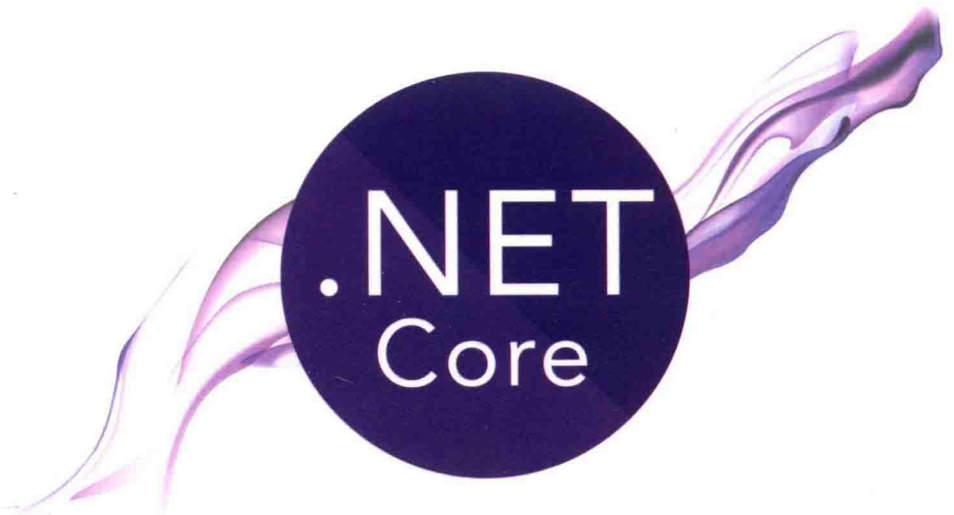
Broadview®
www.broadview.com.cn



ASP.NET Core跨平台开发入门之作

ASP.NET Core

跨平台开发从入门到实战

张剑桥 编著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

ASP.NET Core

跨平台开发从入门到实战

张剑桥 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书深入浅出地介绍了 ASP.NET Core 基础及实战方面的知识，主要有.NET Core 的基础知识及安装、dotnet 命令、ASP.NET Core 开发工具操作及使用、ASP.NET Core 原理和组件介绍、ASP.NET Core MVC 框架学习、ASP.NET Core Web API 学习和扩展 Web API 输出格式，以及.NETCore 单元测试，最后以一个完整的项目结尾，并讲解了项目的发布和部署。

本书既适合初学者及有.Net 基础的开发者，也适合作为院校教材。

图书在版编目（CIP）数据

ASP.NET Core 跨平台开发从入门到实战 / 张剑桥编著. —北京：电子工业出版社，2017.5
ISBN 978-7-121-31145-1

Ⅰ. ①A… Ⅱ. ①张… Ⅲ. ①网页制作工具－程序设计 Ⅳ. ①TP393.092.2

中国版本图书馆 CIP 数据核字(2017)第 057510 号

责任编辑：安　娜
印　　刷：北京中新伟业印刷有限公司
装　　订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　印张：20.5　字数：366 千字
版　　次：2017 年 5 月第 1 版
印　　次：2017 年 5 月第 1 次印刷
印　　数：3000 册　　定价：65.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

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 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，因此在构建解决方案的同时可以保持灵活性。我们可以在 Windows、Mac 和 Linux 上跨平台的开发和运行自己的 ASP.NET Core 应用。ASP.NET Core 开源在 GitHub（https://github.com/aspnet/home）上。

本书对 ASP.NET Core 进行了全面讲解，包括 ASP.NET Core 的 Web 框架以及 ASP.NET Core MVC 学习等，并从零开始讲解一个完整的 ASP.NET Core 项目开发及发布部署，带你走进 ASP.NET Core 跨平台开发的世界。

全书共分 11 章，内容如下：

第 1 章	NET Core。本章介绍了.NET Core 的基础知识以及.NET Core SDK 下载安装
第 2 章	dotnet 命令。本章详细介绍了 dotnet 命令语法及 dotnet 命令实战
第 3 章	VS Code 开发.NET Core。本章详细介绍了 VS Code 安装和开发调试.NET Core 应用，以及 VS Code C#插件配置介绍
第 4 章	VS 2015 开发.NET Core。本章详细介绍了 VS 2015 安装及开发.NET Core 应用
第 5 章	ASP.NET Core。本章详细介绍了 ASP.NET Core 及内置功能组件
第 6 章	EF Core。本章详细介绍了 EF Core 的两种使用：Database First 和 Code First
第 7 章	ASP.NET Core MVC。本章详细介绍了框架中的路由、模型、视图、控制器及新加入的视图组件标签助手等，以及依赖注入的使用
第 8 章	ASP.NET Core Web API。本章详细介绍了 Web API，自定义格式化
第 9 章	单元测试。本章详细介绍了单元测试及 xUnit.net 实战 MSTest 实战，测试控制器逻辑
第 10 章	项目实战 NetNote 系统。本章详细介绍了从零开始开发 NetNote 系统的流程和方法，其中包含 EF Core、Identity、以及中间件的开发使用
第 11 章	跨平台发布及部署。本章详细介绍了.NET Core 依赖框架部署及独立部署发布，部署至 Ubuntu、CentOS 系统以及 Docker 部署和 IIS 部署

