S70 框架说明

目录

1. 开发环境介绍

后端采用 NetCore 3.1；

数据库支持 MySql 和 SqlServer，建议使用 MySql；

缓存支持本地缓存与 Redis 共享缓存；

前端框架 Vue 3，ElementPlus UI 组件库；

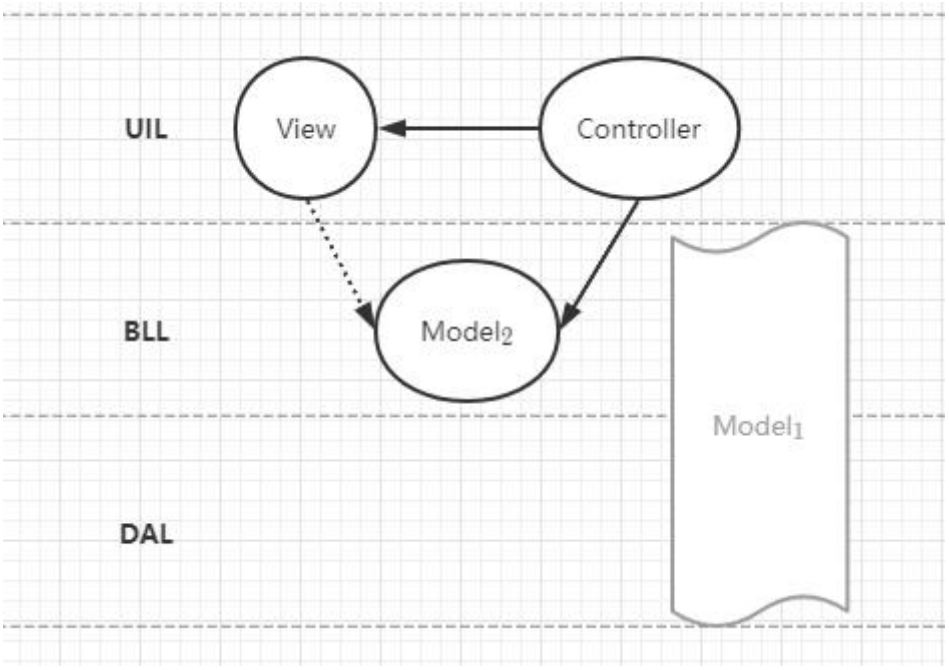
开发工具推荐系统开发使用 VS2019，前端组件开发使用 VS Code；

部署环境支持 Windows，Linux，Docker。

2. 系统架构说明

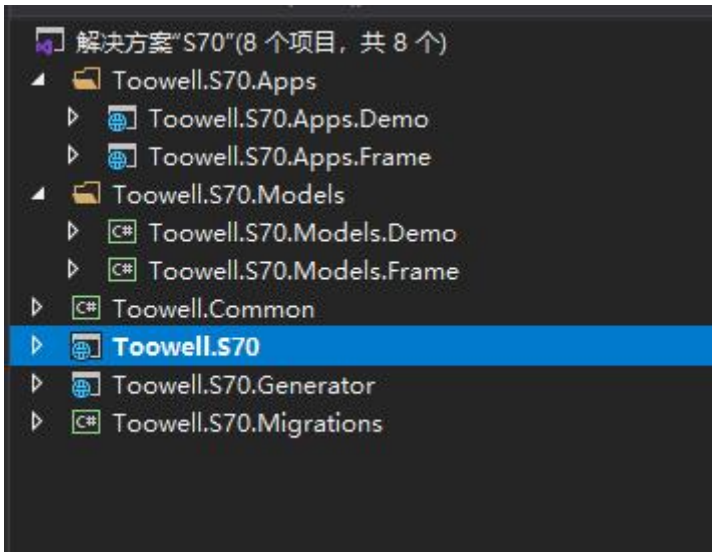
S70 软件架构属于三层架构，即表现层（UIL）、业务逻辑层（BLL）、数据访问层（DAL），三层间采用 $Model_{(1)}$ 实体对象传递参数，$Model_{(1)}$ 对于三层架构只是锦上添花的存在，并非必须，例如 user_bll.add(user)和 user_bll.add(name,age,sex,....)是等效的；

S70 的 UIL 表现层采用 MVC($Model_{(2)}$ - View - Controller)设计模式，$Model_{(2)}$ 为业务模型，其包含数据实体和业务逻辑，$Model_{(2)}$ 与 BLL 重叠，表现层基本思路是控制器 Controller 通过业务模型 $Model_{(2)}$（即与 BLL 对话，更新和获得数据通常是 $Model_{(1)}$ 并封装成 ViewModel），实现视图 View 和业务模型 $Model_{(2)}$ 协同工作。

