# **什么是 ASP.NET Core？**

ASP.NET Core 是一个新的开源和跨平台的框架，用于构建如 Web 应用、物联网（IoT）应用和移动后端应用等连接到互联网的基于云的现代应用程序。ASP.NET Core 应用可运行于 [.NET Core](https://www.microsoft.com/net/core/platform) 和完整的 .NET Framework 之上。 构建它的目的是为那些部署在云端或者内部运行（on-premises）的应用提供一个优化的开发框架。它由最小开销的模块化的组件构成，因此在构建你的解决方案的同时可以保持灵活性。你可以在 Windows、Mac 和 Linux 上跨平台的开发和运行你的 ASP.NET Core 应用。 ASP.NET Core 开源在 [GitHub](https://github.com/aspnet/home) 上。

# **为什么构建 ASP.NET Core？**

ASP.NET 的首个预览版作为 .NET Framework 的一部分发布于15年前。自那以后数百万的开发者用它开发和运行着众多非常棒的 Web 应用，而且在这么多年之间我们也为它增加和改进了很多的功能。

ASP.NET Core 有一些架构上的改变，这些改变会使它成为一个更为精简并且模块化的框架。ASP.NET Core 不再基于 *System.Web.dll* 。当前它基于一系列颗粒化的，并且良好构建的 [NuGet](http://www.nuget.org/) 包。这一特点能够让你通过仅仅包含需要的 NuGet 包的方法来优化你的应用。一个更小的应用程序接口通过“只为你需要的功能付出”（pay-for-what-you-use）的模型获得的好处包括更可靠的安全性、简化服务、改进性能和减少成本。

通过 ASP.NET Core，你可以获得的改进：

* 一个统一的方式用于构建 web UI 和 web APIs
* 集成 [现代的客户端开发框架](https://docs.asp.net/en/latest/client-side/index.html) 和开发流程
* 一个适用于云的，基于环境的 [配置系统](https://docs.asp.net/en/latest/fundamentals/configuration.html)
* 内置的 [依赖注入](https://docs.asp.net/en/latest/fundamentals/dependency-injection.html)
* 新型的轻量级的、模块化 HTTP 请求管道
* 运行于 IIS 或者自宿主（self-host）于你自己的进程的能力
* 基于支持真正的 side-by-side 应用程序版本化的 [.NET Core](https://microsoft.com/net/core) 构建
* 完全以 [NuGet](https://nuget.org/) 包的形式发布
* 新的用于简化现代 web 开发的工具
* 可以在 Windows 、Mac 和 Linux 上构建和运行跨平台的 ASP.NET 应用
* 开源并且重视社区

# **应用程序剖析**

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

复制代码

using System;using Microsoft.AspNetCore.Hosting;

namespace aspnetcoreapp

{

public class Program

{

public static void Main(string[] args)

{

var host = new WebHostBuilder()

.UseKestrel()

.UseStartup<Startup>()

.Build();

host.Run();

}

}

}

Main 调用遵循 builder 模式的 **WebHostBuilder** ，用于创建一个 web 应用程序宿主。这个 builder 有些用于定义 web 服务器 （如 UseKestrel）和 startup 类型（ UseStartup）的方法。在上面的示例中，web 服务器 Kestrel 被启用，但是你也可以指定其它 web 服务器。我们将会在下一节展示更多关于 UseStartup 的内容。WebHostBuilder 提供了一些可选方法，其中包括寄宿在 IIS 和 IIS Express 中的 UseIISIntegration 和用于指定根内容目录的 UseContentRoot。Build 和 Run 方法构建了用于宿主应用程序的 IWebHost 然后启动它来监听传入的 HTTP 请求。

# **Startup**

WebHostBuilder 的 UseStartup 方法为你的应用指定了 Startup 类。

复制代码

public class Program

{

public static void Main(string[] args)

{

var host = new WebHostBuilder()

.UseKestrel()

.UseStartup<Startup>()//手工高亮

.Build();

host.Run();

}

}

Startup 类是用来定义请求处理管道和配置应用需要的服务。 Startup 类必须是公开的（public）并且包含如下方法：

复制代码

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

}

public void Configure(IApplicationBuilder app)

{

}

}

* ConfigureServices 定义你的应用所使用的服务（在下面查看 [服务（Services）](http://www.cnblogs.com/dotNETCoreSG/p/aspnetcore-0-intro.html" \l "services) ）（例如 ASP.NET MVC Core framework、Entity Framework Core、Identity 等等）
* Configure 定义你的请求管道中的 [中间件（middleware）](https://docs.asp.net/en/latest/fundamentals/middleware.html)
* 更多内容请参考： [Application Startup](https://docs.asp.net/en/latest/fundamentals/startup.html)

# **服务（Services）**

服务是应用中用于通用调用的组件。服务通过依赖注入获取并使用。 ASP.NET Core 内置了一个简单的控制反转（IoC） 容器，它默认支持构造器注入，并且可以方便的替换成你自己选用的 IoC 容器。由于它的松耦合特性，依赖注入（DI） 使服务在整个应用中都可以使用。例如，[Logging](https://docs.asp.net/en/latest/fundamentals/logging.html) 在你整个应用中都可用。查看 [Dependency Injection](https://docs.asp.net/en/latest/fundamentals/dependency-injection.html) 获取更多信息。

