Abp基于模型系统，所以推荐你新建的项目都应该模块化

**定义模块类**

我们需要再项目中安装包：Install-Package Volo.Abp.Core

定义一个模块我们只需要继承AbpModule即可

public class BlogModule : AbpModule

{

    public override void ConfigureServices(ServiceConfigurationContext context)

    {

        //...

    }

}

如果我们模块需要依赖其他模块，则通过DependsOn指定，同理其他模块依赖于我们模块也一样

[DependsOn(typeof(AbpAspNetCoreMvcModule))]

[DependsOn(typeof(AbpAutofacModule))]

public class BlogModule

{

    //...

}

**模块的生命周期函数**

- PreConfigureServices：配置前

- ConfigureServices：前置时

- PostConfigureServices：配置后

- OnPreApplicationInitialization：应用程序启动前

- OnApplicationInitialization：启动时

- OnPostApplicationInitialization：启动后

- OnApplicationShutdown：应用程序关闭时

**Abp提供的模块**

https://docs.abp.io/zh-Hans/abp/latest/Modules/Index

**模块化插件**

模块可配置为可插拔的dll

如下我们再程序启动时动态加载文件夹下的模块

public class Startup

{

    public void ConfigureServices(IServiceCollection services)

    {

        services.AddApplication<MyPlugInDemoWebModule>(options =>

        {

            options.PlugInSources.AddFolder(@"D:\Temp\MyPlugIns");

        });

    }

    public void Configure(IApplicationBuilder app)

    {

        app.InitializeApplication();

    }

}

ABP将在这个目录中发现这些模块，像其它模块一样配置和初始化它们

**生命周期函数 PreConfigureServices**

PreConfigureServices对我们修改Abp模块有很大的帮助

在观察源码时会经常看到ExecutePreConfiguredActions这个方法

IIdentityServerBuilder identityServerBuilder = services.AddIdentityServer(options =>

{

});

...

services.ExecutePreConfiguredActions(identityServerBuilder);

这说明我们可以在与配置中对IIdentityServerBuilder进行操作，如下

public override void PreConfigureServices(ServiceConfigurationContext context)

{

    // 对IIdentityServerBuilder进行操作

    PreConfigure(delegate (IIdentityServerBuilder options)

    {

        options.AddResourceOwnerValidator<IceResourceOwnerPasswordValidator>();

    });

    base.PreConfigureServices(context);

}