

第13期-依赖注入相关配置

2020年7月31日 9:39

依赖注入概要

ABP 的依赖注入系统是基于微软依赖注入扩展库的，如果要使用动态拦截器，需要依赖于 Autofac 开源库，目的是为了实现在动态代理。

模块化

由于ABP是一个模块化框架，因此每个模块都定义它自己的服务并在它自己的单独模块类中通过依赖注入进行注册。

```
public class BlogModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        //在此处注入依赖项
    }
}
```

约定大于配置的原则

ABP 默认依赖注入约定

- 模块类注册为Singleton。
- MVC控制器（继承Controller或AbpController）被注册为Transient。
- MVC页面模型（继承PageModel或AbpPageModel）被注册为Transient。
- MVC视图组件（继承ViewComponent或AbpViewComponent）被注册为Transient。
- 应用程序服务（实现IApplicationService接口或继承ApplicationService类）注册为Transient。
- 存储库（实现IRepository接口）注册为Transient。
- 域服务（实现IDomainService接口）注册为Transient。

禁用默认约定

你可以通过重写 PreConfigureServices 方法，设置SkipAutoServiceRegistration 为 true 。

```
public class BlogModule : AbpModule
```

```
{
    public override void PreConfigureServices(ServiceConfigurationContext
context)
    {
        SkipAutoServiceRegistration = true;
    }
}
```

一旦跳过自动注册，你应该手动注册你的服务，在这种情况下，AddAssemblyOf 扩展方法可以帮助你依照约定注册所有服务。

```
public class BlogModule : AbpModule
{
    public override void PreConfigureServices(ServiceConfigurationContext
context)
    {
        SkipAutoServiceRegistration = true;
    }

    public override void ConfigureServices(ServiceConfigurationContext
context)
    {
        context.Services.AddAssemblyOf<BlogModule>();
    }
}
```

依赖注入接口约定

如果实现这些接口,则会自动将类注册到依赖注入:

- ITransientDependency 注册为Transient生命周期。
- ISingletonDependency 注册为Singleton生命周期。
- IScopedDependency 注册为Scoped生命周期。

依赖注入 Attribute 特性约定

配置依赖注入服务的另一种方法是使用 DependencyAttribute，它具有以下属性：

- Lifetime：注册的生命周期：Singleton，Transient 或 Scoped。
- TryRegister：设置 true 则只注册以前未注册的服务，使用IServiceCollection 的 TryAdd ... 扩展方法。
- ReplaceServices：设置 true 则替换之前已经注册过的服务，使用 IServiceCollection的 Replace扩展方法。

```
[Dependency(ServiceLifetime.Transient, ReplaceServices = true)]
public class TaxCalculator
```

ExposeServicesAttribute 用于控制相关类提供了什么服务

```
[ExposeServices(typeof(ITaxCalculator))]  
public class TaxCalculator: ICalculator, ITaxCalculator, ICanCalculate,  
ITransientDependency  
{  
  
}
```

如果你未指定要公开的服务，则ABP依照约定公开服务，使用 I+类名约定：

ICalculator 接口和 Calculator 类关联，ITaxCalculator 接口和 TaxCalculator 类。

多种方式符合使用

```
[Dependency(ReplaceServices = true)]  
[ExposeServices(typeof(ITaxCalculator))]  
public class TaxCalculator : ITaxCalculator, ITransientDependency  
{  
  
}
```

手动注册服务

```
public class BlogModule : AbpModule  
{  
    public override void ConfigureServices(ServiceConfigurationContext  
context)  
    {  
        //注册一个singleton实例  
        context.Services.AddSingleton<TaxCalculator>(new  
TaxCalculator(taxRatio: 0.18));  
  
        //注册一个从IServiceProvider解析得来的工厂方法  
        context.Services.AddScoped<ITaxCalculator>(sp =>  
sp.GetRequiredService<TaxCalculator>());  
    }  
}
```

使用注册的服务

构造函数，属性注入和从 IServiceProvider 解析服务3中方式。

构造函数

```

public class TaxAppService : ApplicationService
{
    private readonly ITaxCalculator _taxCalculator;

    public TaxAppService(ITaxCalculator taxCalculator)
    {
        _taxCalculator = taxCalculator;
    }

    public void DoSomething()
    {
        //... 使用 _taxCalculator...
    }
}

```

基于 Autofac 属性注入

```

public class MyService : ITransientDependency
{
    public ILogger<MyService> Logger { get; set; }

    public MyService()
    {
        Logger = NullLogger<MyService>.Instance;
    }

    public void DoSomething()
    {
        //... 使用 Logger 写日志...
    }
}

```

属性注入与构造函数，推荐构造函数注入，属性可选注入。

从 IServiceProvider 解析服务

```

public class MyService : ITransientDependency
{
    private readonly IServiceProvider _serviceProvider;

    public MyService(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    public void DoSomething()
    {
        var taxCalculator =
        _serviceProvider.GetService<ITaxCalculator>();
    }
}

```

```

        //...
    }
}

using (var scope = _serviceProvider.CreateScope())
{
    var service1 = scope.ServiceProvider.GetService<IMyService1>();
    var service2 = scope.ServiceProvider.GetService<IMyService2>();
}

```

服务注册通知

```

public class AppModule : AbpModule
{
    public override void PreConfigureServices(ServiceConfigurationContext
context)
    {
        context.Services.OnRegistered(ctx =>
        {
            if
            (ctx.ImplementationType.IsDefined(typeof(MyLogAttribute), true))
            {
                ctx.Interceptors.TryAdd<MyLogInterceptor>();
            }
        });
    }
}

```