ASP.NET Core

Creedon

序章 ASP.NET Core 介绍与安装

# ASP.NET Core特点

* 微软一开始发展就开源的平台。
* 跨平台。支持Windows、Linux、Mac等。
* 高性能、精简、模块化。

# 新特性（区别于.NET Framework）

* 新的Core CLR与RyuJIT编译器。
* ASP.NET基于一系列颗粒化，良好构建的NuGet包。不再基于System.Web.dll。
* ASP.NET Core的文件伺服：白名单策略，只有wwwroot会被伺服。传统ASP.NET文件伺服：黑名单策略，除了一些重要文件，根目录文件都会被伺服

# ASP.NET安装与配置

1. 下载安装ASP.NET Core SDK(开发工具包)。
2. 下载安装ASP.NET Core Runtime(运行时)。
3. 使用Visual Studio 2017或者VS Code开发。
4. ASP.NET Core应用程序介绍

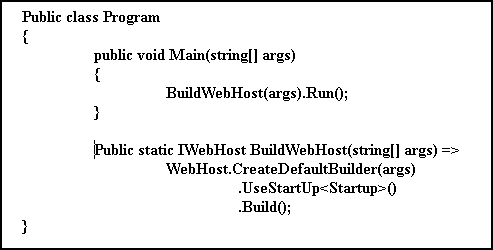
# 概述

ASP.NET Core应用其实就是一个在Main方法中创建一个Web服务器的简单控制台应用程序。

# Program类与Main方法

Main方法调用WebHost.CreateDefaultBuilder,按照生成器(builder)模式创建一个Web应用程序主机（Host）。生成器提供了定义Web服务器和Startup类的方法。

Build和Run方法生成IWebHost对象，然后启动它并开始侦听HTTP请求。



# Startup类

WebHostBuilder的UseStartup方法指定Startup类。

Startup类的两个方法：

1. ConfigureServices方法用于注册你的应用所使用的服务。(注册ASP.NET Core MVC、EF Core、Identity等)

（注：这些服务有三种生命周期：Transient、Scoped、Singeton。）

2.Configure方法用于定义你的请求管道中间件。

(参数： IapplicationBuilder app、IhostingEnvironmentenv、IConfiguration configuration以及一些其他服务。可以使用configuration[“…”]来调用appsetting.json里面的配置信息)

# 静态文件

静态文件如HTML、CSS、图片、脚本等资源会被ASP.NET Core直接提供给客户端（伺服），他们在网站根目录（wwwroot）里面。

# 服务器

ASP.NET Core托管模式不直接监听请求，而是依赖一个HTTP Server实现转发请求到应用程序。

ASP.NET Core包含了一个托管的跨平台Web服务器Kestrel，它往往会被运行在IIS等生产Web服务器之后。

# 配置

不再使用Web.config，而是appsettings.json。

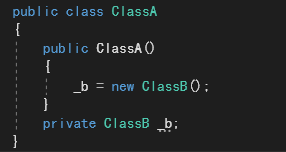
补充：ConfigureServices和依赖注入的详细介绍

# 控制反转（IOC）和依赖注入（DI）

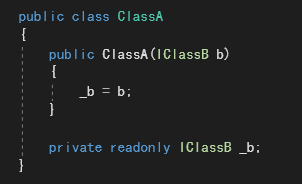
依赖注入是控制反转的别名。

**已知:**类A依赖类B。

**控制正转：**直接在类A里实例化依赖对象。



**控制反转：**不在类A里面实例化类B，而是其他方式将依赖注入。



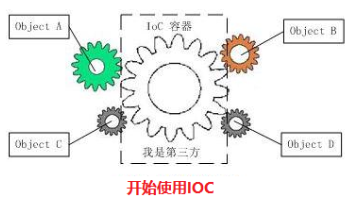
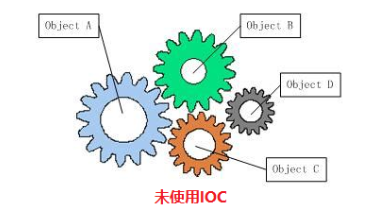
由IOC容器通过构造函数参数将B的实例对象注入类A。称为构造函数注入。其他的方式还有方法注入和属性注入。

**优势：**依赖注入的好处是解耦和，便于调试维护。

**注意：** 依赖抽象而不要依赖具体类，面向接口(面向抽象)编程而不要面向实现编程。

# IOC容器

IoC容器本质是一个工厂，负责提供向它请求的类型示例。全部对象的控制权全部上缴给“第三方”IOC容器，IoC容器通常还会管理对象的生命周期。



# ConfigureServices方法里面注册服务（依赖）

需要注入的实例或实现称为服务（service）。Creedon: ASP.NET Core内置IoC容器ServiceProvider，它管理的类型为services。

**注册Core提供的服务**

例：services.AddDbContext<ApplicationDbContext>(…);

services.AddMvc();

**注册自定义的服务**

例：services.AddSingleton<IProductService,ProductService>();

(当IProductService被请求时，容器返回ProduceService的实例)