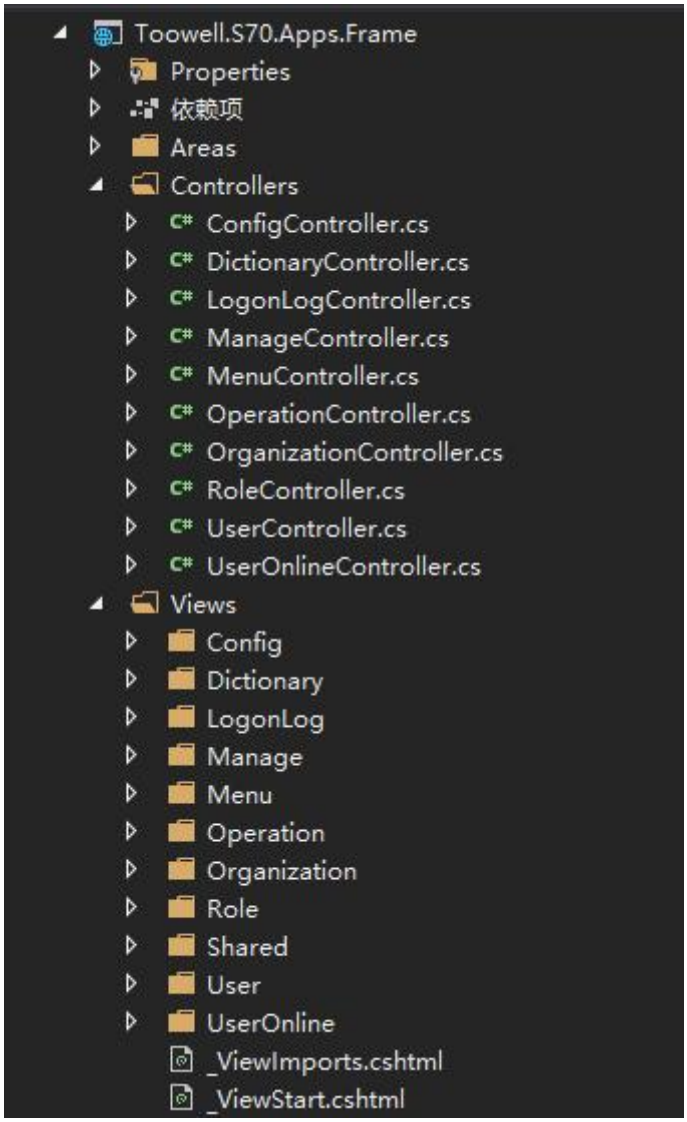


3. 解决方案结构介绍

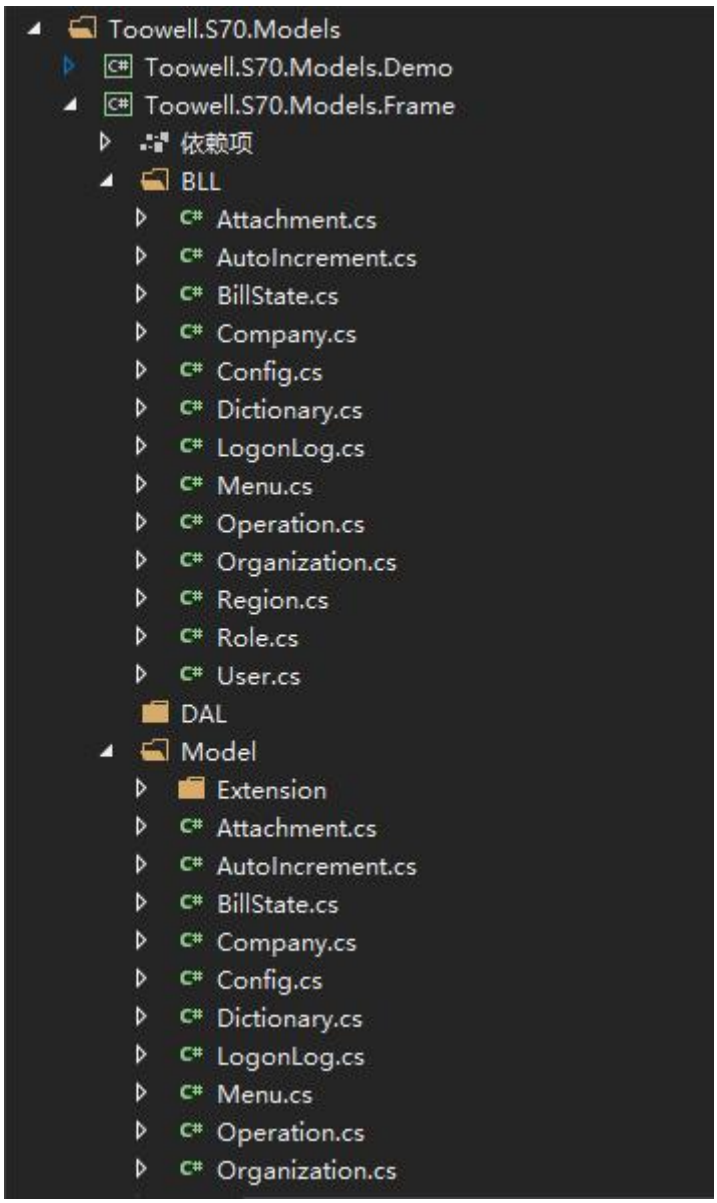
项目命名规则，后端项目采用单词首字母大写，前端项目采用全小写字母命名；顶级命名空间 Toowell，Toowell.S70 为启动项目。



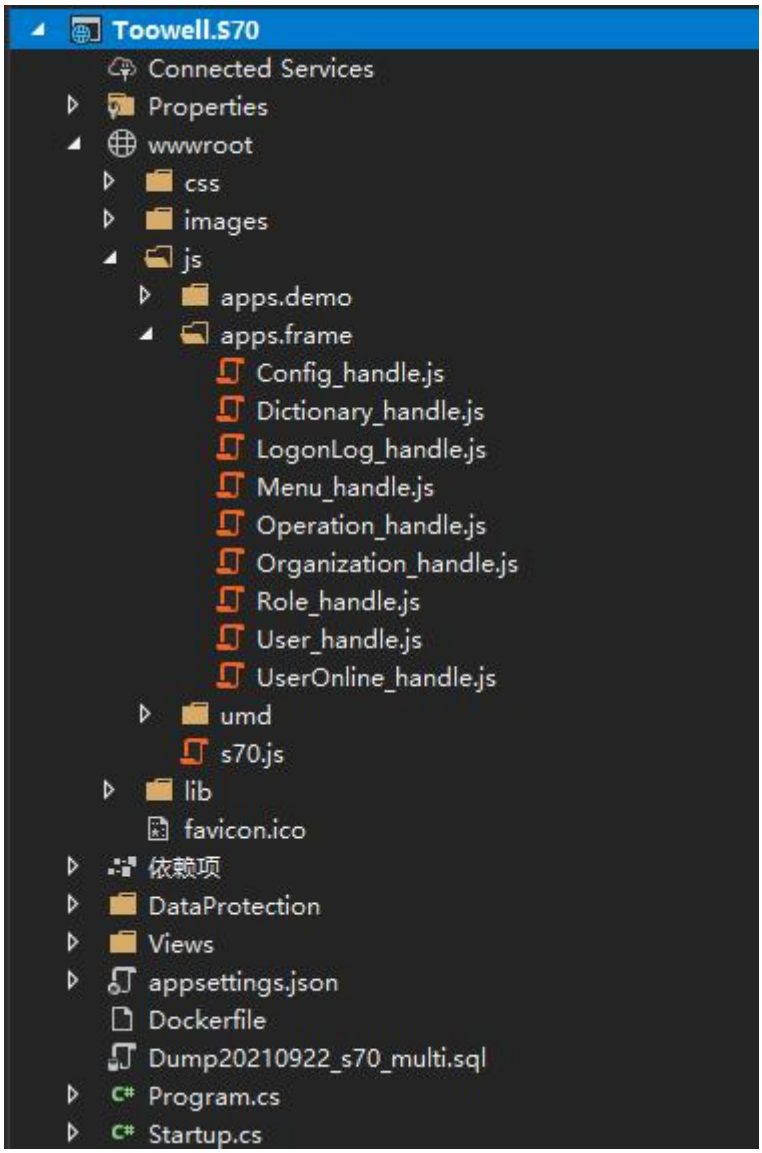
Toowell.S70.Apps.XXX 按应用划分项目，例如：基础后台项目 Toowell.S70.Apps.Frame，项目包含 MVC 的 Controller 和 View，主要功能有系统布局框架，登陆校验，授权控制，组织架构，用户，角色，权限，字典，配置，菜单等管理模块；



