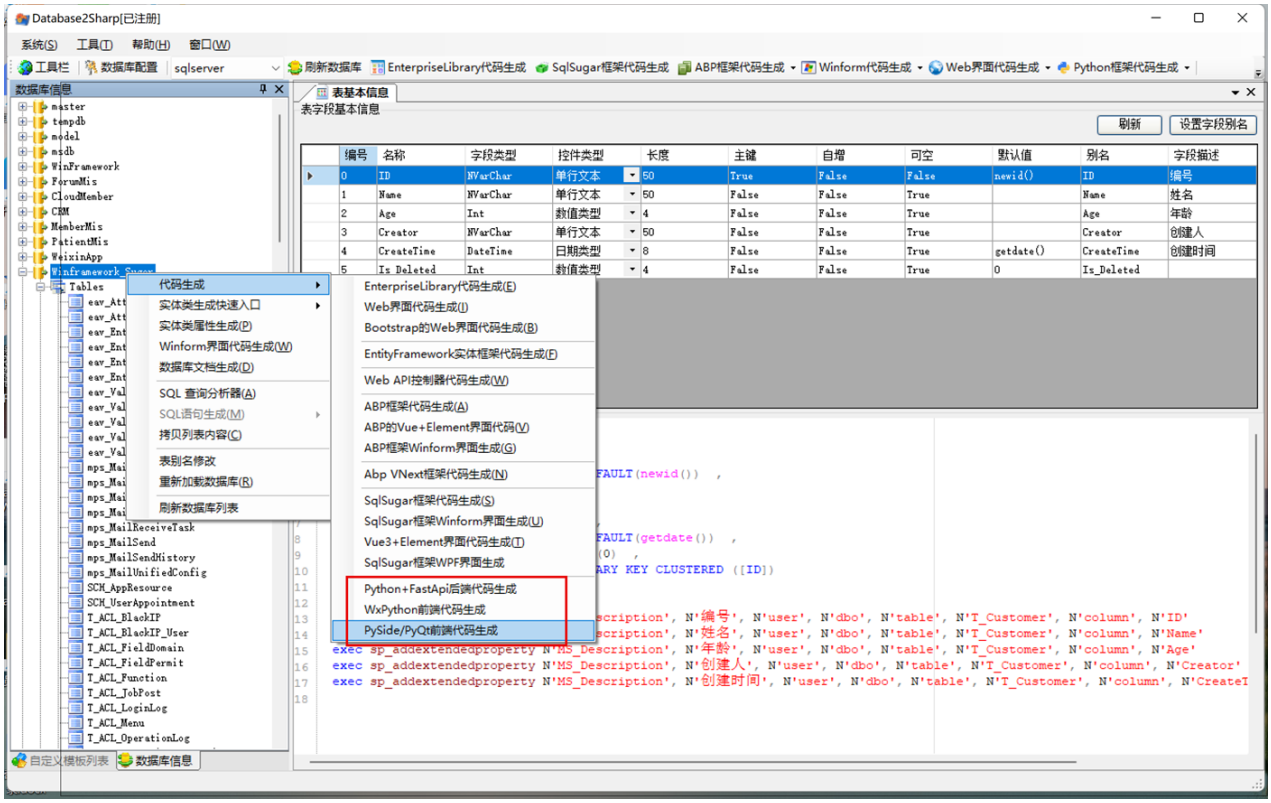
Database2Sharp 是一款最初主要用于 C#代码生成以及数据库文档生成的工具，软件支持 Oracle、SqlServer、MySql、PostgreSQL、Sqlite 以及国产达梦等数据库的代码生成。随着我们框架的扩展，从最初的.NET 体系也扩展到了纯前端的 Vue 项目，以及跨平台的 **Python 开发框架**等。