# **中间件（Middleware）**

在 ASP.NET Core 中，你可以使用 [Middleware](https://docs.asp.net/en/latest/fundamentals/middleware.html) 构建你的请求处理管道。 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。一般来说你要使用一个中间件，只需要在 Configure 方法里调用 IApplicationBuilder 上一个对应的 UseXYZ 扩展方法。

ASP.NET Core 带来了丰富的内置中间件：

* [静态文件（Static files）](https://docs.asp.net/en/latest/fundamentals/static-files.html)
* [路由（Routing）](https://docs.asp.net/en/latest/fundamentals/routing.html)
* [身份验证（Authentication）](https://docs.asp.net/en/latest/security/authentication/index.html)

你也可以创建你自己的 [自定义中间件](https://docs.asp.net/en/latest/fundamentals/middleware.html)。

你也可以在 ASP.NET Core 中使用任何基于 [OWIN](http://owin.org/) 的中间件。查看 [OWIN](https://docs.asp.net/en/latest/fundamentals/owin.html) 获取更多信息。

# **服务器（Servers）**

ASP.NET Core 托管模式并不直接监听请求；而是依赖于一个 HTTP [server](https://docs.asp.net/en/latest/fundamentals/servers.html) 实现来转发请求到应用程序。这个被转发的请求会以一组 feature 接口的形式被包装，然后被应用程序组合到一个 HttpContext中去。 ASP.NET Core 包含了一个托管的跨平台 web 服务器，被称为 [Kestrel](https://docs.asp.net/en/latest/fundamentals/servers.html" \l "kestrel)，它往往会被运行在一个如 [IIS](https://iis.net/) 或者 [nginx](http://nginx.org/) 的生产 web 服务器之后。

# **内容根目录（Content root）**

内容根目录是应用程序所用到的所有内容的根路径，例如它的 views 和 web 内容。内容根目录默认与宿主应用的可执行程序的应用根目录相同；一个替代的地址可以通过 *WebHostBuilder* 来设置。

# **Web根目录（Web root）**

你的应用的Web根目录（Web root）是你项目中所有公共的、静态的资源，如 css、js 和 图片文件的目录。静态文件中间件将默认只发布 Web 根目录（Web root）和其子目录中的文件。 Web 根目录（Web root）默认为 */wwwroot*，但是你也可以通过 *WebHostBuilder* 来指定另外一个地址。

# **配置（Configuration）**

ASP.NET Core 使用了一个新的配置模型用于处理简单的键值对。新的配置模型并非基于System.Configuration 或者 *web.config* ；而是从一个有序的配置提供者集合拉取数据。内置的配置提供者支持多种不同的文件格式如（XML，JSON， INI）和用于支持基于环境的配置环境变量。你也可以实现你自己自定义的配置提供者。查看 [Configuration](https://docs.asp.net/en/latest/fundamentals/configuration.html) 获取更多信息。

# **环境（Environments）**

环境，如 “Development” 和 “Production”，是 ASP.NET Core 中的第一级概念而且它可以设置成使用环境变量。查看 [Working with Multiple Environments](https://docs.asp.net/en/latest/fundamentals/environments.html) 获取更多信息。

# **使用 ASP.NET Core MVC 构建 web UI 和 web APIs**

* 你可以使用 Model-View-Controller（MVC）模式创建优秀的并且可测试的 web 应用程序。查看 [MVC](https://docs.asp.net/en/latest/mvc/index.html) 和 [测试](https://docs.asp.net/en/latest/testing/index.html)。
* 你可以构建支持多种格式并且完全支持内容协商的 HTTP 服务。 查看 [Formatting](https://docs.asp.net/en/latest/mvc/models/formatting.html)
* [Razor](http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c) 提供了一种高效的语言用于创建 [Views](https://docs.asp.net/en/latest/mvc/views/index.html)
* [Tag Helpers](https://docs.asp.net/en/latest/mvc/views/tag-helpers/intro.html) 启用服务器端的代码参与到 - Razor 文件的创建和 HTML 元素渲染
* 你可以使用自定义或者内置的 formatters （JSON， XML）来构建完全支持内容协商的 HTTP 服务
* [Model Binding 模型绑定](https://docs.asp.net/en/latest/mvc/models/model-binding.html) 自动的映射 HTTP 请求中的数据到 action 方法参数
* [Model Validation 模型验证](https://docs.asp.net/en/latest/mvc/models/validation.html) 自动的执行客户端和服务器端验证

# **客户端开发**

ASP.NET Core 在设计时已考虑到和各种客户端框架（[AngularJS](https://docs.asp.net/en/latest/client-side/angular.html)，[KnockoutJS](https://docs.asp.net/en/latest/client-side/knockout.html) 和 [Bootstrap](https://docs.asp.net/en/latest/client-side/bootstrap.html)）的无缝集成。查看 [Client-Side Development](https://docs.asp.net/en/latest/client-side/index.html) 获取更多信息。

http://www.cnblogs.com/dotNETCoreSG/p/aspnetcore-0-intro.html