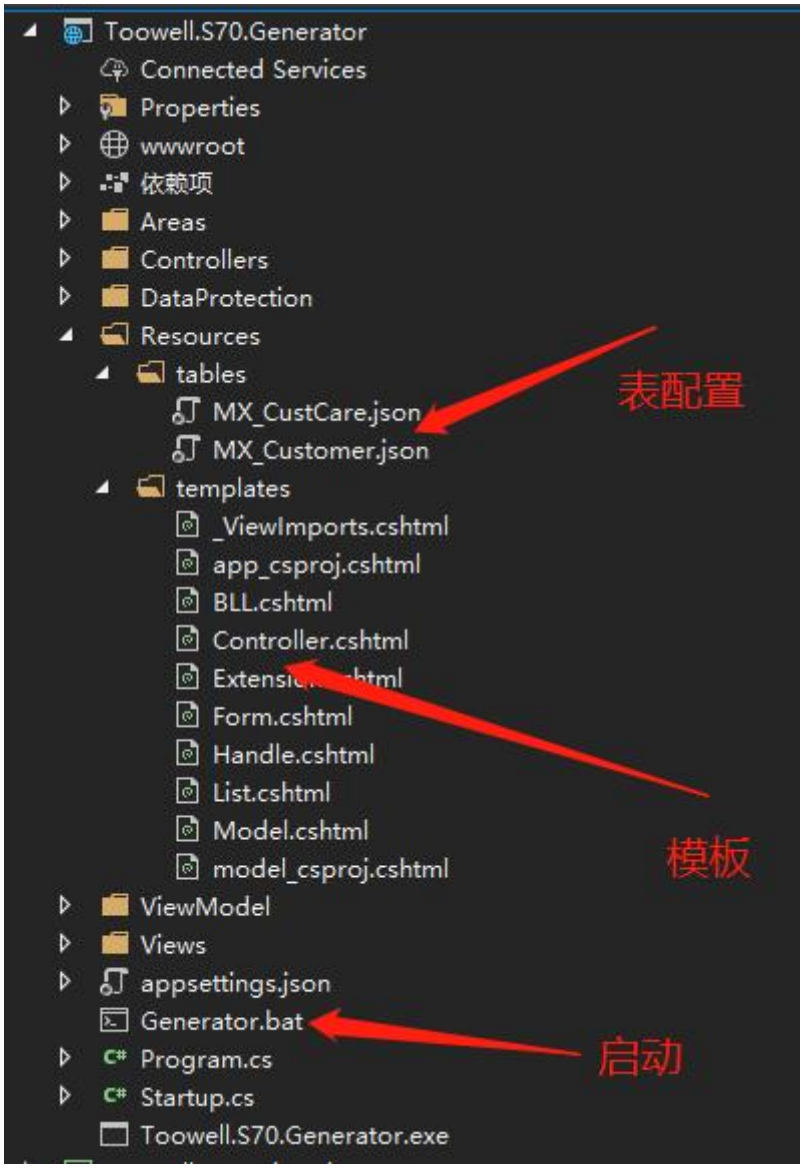
其对应业务模型层划分在 Toowell.S70.Models.Frame 项目中，其包含上述模块的 BLL 层，DAL 层，Model 实体类库；



对应的前端交互脚本在 Toowell.S70/wwwroot/js/apps.frame 目录中，JS 脚本采用 Vue 前端框架。



Toowell.S70.Generator 为 S70 框架定制的代码生成器项目，其生成器前端项目源码为 toowell.s70.designer

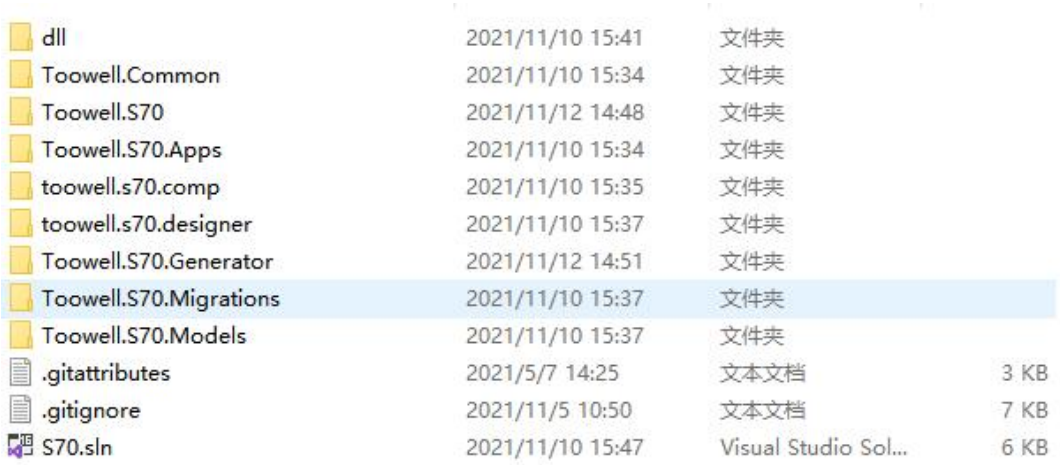


Toowell.S70.Migrations 项目是利用 EF Code First 做数据迁移，主要作用是生成数据表。

```
namespace Toowell.S70.Migrations
{
    /// <summary>
    /// 安装命令工具 dotnet tool install --global dotnet-ef
    /// 迁移命令：
    /// 添加迁移 dotnet ef migrations add demo_1 --context DemoDbContext
    /// 删除迁移 dotnet ef migrations remove
    /// 迁移生效 dotnet ef database update  --context DemoDbContext
    /// 迁移回滚 dotnet ef database update demo_1  --context DemoDbContext
    /// </summary>
    5 个引用
```

toowell.s70.comp 是 S70 框架前端基础组件项目，采用 Vue3+ElementPlus 开发，其主要包含用户，部门，角色，字典，配置，区域等常用的下拉，多选，单选等组件，组件编译打包文件为 Toowel.S70 项目下 wwwroot/js/umd 目录，详细文档参考 toowell.s70.comp 目录下 DOC.pdf 文件。