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。

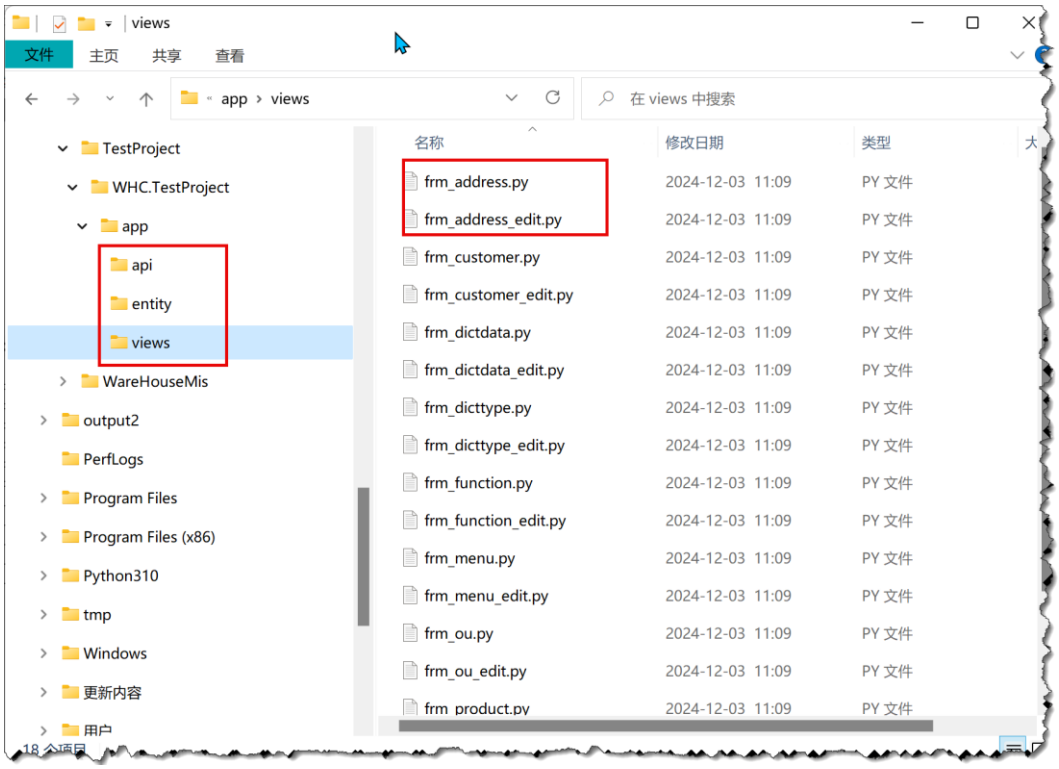
由于我们设计的开发框架，一般都是和支持多种数据库使用的，因此生成的框架代码也就支持多种数据库，一种适应性非常强、弹性很好的应用框架。

