ASP.NET Core 基础系列（4）(Startup)

原创 痕迹 CodeShare 1周前

收录于话题
#ASP.NET Core

8个

Startup

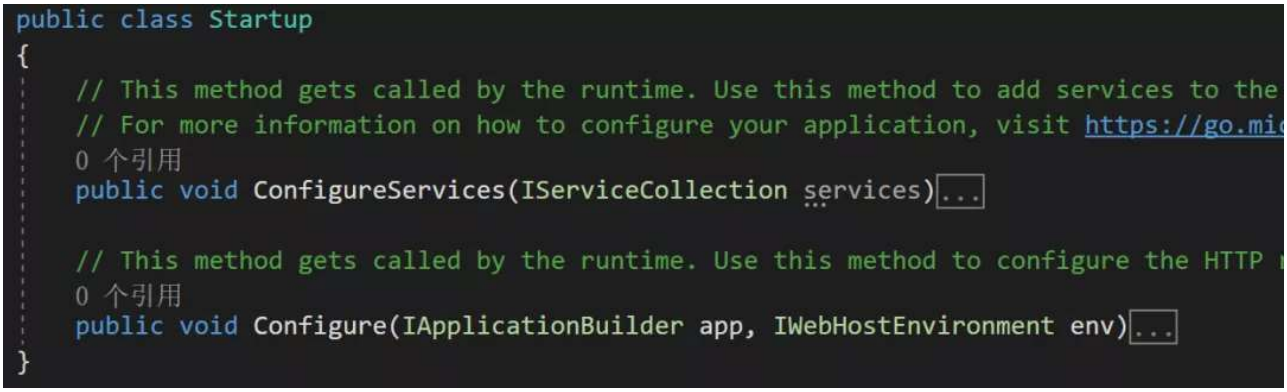
ASP.NET Core必须包含Startup类。它就像 Global.asax 文件，我们传统的 .NET 应用程序。如名称建议的那样，在应用程序启动时首先执行它。在程序类的Main方法中配置主机时，可以使用 UseStartup() 扩展方法配置启动类。请查看下面的程序类，并重点介绍 WebBuilder.UseStartup() 方法。

```
public class Program
{
    0 个引用
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    1 个引用
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

名称"Startup"是按照ASP.NET Core约定进行的。但是，您可以给Startup类指定任何名称，只需在UseStartup()方法中将其指定为通用参数即可。
例如，要将启动类命名为MyStartup，则将其指定为UseUseup ()。

通过在解决方案资源管理器中单击Startup.cs类文件，在Visual Studio中打开Startup类。以下是ASP.NET Core 3.x中的默认启动类。



如上面在代码中看到的，Startup类包含两个公共方法：ConfigureServices和Configure。Startup类必须包含Configure方法，并且可以选择包含ConfigureService方法。

ConfigureServices()方法

依赖注入模式在ASP.NET Core体系结构中大量使用。它包括内置的IoC容器，以使用构造函数提供相关的对象。
在ConfigureServices方法中，可以使用内置IoC容器注册依赖类。
注册依赖类后，可以在应用程序中的任何位置使用它。您只需要在要使用它的类的构造函数的参数中包含它即可。IoC容器将自动注入它。
ASP.NET Core将依赖类称为服务。因此，每当您阅读“服务”，然后将其理解为将在其他一些类中使用的类。
ConfigureServices方法包含IServiceCollection参数，以将服务注册到IoC容器。
例如，如果要将RazorPages服务或MVC服务添加到asp.net核心应用程序，则需要将这些服务添加到该方法接受的参数中，如下图所示。

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();
    services.AddMvc();
}
```

Configure()方法

在Configure方法中，我们可以使用内置IoC容器提供的IApplicationBuilder实例为asp.net核心应用程序配置应用程序请求管道。
ASP.NET Core引入了中间件组件来定义请求管道，该管道将在每个请求上执行。您仅包括应用程序所需的那些中间件组件，从而提高了应用程序的性能。
带有Empty模板的ASP.NET Core应用程序的默认配置方法包括以下三个中间件，如下图所示。

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment()) //是否处于开发模式
    {
        //异常中间件
        app.UseDeveloperExceptionPage();
    }
    //路由
    app.UseRouting();

    //终结点
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGet("/", async context =>
        {
            //输出当前进程名称
            await context.Response.WriteAsync($"Process:{System.Diagnostics.Process.GetCurrentProcess().ProcessName}");
        });
    });
}
```