希望读者在阅读完本书后，能够了解 ASP.NET Core 的基础及原理,学会使用 ASP.NET Core 开发简单的应用程序，并能将 ASP.NET Core 开发的应用程序部署至不同的系统中运行，实现 ASP.NET Core 的跨平台应用。

由于作者水平有限，书中疏漏之处在所难免，恳请读者批评指正。

目　录

第 1 章　.NET Core 1

1.1　.NET Core 介绍 1
1.2　.NET Core 跨平台 3
1.3　.NET Core SDK 下载安装 3

第 2 章　dotnet 命令 5

2.1　dotnet 命令介绍 5
2.2　dotnet-new 6
2.3　dotnet-restore 7
2.4　dotnet-run 8
2.5　dotnet-build 10
2.6　dotnet-test 11
2.7　dotnet-pack 14
2.8　dotnet-publish 15
2.9　dotnet 命令实战 17

第 3 章　VS Code 开发.NET Core 25

3.1　VS Code 安装及介绍 25
3.2　VS Code 开发调试.NET Core 26
3.3　VS Code C#插件配置介绍 30

第 4 章　VS 2015 开发.NET Core 32

4.1　VS 2015 安装及介绍 32
4.2　VS 2015 新建应用 33

第 5 章 ASP.NET Core 34

5.1 ASP.NET Core 介绍 34
5.2 Application Startup 38
5.3 中间件 42
5.4 静态文件 51
5.5 配置文件 63
5.6 日志（Logging） 70
5.7 依赖注入 DI 83
5.8 多环境 95
5.9 应用程序状态（Session） 99
5.10 程序缓存（Caching） 109

第 6 章 EF Core 113

6.1 EF Core 介绍 113
6.2 Code First 115
6.3 Database First 122

第 7 章 ASP.NET Core MVC 129

7.1 ASP.NET Core MVC 介绍 129
7.2 路由（Routing） 131
7.3 模型（Model） 156
7.4 视图（View） 165
7.5 布局（Layout） 171
7.6 标签助手（Tag Helpers） 177
7.7 局部视图（Partial Views） 190
7.8 视图组件（View Component） 194
7.9 控制器（Controller）与 Action 以及 Action Result 199
7.10 过滤器（Filter） 201
7.11 依赖注入 View 212
7.12 依赖注入 Controller 218
7.13 区域（Areas） 223

第 8 章　ASP.NET Core Web API 228

8.1　Web API 介绍 228
8.2　自定义格式化（Format） 232

第 9 章　单元测试 240

9.1　单元测试介绍 240
9.2　单元测试实战 xUnit.net 240
9.3　单元测试实战 MSTest 246
9.4　测试控制器逻辑 250

第 10 章　项目实战 NetNote 系统 258

10.1　新建项目 258
10.2　添加及查看 263
10.3　分类和分页 269
10.4　密码查看附件上传 276
10.5　Web API 282
10.6　Basic 基本认证中间件 285
10.7　用户登录 289
10.8　数据库切换 295

第 11 章　跨平台发布及部署 298

11.1　发布和部署 298
11.2　部署至 Ubuntu 系统 303
11.3　部署至 CentOS 系统 306
11.4　部署至 Docker 311
11.5　部署至 IIS 313

第1章
.NET Core

1.1 .NET Core 介绍

.NET Core 是.NET Framework 的新一代版本，是微软开发的第一个具有跨平台（Windows、Mac OSX、Linux）能力的应用程序开发框架，未来也将会支持 FreeBSD 与 Alpine 平台，是微软在一开始发展时就开源的软件平台，它也经常被拿来和现有的开源.NET 平台 Mono 比较。

由于.NET Core 的开发目标是跨平台的.NET 平台，因此.NET Core 会包含.NET Framework 的类库。与.NET Framework 不同的是，.NET Core 采用包化（Packages）的管理方式，应用程序只需获取需要的组件即可。与.NET Framework 大包式安装的做法截然不同，并且各包亦有独立的版本线，不再硬性要求应用程序跟随主线版本。

.NET Core 由许多项目所组成，除了基本的类库（Core FX）外，还包含了采用 RyuJIT 编译的运行平台 Core CLR、编译器平台.NET Compiler Platform、采用 AOT 编译技术运行最优化的包 Core RT（.NET Core Runtime），以及跨平台的 MSIL 编译器 LLILC（LLVM-based MSIL Compiler）等项目。