5.2. 使用代码工具生成框架代码

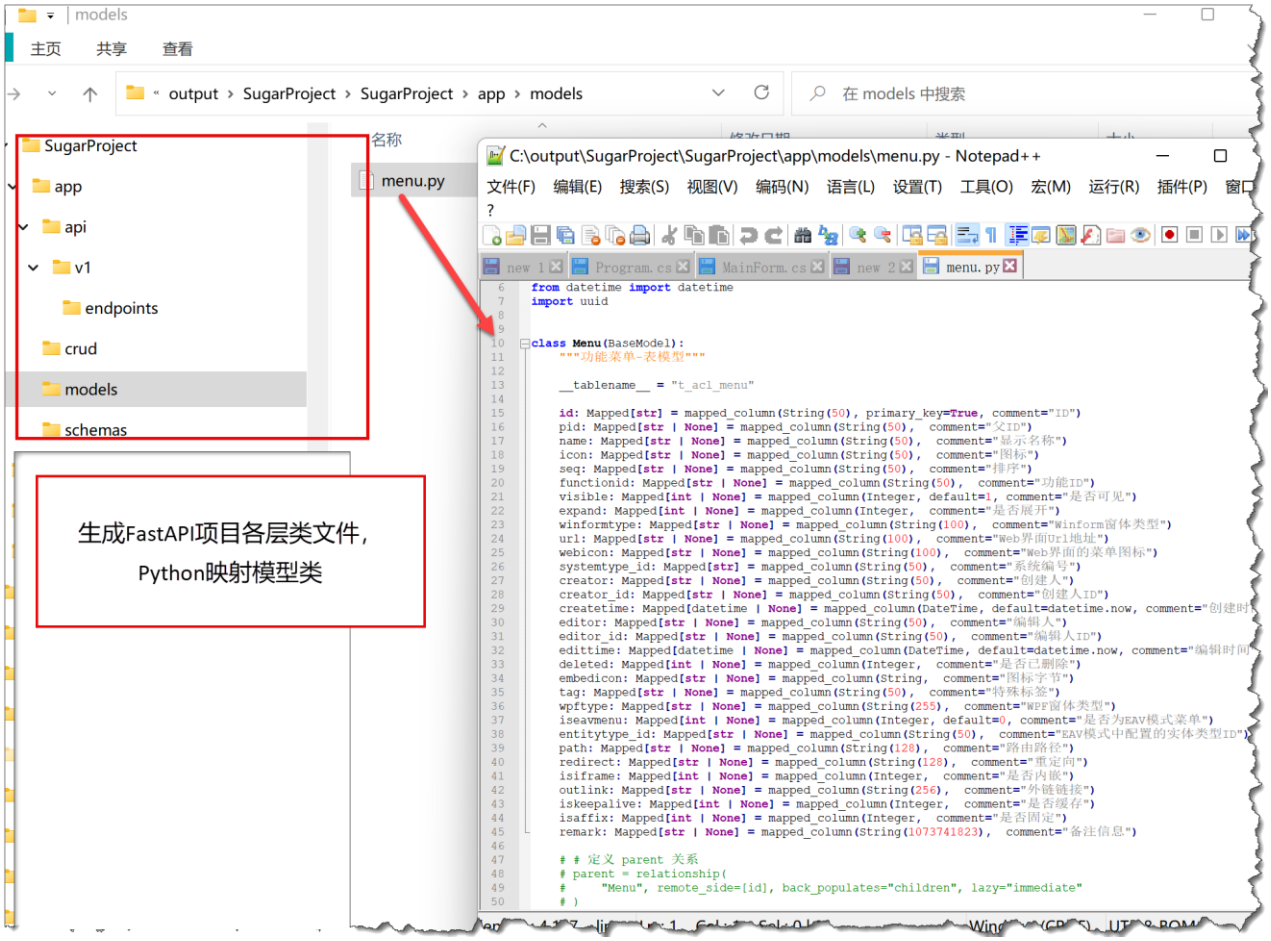
在展开数据库信息后，可以再代码生成工具的主工具栏或者右键菜单上执行 WxPython 前端代码和 PySide/PyQt 前端代码生成的操作，以及基于 FastApi 的后端 Web API 项目，如下界面所示。



选择相关的数据表后，一键生成前端相关的代码，如下所示。



后端 FastApi 项目代码生成相关的模型类、Crud 类和 endpoits 的路由处理类等内容。



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。

5.3. 使用代码生成工具生成数据库设计文档

除了生成较高质量的 C#项目代码外，工具还提供了一个非常不错的数据库文档的生成，方便数据库定型后，统一生成数据库设计文档，然后再在此基础上做一些修饰就非常不错的了。