整个解决方案目录结构



4. 前端组件开发说明

前端组件项目 toowell.s70.comp 采用 Vue3+ElementPlus 开发；

Vue3 环境安装参考 vue 官网安装说明 https://v3.cn.vuejs.org/guide/installation.html

ElementPlus 官网地址 https://element-plus.gitee.io/zh-CN/guide/quickstart.html

简单说明下编译命令，在项目 toowell.s70.comp 目录下执行

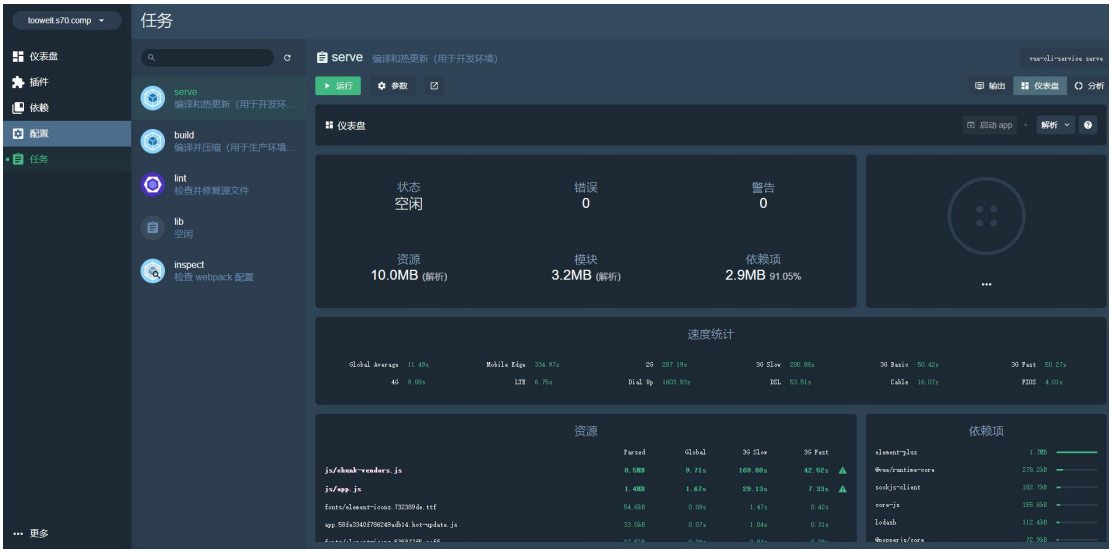
npm install　　#安装依赖

vue ui　　#启动 Vue 项目管理器

npm run serve　　#启动测试 app

npm run lib　　#编译打包组件，输出目录已指定在 Toowell.S70/wwwroot/js/umd/

也可以通过 GUI 操作，界面大致如下：



5. 应用开发说明

本节介绍利用 S70 框架快速开发应用系统流程，并介绍两种开发集成模式。

先以简单的信息管理模块为例，其特点是功能简单但各模块间关联性较多，这类应用开发首先根据需求确定业务模型实体（Model），设计表结构，实现数据访问层（DAL）和业务逻辑层（BLL），然后实现控制器（Controller）和视图（View），交互脚本（JS），这种简单模块的代码重复性高，结构简单，适合前后端混合开发模式，在开发过程中为了提高开发效率并专注于解决业务逻辑和交互体验，一般采用代码生成器来生成模块功能的基础代码，在生成的代码基础上再进行增进功能和优化。

另外一种较为复杂的应用，特点是功能较为独立，用户交互强，可移植性要求强，例如，流程设计器，事件日历， 利率计算器等应用，这类应用比较适合用前后端分离模式来开发，前端以组件的形式发布，后端以独立 app.dll 来发布，可以移植或嵌入到不同项目中。

1. 前后端混合开发模式

在 S70 框架中，Toowell.S70.Apps.Frame 项目就属于这种开发模式。

2. 前后端分离开发模式

toowell.s70.comp 和 toowell.s70.designer 都是采用前后端分离模式开发的，并打包成组件，支持其他组件引用或嵌入到其他页面。

6. 单点登录说明

S70 框架对单点登录支持三种模式

1. 用户密码签名模式

此模式需同步系统间用户名和密码，使用用户名和密码等生成 token 信息来交换认证，token 生成规则：

token=base64(UserName=xx&TimeStamp=xxx&Sign=md5(username+timestamp+password))

示例：/user/list?mode=0&token=xxxxxxxxxx

示例：User.SSO_URL(“/user/list?{token}”)

2. 秘钥签名模式

此模式需同步系统间用户名及设置指定共享密钥，token 生成规则：

token=base64(UserName=xx&TimeStamp=xxx&Sign=md5(username+timestamp+SecretKey))

示例：/user/list?mode=1&token=xxxxxxxxxx

示例：User.SSO_URL(“/user/list?{token(1,SecretId)}”)

3. 简单秘钥模式

此模式需同步系统间用户名及设置指定共享密钥，token 直接使用指定共享密钥

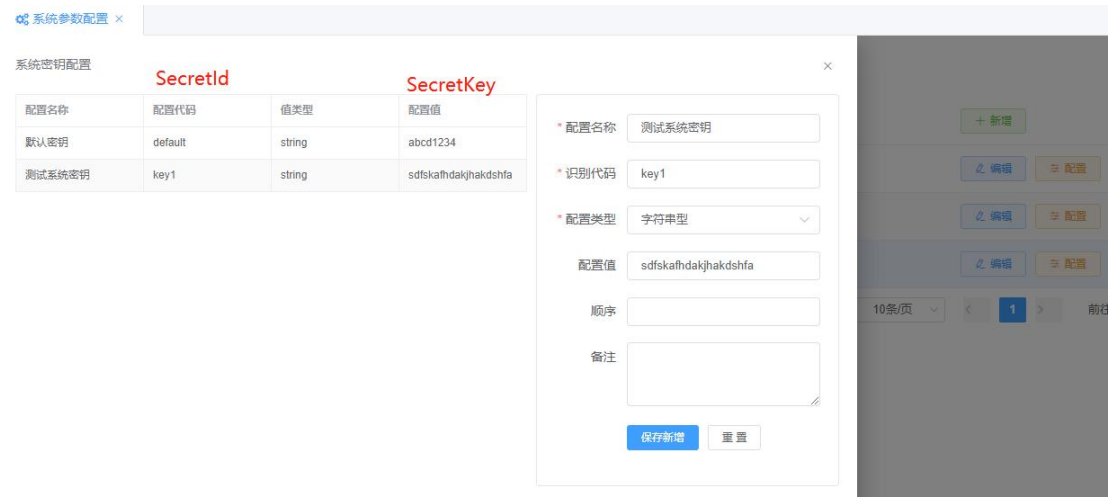
token = SecretKey

生成:/user/list?mode=2&username=admin&secretid=xxxx&token=xxxxxxxxxx

示例：User.SSO_URL(“/user/list?{token(2,SecretId)}”)

