如何创建一个WPF模块?

本文主要讲解如何在框架当中, 新建一个WPF模块，主要分为两个部分:

后端

- 创建表模型 并在 EntityFramework Core中添加表定义
- 创建迁移文件以及更新数据库 add-migration / update-database
- 添加当前表对应的模块以及功能权限
- 创建接口以及实体模型
- 实现接口以及数据映射模型配置
- 添加对应功能的多语言翻译

WPF客户端

- 实现接口用于Web请求服务
- 创建视图(View) 以及 视图模型(ViewModel)
- 容器中注入Web服务接口以及 视图(View)
- 添加系统导航的菜单定义
- 添加对应的权限定义 (与后端的定义相同)

演示

1.创建后端服务。

第一步首先在 **AppFramework.Core** 项目中创建名为 Demo文件夹，里面创建一个表名为 **AbpDemo** ，如下:

```
    [Table("AbpDemos")]
    public class AbpDemo : Entity, IMayHaveTenant
    {
        public int? TenantId { get; set; }
        public string Name { get; set; }
        public string Sex { get; set; }
        public string Address { get; set; }
    }
```

> IMayHaveTenant 为多租户接口

第二步在 **AppFramework.EntityFrameworkCore** 项目当中找到 AppFrameworkDbContext 类， 新增 AbpDemo表定义, 如下:

```
 public class AppFrameworkDbContext : AbpZeroDbContext<Tenant, Role, User,
AppFrameworkDbContext>, IAbpPersistedGrantDbContext
    {
        public virtual DbSet<AbpDemo> AbpDemos { get; set; }

        //.....
    }
```

第三步打开 工具 > NuGet包管理器 > 程序包管理器控制台 , 将默认项目设置设置为 EntityFrameworkCore,

依次输入:

```
add-migration xxxx
```

> xxxx 是根据你实际输入

```
update-database
```

> 生成完成后, 检查表是否已生成至数据库当中。

第四步, 打开 AppFramework.Core > Authorization ， 找到 AppAuthorizationProvider 、AppPermissions。

在AppPermissions 添加权限的名称常量定义, 如下:

```
        public const string Pages_AbpDemos = "Pages.AbpDemos";
        public const string Pages_AbpDemos_Create = "Pages.AbpDemos.Create";
        public const string Pages_AbpDemos_Edit = "Pages.AbpDemos.Edit";
        public const string Pages_AbpDemos_Delete = "Pages.AbpDemos.Delete";
```

在AppAuthorizationProvider 添加权限的子节点, 如下:

```
 var abpDemos = pages.CreateChildPermission(AppPermissions.Pages_AbpDemos,
L("AbpDemos"));
            abpDemos.CreateChildPermission(AppPermissions.Pages_AbpDemos_Create,
L("CreateNewAbpDemo"));
            abpDemos.CreateChildPermission(AppPermissions.Pages_AbpDemos_Edit,
L("EditAbpDemo"));
            abpDemos.CreateChildPermission(AppPermissions.Pages_AbpDemos_Delete,
L("DeleteAbpDemo"));
```

第五步，在前面几个步骤中, 已经完成了对数据库表以及权限定义的操作, 接下来主要是创建用于Web服务的接口，以及接口实现。在 AppFramework.Application.Shared 项目中, 创建Demo文件夹， 添加IAbpDemoAppService 接口 以及Dtos文件夹，接口定义如下:

```
    public interface IAbpDemoAppService : IApplicationService
    {
        Task<PagedResultDto<AbpDemoDto>> GetAll(GetAllAbpDemoInput input);
    }
```

在Dtos中 定义 AbpDemoDto ,GetAllAbpDemoInput ， 如下:

```
    public class AbpDemoDto : EntityDto
    {
        public string Name { get; set; }

        public string Sex { get; set; }

        public string Address { get; set; }
    }

    public class GetAllAbpDemoInput : PagedAndSortedResultRequestDto
    {
        public string Filter { get; set; }
```

```
        public string NameFilter { get; set; }
    }
```