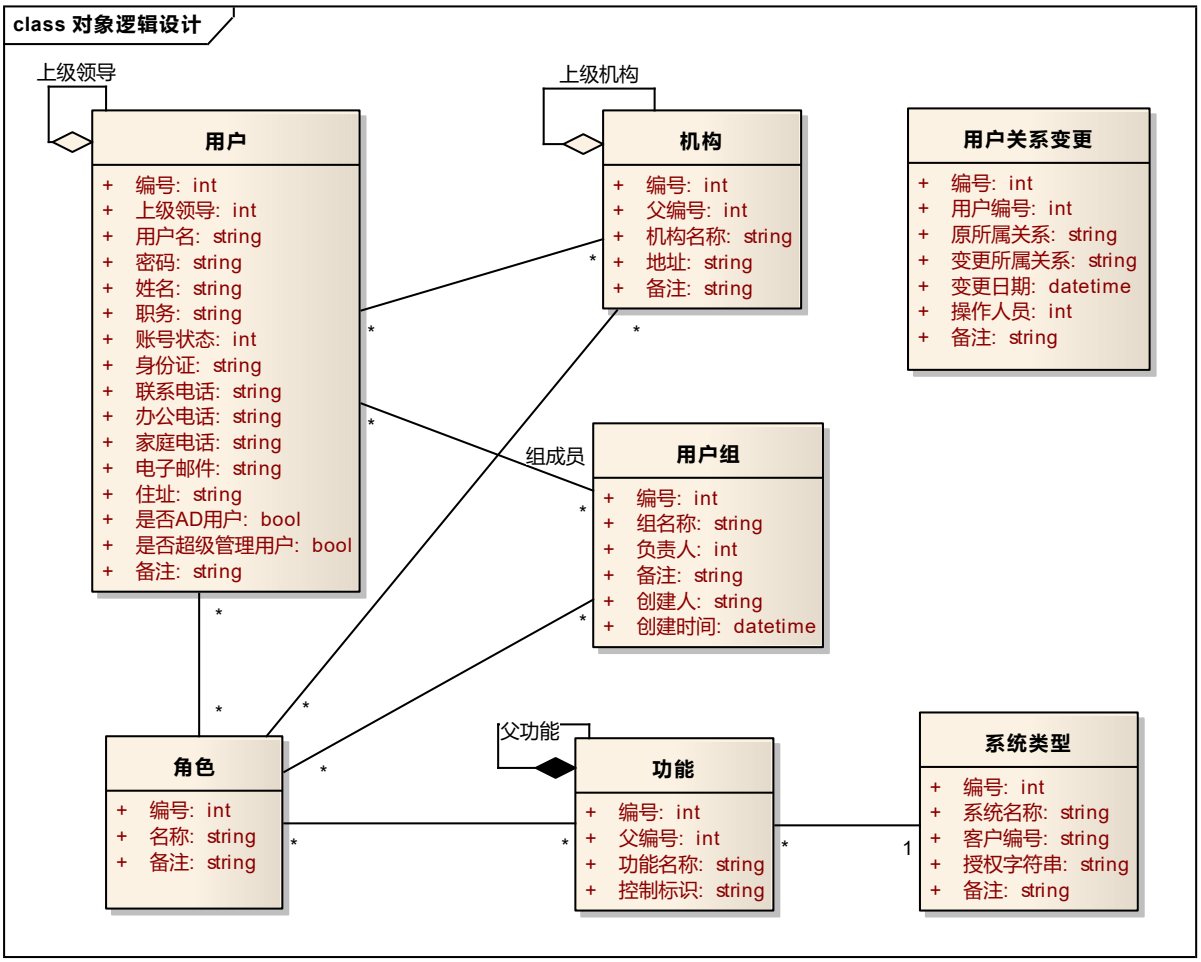
6. 数据库设计

6.1. 框架数据库设计

由于内容比较多，请参考另外的独立数据库设计文档《**开发框架_数据库设计说明书**》。

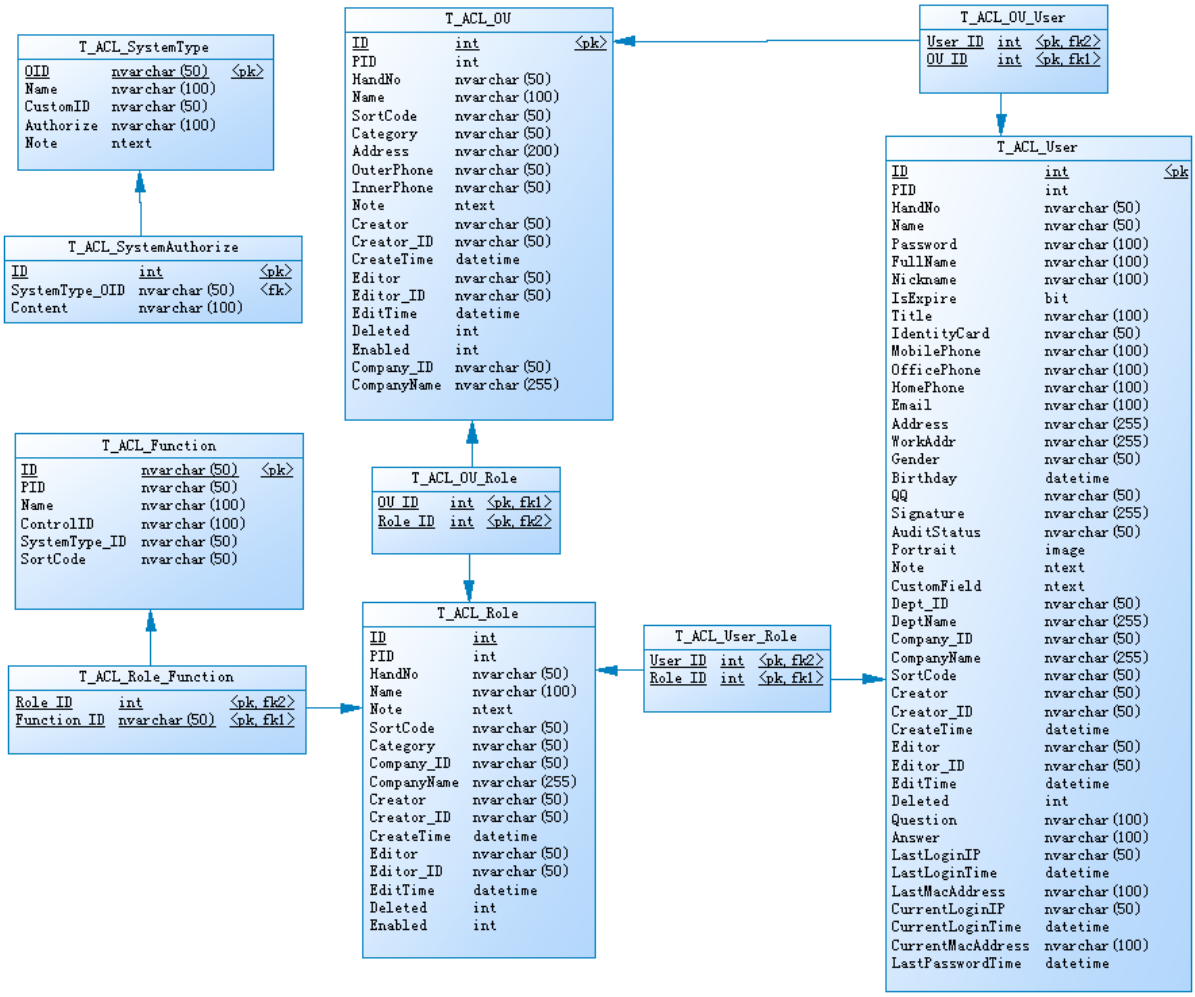
6.2. 权限数据库设计

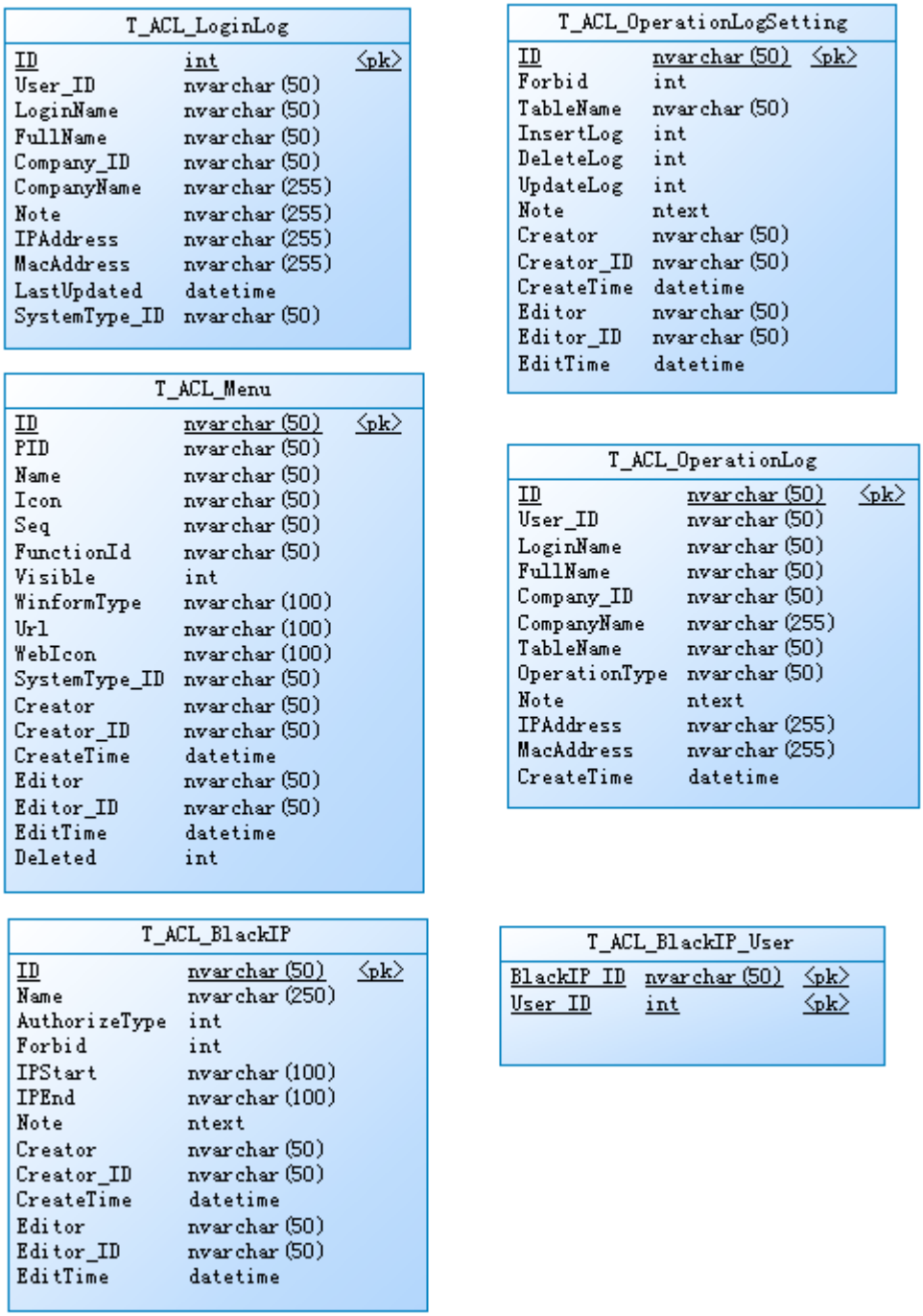
6.2.1. 逻辑设计



6.2.2. 物理设计

数据库分为两部分：一部分是权限系统模块的核心关系表，包括用户、角色、机构、功能、系统类型以及它们之间的中间表等关系；另一部分是权限系统的拓展内容，包括菜单管理、登陆日志、操作日志、黑白名单表等内容。


T_ACL_SystemType
OID nvarchar(50) <pk>
Name nvarchar(100)
CustomID nvarchar(50)
Authorize nvarchar(100)
Note ntext
T_ACL_SystemAuthorize
ID int <pk>
SystemType_OID nvarchar(50) <fk>
Content nvarchar(100)
T_ACL_OU
ID int <pk>
PID int
HandNo nvarchar(50)
Name nvarchar(100)
SortCode nvarchar(50)
Category nvarchar(50)
Address nvarchar(200)
OuterPhone nvarchar(50)
InnerPhone nvarchar(50)
Note ntext
Creator nvarchar(50)
Creator_ID nvarchar(50)
CreateTime datetime
Editor nvarchar(50)
Editor_ID nvarchar(50)
EditTime datetime
Deleted int