4. 密钥设置与管理

增加系统配置项 “系统密钥配置”



5. 菜单使用示例

```
0 个引用 | 0 项更改 | 0 名作者，0 项更改
public JsonResult GetLeftMenuData(int RootID)
{
    string strWhere = string.Format("RootID={0} AND Depth>=2 AND IsHide=
    List<Model.Menu> list = bll.GetModelList(strWhere).Where(m => UserAu
    List<object> result = list.Select(m => new
    {
        ID = m.ID.ToString(),
        ParentID = m.ParentID.ToString(),
        children = new List<object>(),
        m.MenuName,
        MenuUrl = User.SSO_URL(m.MenuUrl),
        m.MenuIcon,
        m.Sequence,
        m.IsLogin,
        m.IsHide,
    }).ToList<object>().GenTreeData();
    return Json(result);
}
```

7. 代码生成器说明

项目代码结构，前端项目 toowell.s70.designer，后端项目 Toowell.S70.Generator，交互接口 Toowell.S70.Generator/Area/APIcomp/GeneratorController。

主要功能是在前端配置表结构，将表结构配置保存在 Resources/tables 目录中以 json 格式存储，读取 Resources/templates 中的模板文件，并利用 RazorLight 模板引擎解析生成工程代码文件至指定目录。生成器使用如下：

1. 编译并运行生成器

运行 Generator.bat 批处理文件，该批处理先执行.netcore 的 publish 命令，发布生成可执行文件 Toowell.S70.Generator.exe，然后移动到根目录并运行该文件。该项目是一个 web 项目，运行后会启动一个控制台程序并打开默认浏览器地址为 http://localhost:3333/，界面如下：

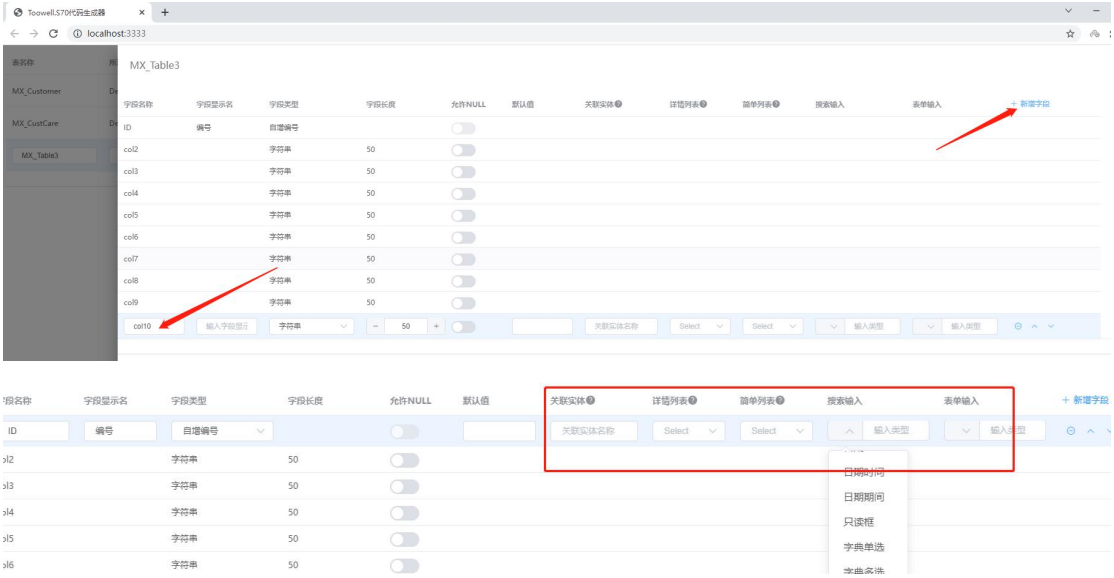
2. 新建表和表结构配置

点击“新建表”，填写表名称，所属应用，描述和实体映射，表名称规则按 MX_ 开头，实体映射则是与表对应的 Model 实体类名称，该会命名影响 BLL，Controller，Js 等命名。

点击“详情”，打开表结构设计页面，点击“新建字段”可添加表字段，字段名区分大小写，字段命名会影响 Model 实体类属性名称，如无法满足复杂表设计要求，可使用数据库管理工具来设计表结构；