**服务的生命周期**

Transient(瞬时)：在它们每次被请求时被创建。

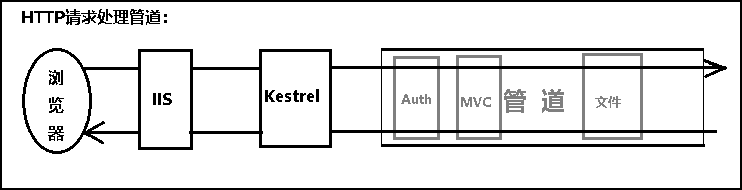
Scoped(作用域):在每次请求时被创建一次。

Singeton(单例):只被创建一次，后续请求使用相同实例。

补充：Configure和中间件详细介绍

# HTTP请求管道简介

发使用中间件构建你的请求处理管道:



（ASP.NET Core的内置中间件：静态文件StaticFiles、路由Routing、身份验证Authentication等）

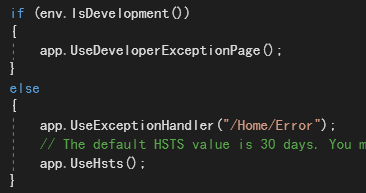
# 中间件简介

中间件是用于组成应用程序管道来处理请求和响应的组件。管道内每一个组件都可以选择是否将请求交给下一个组件，并在管道内调用下一组件之前和之后执行某些操作。

如果不把请求交给下一个组件，而是自己处理该请求，这就称为请求管道的短路。

**ASP.NET Core中间件为一个HttpContext执行异步逻辑，然后按顺序调用下一个中间件或者直接终止请求。除了预定义的中间件以外。还可以自定义中间件。**

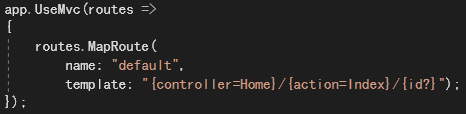
1. **中间件示例：**



（开发环境和非开发环境的）错误处理。

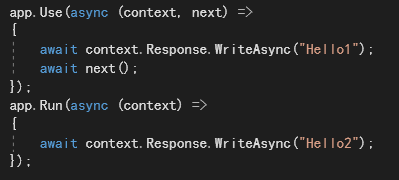


静态文件服务器。



MVC。

**2、自定义中间件示例：**

**

（Run方法会短路管道，因为他不会调用next请求委托。因此Run方法一般只在管道底部被调用。）

补充：静态文件

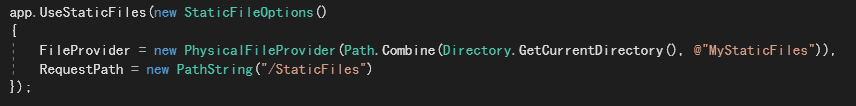
# 概述

静态文件（StaticFiles），例如HTML、CSS、图片、脚本之类的资源会被直接提供给客户端。静态文件通常位于网站根目录(wwwroot)中。

为了能使用静态文件服务，必须把静态文件中间件加入管道。App.UseStaticFiles()使wwwroot下的文件可以被访问。

# 其他目录下的内容访问（白名单）

通过给App.UseStaticFiles()添加一个StaticFileOptions类对象参数将其他目录下的内容页向外提供服务。app.UseStaticFiles(new StaticFileOptions(){…})



# 静态文件授权

静态文件默认不提供授权检查。任何通过该模块提供访问的文件，包括wwwroot下的文件都是公开的。

如果要实现授权才能访问，需要将文件保存在wwwroot之外，并将目录设置为可被静态文件中间件访问到，同时通过一个Controller Action来访问，通过授权后返回FileResult。

# 默认文件

设置默认首页能给站点的每个访问者提供一个起始页。

为了使站点能够提供默认页，避免用户输入完整URI,必须在Configure中调用app.UseDefaultFiles()方法，而且必须先app.UseStaticFiles()开启静态文件中间件。

（会自动检索以下文件：default.html、defaul.htmt、Index.html、Index.htm。）

# UseFileServer

它集合了UseStaticFiles、UseDefaultFiles以及UseDirectoryBrowser。

调用app.UserFileServer(enableDirectoryBrowseing:true)启用静态文件、默认文件以及目录浏览功能。并且还需要在ConfigureServices中调用services.AddDirectoryBrowser()方法。

# （了解）FileExtensionContentTypeProvider

File FileExtensionContentTypeProvider类包含一个将文件扩展名映射到MIME内容类型的集合。用于自定义类型文件（.rew .zqj）。

1. 配置MVC

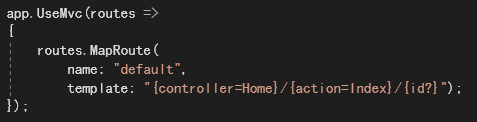
# 注册MVC服务到IOC容器

在ConfigureServices里面services.AddMvc();注册MVC相关的服务到IOC容器。



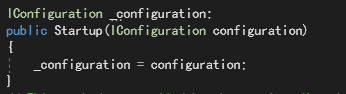
# 在管道里使用和配置MVC中间件

在Configure里面App.UseMvc();使用和配置MVC中间件。



# 注入IConfiguration

（命名空间Microsoft.Extensions.Configuration）



# 其他

配置其他的服务和中间件。