Enabled int
Company_ID nvarchar(50)
CompanyName nvarchar(255)
T_ACL_OU_User
User_ID int <pk, fk2>
OU_ID int <pk, fk1>
T_ACL_User
ID int <pk
PID int
HandNo nvarchar(50)
Name nvarchar(50)
Password nvarchar(100)
FullName nvarchar(100)
Nickname nvarchar(100)
IsExpire bit
Title nvarchar(100)
IdentityCard nvarchar(50)
MobilePhone nvarchar(100)
OfficePhone nvarchar(100)
HomePhone nvarchar(100)
Email nvarchar(100)
Address nvarchar(255)
WorkAddr nvarchar(255)
Gender nvarchar(50)
Birthday datetime
QQ nvarchar(50)
Signature nvarchar(255)
AuditStatus nvarchar(50)
Portrait image
Note ntext
CustomField ntext
Dept_ID nvarchar(50)
DeptName nvarchar(255)
Company_ID nvarchar(50)
CompanyName nvarchar(255)
SortCode nvarchar(50)
Creator nvarchar(50)
Creator_ID nvarchar(50)
CreateTime datetime
Editor nvarchar(50)
Editor_ID nvarchar(50)
EditTime datetime
Deleted int
Question nvarchar(100)
Answer nvarchar(100)
LastLoginIP nvarchar(50)
LastLoginTime datetime
LastMacAddress nvarchar(100)
CurrentLoginIP nvarchar(50)
CurrentLoginTime datetime
CurrentMacAddress nvarchar(100)
LastPasswordTime datetime
T_ACL_Function
ID nvarchar(50) <pk>
PID nvarchar(50)
Name nvarchar(100)
ControlID nvarchar(100)
SystemType_ID nvarchar(50)
SortCode nvarchar(50)
T_ACL_OU_Role
OU_ID int <pk, fk1>
Role_ID int <pk, fk2>
T_ACL_Role
ID int
PID int
HandNo nvarchar(50)
Name nvarchar(100)
Note ntext
SortCode nvarchar(50)
Category nvarchar(50)
Company_ID nvarchar(50)
CompanyName nvarchar(255)
Creator nvarchar(50)
Creator_ID nvarchar(50)
CreateTime datetime
Editor nvarchar(50)
Editor_ID nvarchar(50)
EditTime datetime
Deleted int
Enabled int
T_ACL_User_Role
User_ID int <pk, fk2>
Role_ID int <pk, fk1>
T_ACL_Role_Function
Role_ID int <pk, fk2>
Function_ID nvarchar(50) <pk, fk1>