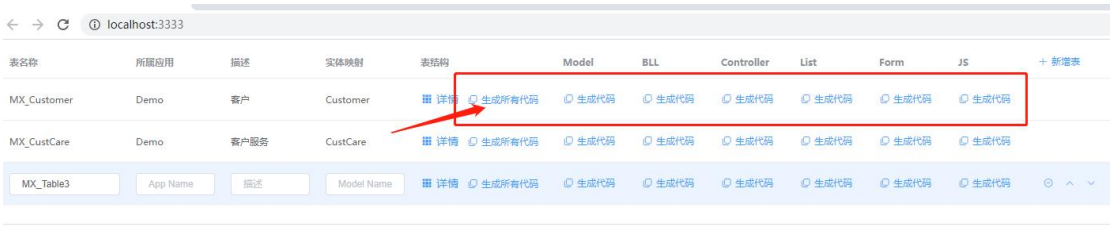
关联实体，详情列表，简单列表，搜索输入，表单输入这些都是配置视图显示和输

入类型，以勾选为主，在代码生成时会按配置来生成代码。

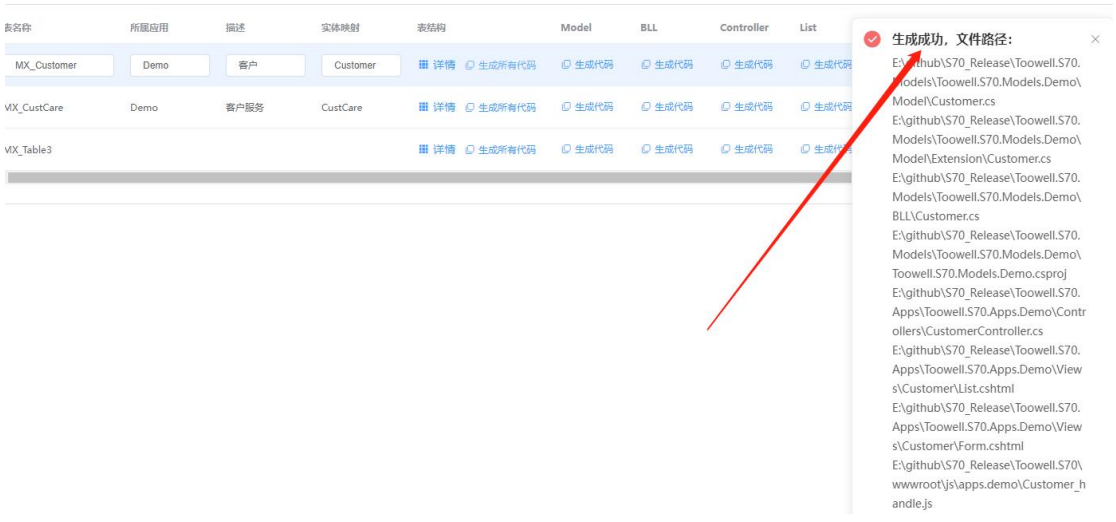


3. 生成代码

完成表配置后直接点击右上角“X”按钮关闭返回，点击列表中的“生成所有代码”，也可以单独生成代码。

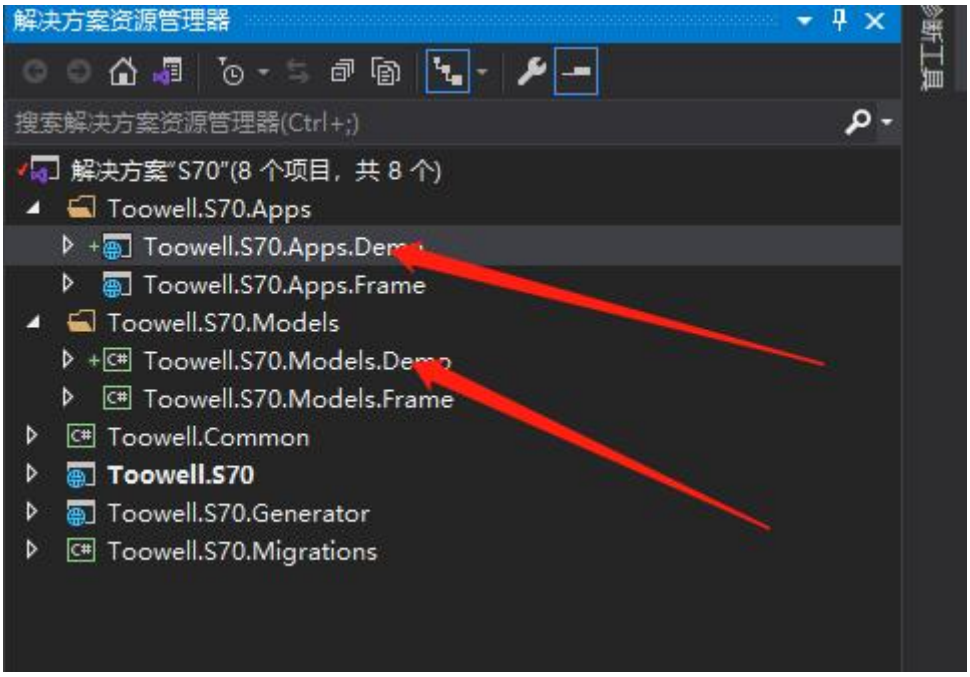


生成代码成功时提示

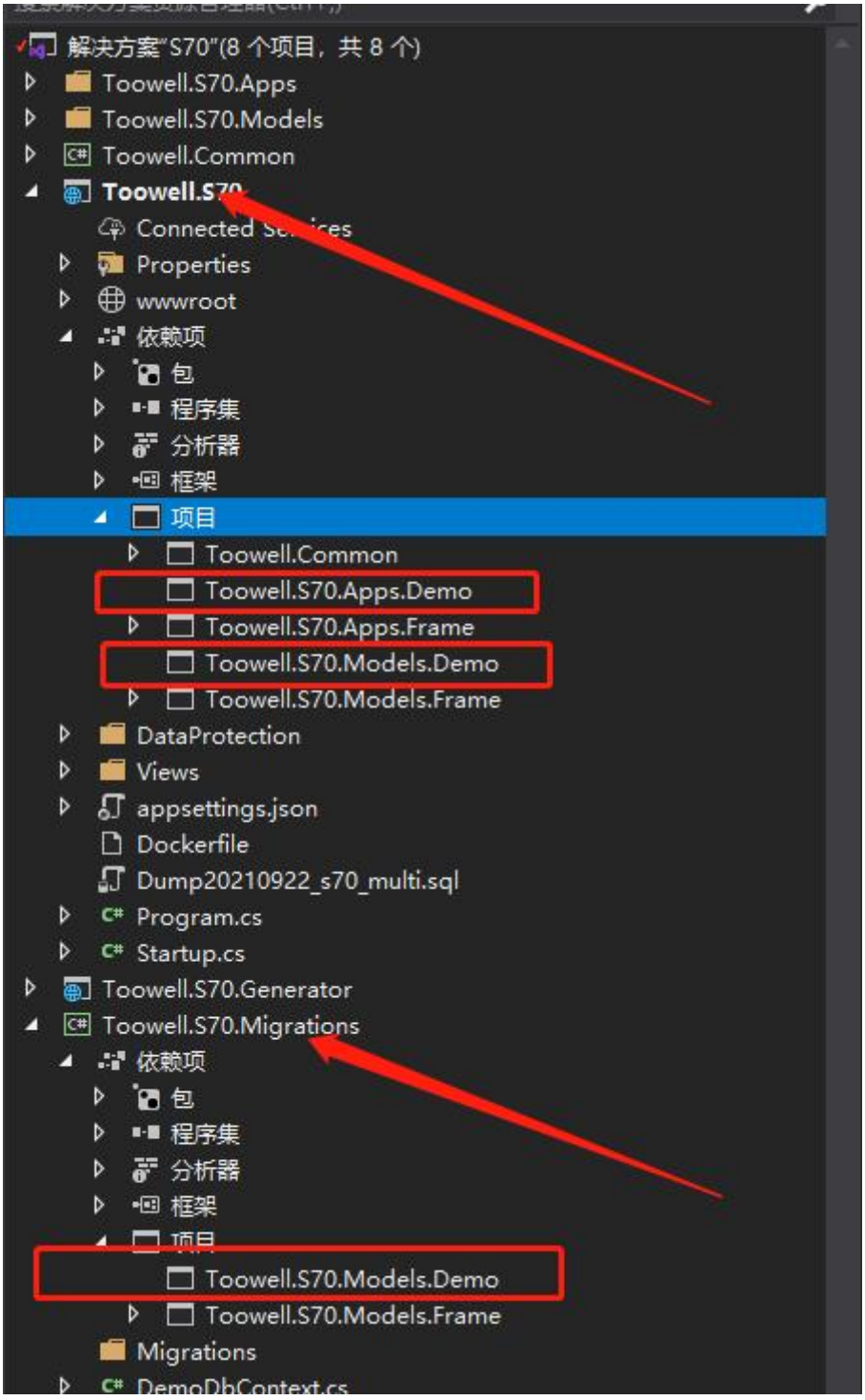


4. 引用项目

将生成好的项目和代码添加至解决方案中，如下图：


解决方案资源管理器
搜索解决方案资源管理器(Ctrl+;)
解决方案"S70"(8 个项目，共 8 个)
Toowell.S70.Apps