同时，微软也发展了一个构建技术文件的平台 docfx，并运用于.NET Core 的文件网站。

1. RyuJIT

RyuJIT 是微软发展的新式即地编译器（Just-in-Time Compiler），用以替换现有的.NET Framework 的 JIT 以及 JIT64 即地编译器。根据微软公布的测试报告，RyuJIT 的性能较前一代的 JIT 提升了约 25%，并支持 SIMD（Single Instruction, Multiple Data）技术。RyuJIT 同时应用

于.NET Framework 4.6 以及.NET Core[4]。

2. Core CLR

Core CLR 移植了.NET Framework 的 CLR 的功能，包含核心程序库 mscorlib、JIT 编译器、垃圾收集器（GC）以及其他运行 MSIL 所需要的运行期环境。

3. Core RT

Core RT 是以 AOT(Ahead-of-time)编译方式为主的核心功能，在.NET Core 内称为 Core RT，在 UWP（Universal Windows Platform，通用应用平台）则被称为.NET Native。

Core RT 会在构建时期（非运行期）在编译时将 MSIL 转换成平台本地的机器码，以优点是引导时间短（JIT 采用的是运行时期编译，使得引导时间拉长），并且内存用量少。Core RT 在不同的平台会使用不同的 AOT 技术：

◎ Windows 上使用的是.NET Native。
◎ Mac OS X 与 Linux 上使用的是 LLILC（同时支持 JIT 和 AOT）。

4. LLILC

LLILC（LLVM-based MSIL Compiler，英文发音为“lilac”）是.NET Core 在非 Windows 平台的 MSIL 编译器[5]，基于 ECMA-335（Common Language Infrastructure）的标准将 MSIL 编译成原生码运行，适用于可运行 LLVM 的操作系统，例如 Mac OS X 与 Linux 操作系统。

LLILC 同时支持 JIT（内含 RyuJIT 的实现）和 AOT（未来将开始支持）的编译方式。

5. Roslyn

.NET Compiler Platform（项目代码为 Roslyn）是将.NET 平台的编译架构标准化的平台，它可提供程序管理工具（如集成开发环境）相当多的情报，用以发展有助于编写程序与管理程序结构所需要的功能，如类型信息、语法结构、参考链接、语义、编译器、自动化、错误回报等功能，只要是遵循 CLI 标准的编程语言，都可以利用.NET Compiler Platform 实现编译器，让程序管理工具能够实现如语法提示、语法自动完成、关键字高亮等可视化功能。

.NET Compiler Platform 可同时支持.NET Framework 4.6 以上版本，.NET Core 也原生支持。

1.2 .NET Core 跨平台

.NET Core 拥有跨平台能力，并支持多种系统，让我们开发的程序，可以在多个系统中运行。

.NET Core 的 1.0 版本支持下列操作系统。

操作系统	版 本	平 台
Windows 客户端	7 SP1~10	x64, x86
Windows 服务器	R2 SP1~Windows Server 2016 全功能版、Server Core 与 Nano Server	x64, x86
Debian	8.2	x64
Red Hat Enterprise Linux	7.2	x64
Fedora	23	x64
Ubuntu	14.04 LTS, 16.04 LTS	x64
Linux Mint	17	x64
OpenSUSE	13.2	x64
Oracle Linux	7.1	x64
CentOS	7.1	x64
Mac OSX	10.11 (EI Capitan)	x64

1.3 .NET Core SDK 下载安装

.NET Core 1.0 发布以后，微软发布了一个新的 dotnet 官网：http://dot.net。

我们可以访问官网，下载安装 SDK 及 tool。

目前最新的.NET Core SDK win x64 版本下载地址是：https://go.microsoft.com/fwlink/?LinkID=809122。

更多系统版本，可到 https://www.microsoft.com/net/download 下载。有的系统只提供了执行文件，没有提供安装包，需要自己做一些配置。

下载好对应版本后就可以安装了。这里我们在 Windows 10 x64 系统下安装，直接单击安装包，然后一直单击“下一步”按钮即可安装好。

安装好以后我们就可以执行 dotnet 命令，来确认是否安装成功。打开命令提示符，输入 dotnet--info，如下图所示，则代表安装成功。

```
C:\>dotnet --info
.NET Command Line Tools (1.0.0-preview2-003121)

Product Information:
 Version:            1.0.0-preview2-003121
 Commit SHA-1 hash:  1e9d529bc5

Runtime Environment:
 OS Name:     Windows
 OS Version:  10.0.10586
 OS Platform: Windows
 RID:         win10-x64
```