6.3. 字典数据库设计

字典数据库部分只包含两个表，一个是字典数据类型表、一个是字典信息表，字典类型表包括一个可以无限递归的树形结构，可以嵌套很多类型，只是一般字典模块不需要太多的层次。字典数据信息表，则是实际的数据字典内容，有一个键值指向具体的所属类型。

详细的表结构如下所示。

1） 字典数据信息表：TB_DictData

字段列表						
编号	字段列名	字段描述	数据类型	可空	默认值	约束类型
1	ID	编号	NVarChar(50)			主键
2	DictType_ID	字典大类	NVarChar(50)	√		
3	Name	字典名称	NVarChar(50)	√		
4	Value	字典值	NVarChar(50)	√		
5	Remark	备注	NVarChar(255)	√		
6	Seq	排序	NVarChar(50)	√		
7	Editor	编辑者	NVarChar(50)	√		
8	LastUpdated	编辑时间	DateTime(8)	√	getdate()	

2） 字典数据类型表：TB_DictType

字段列表						
编号	字段列名	字段描述	数据类型	可空	默认值	约束类型
1	ID		NVarChar(50)			主键
2	Name	类型名称	NVarChar(50)			
3	Code	类型代码	NVarChar(50)	√		
4	Remark	备注	NVarChar(255)	√		
5	Seq	排序	NVarChar(50)	√		
6	Editor	编辑者	NVarChar(50)	√		
7	LastUpdated	编辑时间	DateTime(8)	√	getdate()	
8	PID	分类	NVarChar(50)	√		

6.4. 其他通用模块

TB_UserParameter(用户参数配置)

字段列表						
编号	字段列名	字段描述	数据类型	可空	默认值	约束类型
1	ID		NVarChar(50)			主键
2	Name	类型名称	NVarChar(255)	√		
3	Content	参数文本内容	NText	√		
4	Creator	创建人	NVarChar(50)	√		
5	CreateTime	创建时间	DateTime	√	getdate()	

TB_FileUpload(上传附件信息)

字段列表						
编号	字段列名	字段描述	数据类型	可空	默认值	约束类型

1	ID		NVarChar(50)			主键
2	Owner_ID	附件组所属记录 ID	NVarChar(50)	√		
3	AttachmentGUID	附件组 GUID	NVarChar(50)	√		
4	FileName	文件名	NVarChar(255)	√		
5	BasePath	基础路径	NVarChar(255)	√		
6	SavePath	文件保存相对路径	NVarChar(255)	√		
7	Category	文件分类	NVarChar(255)	√		
8	FileSize	文件大小	Int	√		
9	FileExtend	文件扩展名	NVarChar(10)	√		
10	Editor	所属用户	NVarChar(50)	√		
11	AddTime	添加时间	DateTime	√		
12	DeleteFlag	删除标志，1 为删除，0 为正常	Int	√	0	