Toowell.S70.Apps.Frame
Toowell.S70.Models
Toowell.S70.Models.Frame
Toowell.Common
Toowell.S70
Toowell.S70.Generator
Toowell.S70.Migrations

引用项目，如下图：



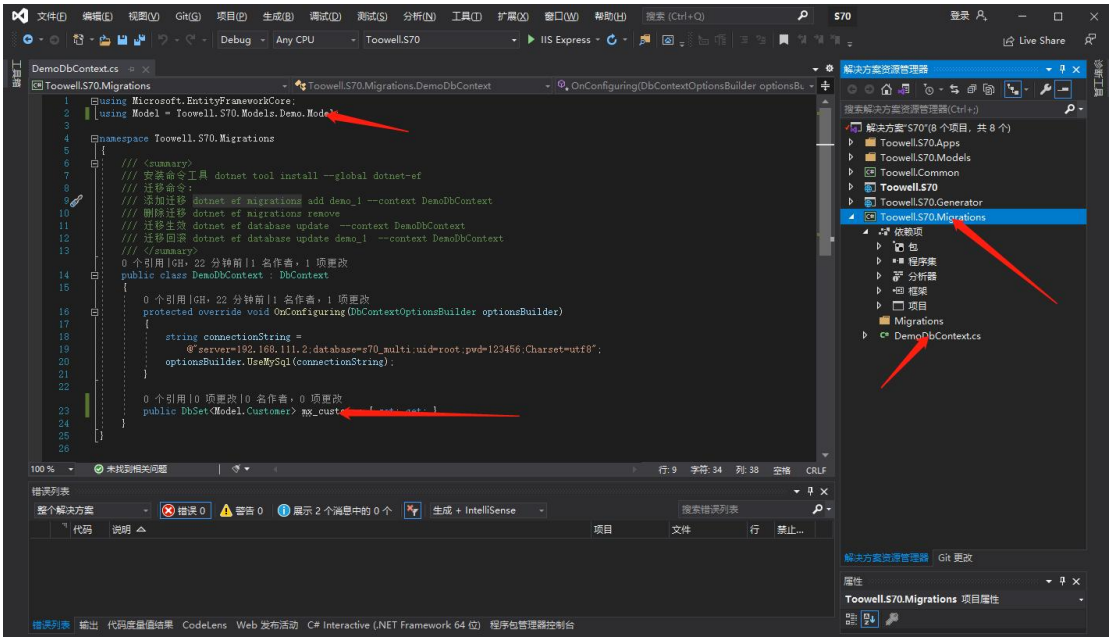
然后编译解决反方案，排查生成代码问题。

8. 数据迁移使用

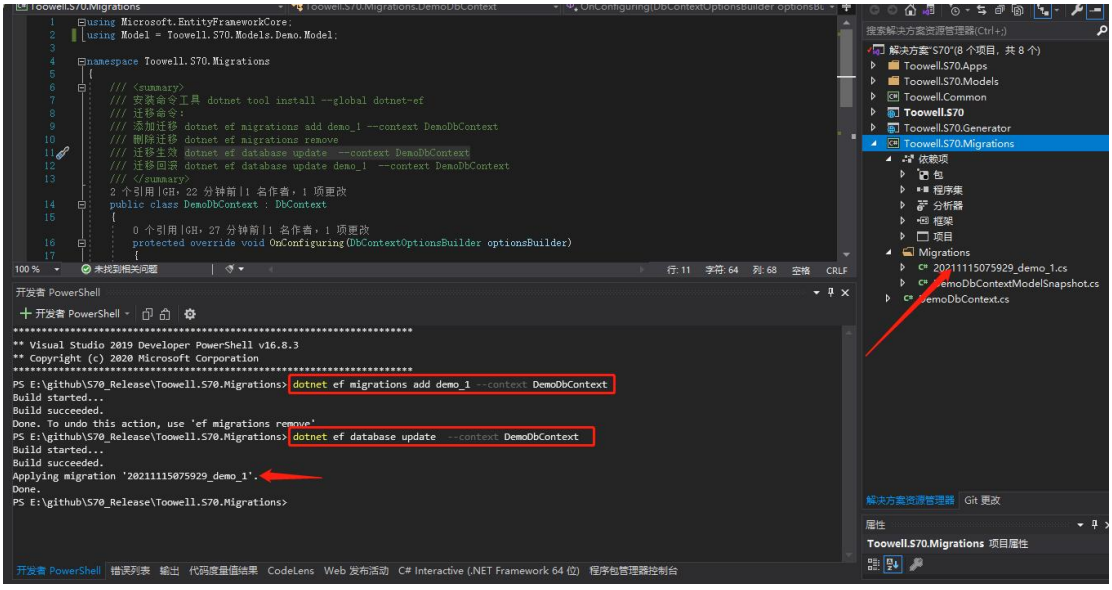
代码生成完成后，我们使用数据迁移工具 dotnet-ef 完成数据库表结构的生成和数据库架构管理，详细使用方法参考微软文档

https://docs.microsoft.com/zh-cn/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli

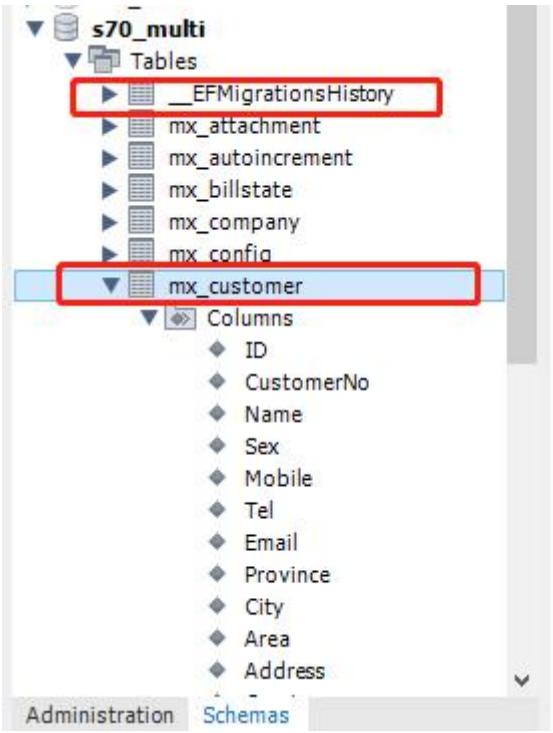
S70 框架使用独立项目 Toowell.S70.Migrations 来完成数据迁移工作，根据下图添加相关代码



完成基本的编码配置后，打开开发者 PowerShell，运行命令“dotnet ef migrations add”和“dotnet ef database update　--context DemoDbContext”

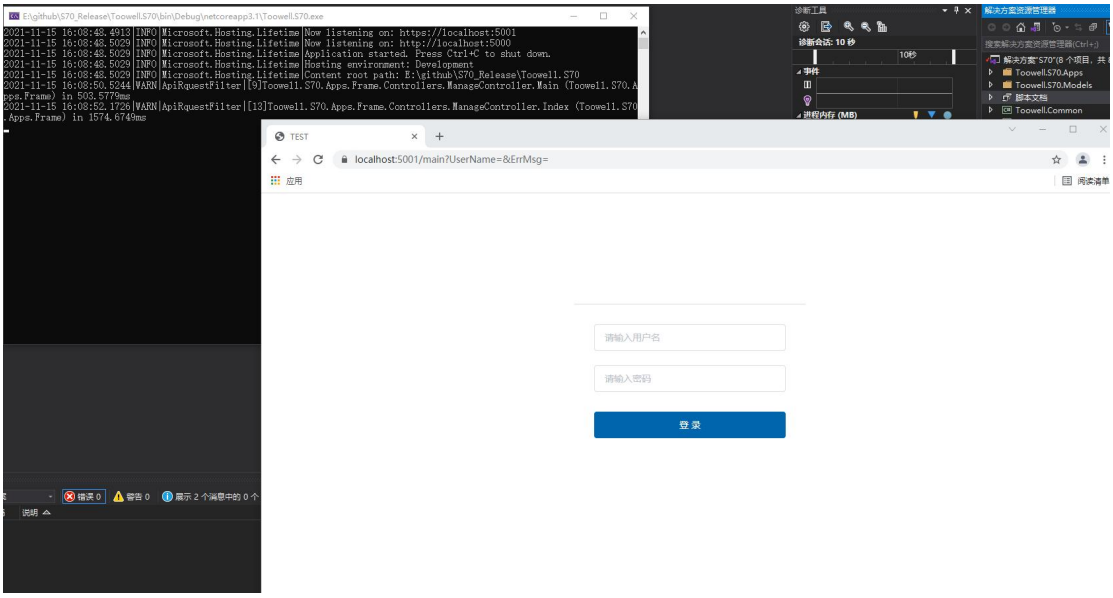


数据库自动生成了 mx_customer 表结构，__EFMigrationsHistory 为迁移记录。

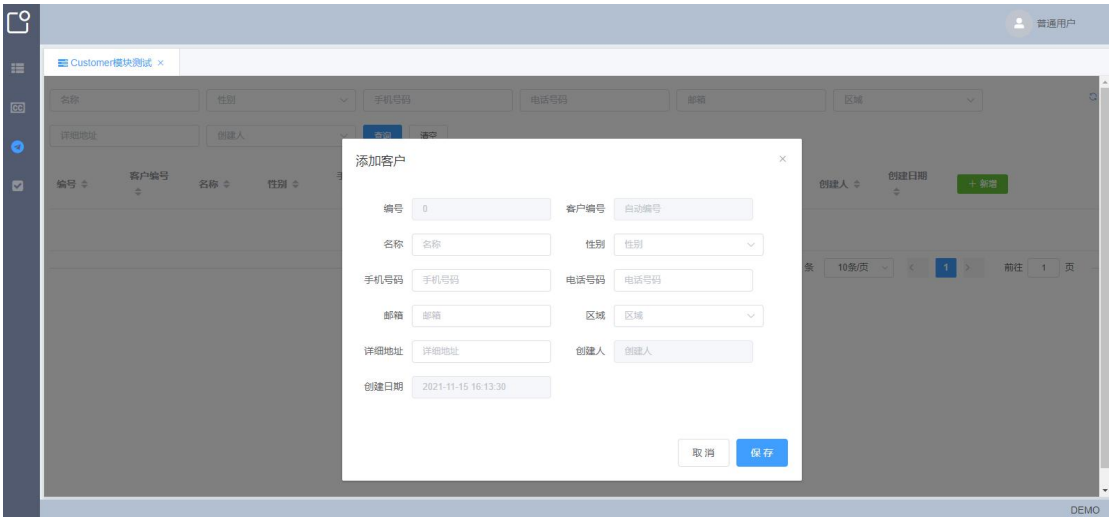


9. **运行与项目部署说明**

完成代码生成和数据迁移后，按 F5 运行调试，Toowell.S70 为启动项目，运行如下：

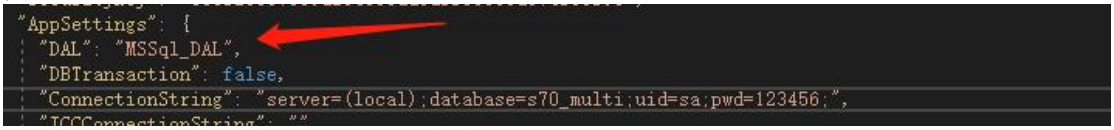


将/Customer/List 添加至菜单中，打开菜单，如下图：



部署环境支持 Windows，Linux，Docker，实践后补充。

部署 SqlServer 配置，修改 appsettings.json 文件 AppSettings 项的”DAL”为”MSSql_DAL”，如下图：



部署 MySql 配置，修改 appsettings.json 文件 AppSettings 项的”DAL”为”MySql_DAL”