.Net Core

依赖注入

一．概念

1. **控制反转（Inversion Of Control IOC）**是设计模式中的一种思想；它的目的就是把“创建对象和组装对象”操作的控制权从业务逻辑的代码中转移到框架中，这样的业务代码中只要说明“我需要某个类型的对象”，框架就会帮助我们创建这个对象。

```C#
    private readonly IUserDataProvider _userDataProvider;

    public UserService( IUserDataProvider userDataProvider)
    {
        this._userDataProvider = userDataProvider;
    }
```

2. 控制反转这种思想实现有两种方式：
   a. **服务定位器（service locator）**；
   b. **依赖注入（DI）**；
3. 控制反转中两个重要的部分：
   a. **"容器"**: 负责提供对象的注册和获取功能的框架
   b. "**服务**": 注册到容器中的对象

   总结：控制反转就是把"我创建对象", 变成"我要对象"。

二、基本使用

依赖注入框架中注册服务之前要知道一个重要的概念叫作"生命周期"，简单理解就是获取服务的时候是在创建一个新对象还是用之前的对象。

生命周期：

1. 瞬态（transient）：每次被请求的被请求的时候都会创建一个新的对象。优点：避免多段代码用于同一个对象而造成对象状态混乱；缺点：生成的对象多，容易浪费内存。（谨慎使用）
2. 范围（scoped）：在给定的范围，多次请求共享同一个服务对象，服务每次被请求的时候都会返回同一个对象；在不同范围内，服务每次被请求的时候会返回不同的对象。范围可以框架定义，也可以自己定义；在 ASP.NET Core 中，服务范围默认是一次 HTTP 请求，也就是在同一次 HTTP 请求中，不同的注入会获得同一个对象。适用于在同一范围内共享同一个对象的情况。
3. 单例（singleton）：全局共享一个服务对象。适用于服务无状态对象。

总结：服务对象无状态用单例（singleton），一个对象有状态且在框架环境中有范围控制用范围（scoped）；瞬态（transient）尽量在子范围中使用它们，而不要在跟范围中使用它们，控制不好容易造成内存泄漏；

线程与进程：https://blog.csdn.net/mu_wind/article/details/124616643?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216990675671680018064 9256%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%2 57D&request_id=169906756716800180649256&biz_id=0&utm_medium=distribute.pc _search_result.none-task-blog-2~all~top_positive~default-1-124616643-null-null.142 ^v96^pc_search_result_base9&utm_term=%E7%BA%BF%E7%A8%8B%E5%92%8 C%E8%BF%9B%E7%A8%8B%E7%9A%84%E5%8C%BA%E5%88%AB&spm=1018.2 226.3001.4187

配置 Autofac

一、引用 Autofac 所需要的包：

Autofac 包：提供依赖注入的功能；

Autofac.Extensions.DependencyInjection 包：提供 将默认依赖注入的容器替换为 Autofac 的功能。

二、注入服务：1. 通过 ContainerBuilder 容器进行注入

C#

```
1   public static void Main(string[] args)
2     {
3         var configuration = new ConfigurationBuilder()
4             .AddJsonFile("appsettings.json")
5             .AddEnvironmentVariables()
6             .Build();
7
8
9         CreateHostBuilder(args).Build().Run(); //如果没有改代码，启动立马就
  会停止
10
11    }
12
13    public static IHostBuilder CreateHostBuilder(string[] args) =>
14        Host.CreateDefaultBuilder(args)
15            .UseServiceProviderFactory(new AutofacServiceProviderFactor
  y())
16            .ConfigureWebHostDefaults(webBuilder => { webBuilder.UseSta
  rtup<Startup>(); });
17    //        .ConfigureContainer<ContainerBuilder>(builder =>
18    // builder.RegisterType<HelloWordService>().As<IHelloWordService>
  ());
```