第六步，在 AppFramework.Application 项目中，创建Demo文件夹，添加AbpDemoAppService，如下:

通过构造函数, 注入了AbpDemo表的仓储服务，用于访问数据库。

```
    [AbpAuthorize(AppPermissions.Pages_AbpDemos)]
    public class AbpDemoAppService : AppFrameworkAppServiceBase,
IAbpDemoAppService
    {
        private readonly IRepository<AbpDemo> _repository;

        public AbpDemoAppService(IRepository<AbpDemo> repository)
        {
            _repository = repository;
        }

        public async Task<PagedResultDto<AbpDemoDto>> GetAll(GetAllAbpDemoInput
input)
        {
            var filteredAbpModels = _repository.GetAll()
                        .WhereIf(!string.IsNullOrWhiteSpace(input.Filter), e =>
false || e.Name.Contains(input.Filter) || e.Name.Contains(input.Filter) ||
e.Address.Contains(input.Filter))
                        .WhereIf(!string.IsNullOrWhiteSpace(input.NameFilter), e
=> e.Name == input.NameFilter);

            var pagedAndFilteredAbpVersions = filteredAbpModels
                .OrderBy(input.Sorting ?? "id asc")
                .PageBy(input);

            var totalCount = await filteredAbpModels.CountAsync();
            var dbList = await pagedAndFilteredAbpVersions.ToListAsync();
            var results = ObjectMapper.Map<List<AbpDemoDto>>(dbList);

            return new PagedResultDto<AbpDemoDto>(totalCount, results);
        }
    }
```

> 该类继承于 AppFrameworkAppServiceBase, 包含ABP内部提供的用户、租户等信息

在该类中, 使用了 ObjectMapper 将AbpDemo 转换成AbpDemoDto, 所以我们还需要在 AppFramework.Application 项目当中的 CustomDtoMapper中添加 表的转换映射关系，如下:

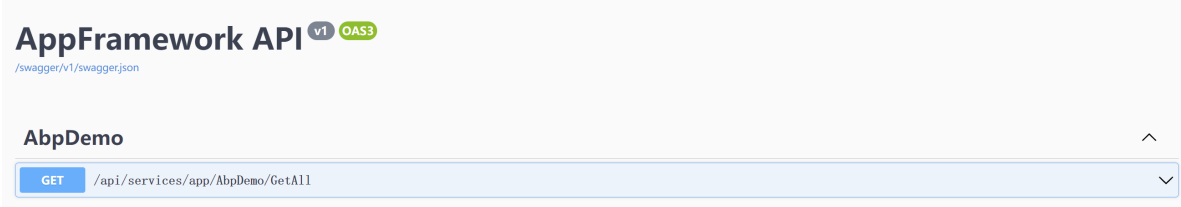
```
 internal static class CustomDtoMapper
    {
        public static void CreateMappings(IMapperConfigurationExpression
configuration)
        {
            configuration.CreateMap<AbpDemoDto, AbpDemo>().ReverseMap();

            //....
        }
    }
```

第七步，通过上面的步骤，已经完成了对单个表的访问数据功能, 还有涉及到多语言的设置, 需要在 AppFramework.Core > Localization > AppFramework 当中进行设置， 如下所示:

```
    <text name="DemoManager">演示</text>
    <text name="DemoManagerInfo">管理演示信息</text>
    <text name="DemoName">姓名</text>
    <text name="DemoSex">性别</text>
    <text name="DemoAddress">地址</text>
    <text name="CreateNewAbpDemo">创建新数据</text>
    <text name="EditAbpDemo">编辑</text>
    <text name="DeleteAbpDemo">删除</text>
```

> 需要涉及多语言的定义, 都可以在不同的国家语言中进行添加配置，表字段多语言，标题等。

完成后, 启动Web.Host，通过Swagger 查看接口已经显示在API列表当中，如下:



2.创建WPF客户端

第一步，在 AppFramework.Application.Client 项目当中, 创建Demo文件夹，创建 AbpDemoAppService 服务，实现IAbpDemoAppService 接口，如下:

```
public class AbpDemoAppService : ProxyAppServiceBase, IAbpDemoAppService
    {
        public AbpDemoAppService(AbpApiClient apiClient) : base(apiClient)
        { }

        public async Task<PagedResultDto<AbpDemoDto>> GetAll(GetAllAbpDemoInput
input)
        {
            return await ApiClient.GetAsync<PagedResultDto<AbpDemoDto>>
(GetEndpoint(nameof(GetAll)), input);
        }
    }
```

> ProxyAppServiceBase 为Web服务基类, 包含用于Web请求的 ApiClient。

第二步，在 AppFramework.Shared 项目当中, SharedModuleExtensions 当中, 注入 AbpDemoAppService 服务。

```
services.RegisterScoped<IAbpDemoAppService, AbpDemoAppService>();
```

第三步，在 AppFramework.Admin 项目当中, 创建ViewModel层,如下:

```
public class DemoViewModel: NavigationCurdViewModel
    {
        private readonly IAbpDemoAppService appService;

        public GetAllAbpDemoInput input;

        public DemoViewModel(IAbpDemoAppService appService)
        {
            Title = Local.Localize("DemoManager");
            this.appService = appService;
            input = new GetAllAbpDemoInput()
            {
                MaxResultCount = 10
            };
            dataPager.OnPageIndexChangedEventhandler +=
DataPager_OnPageIndexChangedEventhandler;
        }

        private void DataPager_OnPageIndexChangedEventhandler(object sender,
PageIndexChangedEventArgs e)
        { }

        public override async Task OnNavigatedToAsync(NavigationContext
navigationContext = null)
        {
            await SetBusyAsync(async () =>
            {
                await GetAbpDemos(input);
            });
        }

        private async Task GetAbpDemos(GetAllAbpDemoInput filter)
        {
            await WebRequest.Execute(() => appService.GetAll(filter),
dataPager.SetList);
        }
    }
```

> 继承 NavigationCurdViewModel 代表该模块具备一般Curd实现, 重写 OnNavigatedToAsync 方法, 主要用于, 当前模块被导航时首先被触发的方法, 主要作用用于加载并且处理数据。

在任意一种UI框架当中新建 View 视图，并且注入容器当中。

```
services.Add<DemoView, DemoViewModel>(AppViews.Demo);
```

第四步， 打开 AppFramework.Admin > Services > Navigation > NavigationMenuService.cs 文件，新增Demo菜单，如下所示:

```
    new NavigationItem("demo","Demos",AppViews.Demo,AppPermissions.AbpDemos)
```

> AppPermissions 中需要添加对应的权限定义, 这一步骤与后端一致，复制即可。

```
public class AppPermissions
{
    public const string AbpDemos = "Pages.AbpDemos";
    public const string AbpDemosCreate = "Pages.AbpDemos.Create";
    public const string AbpDemosEdit = "Pages.AbpDemos.Edit";
    public const string AbpDemosDelete = "Pages.AbpDemos.Delete";
 }
```

完成以上步骤后, 启动应用程序!

常见问题

- 如果请求Web服务接口404, 检查对应的名称是否一致, 首先检查后端的服务是否存在。
- 请求服务500错误, 先检查后端的服务中是否正常
- 登录后,发现模块并不存在, 检查当前用户是否存在该模块的权限? 后端是否进行该模块的权限定义?
- 客户端调试模式启动直接报出异常, 请检查后端服务是否正常启动? 客户端请求的Web服务地址是否一致?
- 模块当中的多语言未生效 , 请检查后端多语言配置中是否存在对应的字符串定义?