第2章
dotnet命令

2.1 dotnet 命令介绍

dotnet 命令是开发.NET Core 应用程序的一个新的跨平台工具链的基础。它是跨平台的，并且对支持的每个平台有相同的表现范围。这意味着，当你学会如何使用工具后，你可以从任何支持的平台上以同样的方式使用它。你安装好 SDK 后就可以使用 dotnet 命令了。

默认它有如下命令。

```
dotnet-new
```

初始化 C#或 F#控制台应用程序项目，在当前目录创建新的.NET Core 项目。

```
dotnet-restore
```

还原指定应用程序的依赖项。

```
dotnet-build
```

生成.NET Core 程序。

```
dotnet-publish
```

发布.NET 可移植或独立应用程序。

```
dotnet-run
```

从源代码运行应用程序。

```
dotnet-test
```

使用 project.json 中指定的测试执行工具执行测试。

```
dotnet-pack
```

使用你的代码创建 NuGet 包。

下面具体介绍每一个命令的使用方法。

2.2 dotnet-new

dotnet new 命令提供了一个方便的方法来初始化一个有效的.NET Core 项目和示例源代码，用来试验命令行界面（CLI）工具集。

这个命令是在目录上下文中被调用。当被调用时，该命令会将两个主要项目放到当前目录：

❶ 包含示例“Hello World”程序的 Program.cs（或 Program.fs）文件。

❷ 有效的 project.json 文件。

放好后，该项目即可被编译和进一步编辑。

dotnet new 选如下。

```
-l -lang [C#|F#]
```

项目的语言。默认为 C#，可以指定为 F#。

```
-t -type
```

项目的类型，C#的有效值为 console、Web、lib 和 xunittest，对于 F#而言，仅 console 有效。

可以使用 dotnet new - h 来查看命令的帮助。

示例如下：

```
dotnet new或dotnet new --lang c#
```

在当前目录创建 C#控制台应用程序项目。

```
dotnet new -lang f#
```

在当前目录创建 F#控制台应用程序项目。

```
dotnet new -t web
```

在当前目录创建新的 ASP.NET Core C#应用程序项目。

2.3 dotnet-restore

dotnet-restore：还原项目的依赖项和工具。

dotnet restore 命令使用 NuGet 还原在 project.json 文件中被指定的依赖项，以及特定于项目的工具。默认情况下，依赖项和工具的还原是并行完成的。

对于依赖项，你可以在还原操作时使用--packages 参数指定还原包的位置。

如果没有指定，则默认使用 NuGet 包缓存。它可以在所有的操作系统上的用户目录下的.nuget/packages 目录中找到（例如，Linux 上的/home/user 或者是 Windows 上的 C:\Users\user）。Windows 下可以使用%HOMEPATH%/.nuget/packages 访问目录。

对于特定于项目的工具，dotnet restore 首先还原该工具的包，然后继续还原在 project.json 中指定的工具依赖项。

dotnet restore 选项如下。

```
[root]
```

还原的项目或者项目目录的列表。该列表可以是包含 project.json 文件的路径，或者是 global.json 文件路径或文件夹的路径。还原操作会以递归方式搜索所有子目录中的 project.json，并还原找到的每一个 project.json 文件。

```
-s, --source [SOURCE]
```

指定一个在还原操作期间使用的源。这覆盖所有在 NuGet.config 文件中指定的源。多个源可以通过多次指定该选项来提供。

```
--packages [DIR]
```

指定放置还原包的目录。

```
--disable-parallel
```

禁用并行还原多个项目。

```
-f, --fallbacksource [FEED]
```

在还原操作中，在其他所有源不能使用的情况下，可指定一个备用来源。所有有效的源格

式都是允许的。多个备用源可以通过多次指定该选项来提供。

```
--configfile [FILE]
```

用于还原操作的配置文件（NuGet.config）。

```
--verbosity [LEVEL]
```

使用日志详细级别。允许的值：Debug、Verbose、Information、Minimal、Warning 或者 Error。

可以使用 dotnet restore –h 来查看命令的帮助。

例如：

```
dotnet restore
```

还原在当前目录中的项目的依赖项和工具。

```
dotnet restore ~/projects/coreapp/project.json
```

还原在给定的路径发现 coreapp 项目依赖项和工具。

```
dotnet restore -f c:\packages\mypackages
```

还原在当前目录中的项目的依赖项和工具，使用文件路径作为备用源。

```
dotnet restore -f c:\packages\mypackages -f c:\packages\myotherpackages
```

还原在当前目录中的项目的依赖项和工具，使用两个文件路径作为备用源。

```
dotnet restore --verbosity Error
```

还原在当前目录中的项目的依赖项和工具，并在输出中仅显示 errors。

2.4 dotnet-run

dotnet-run：运行当前目录源代码。

dotnet run 命令提供了一个方便的选项，就是使用一个命令从源代码来运行你的应用程序。

它会编译源代码，生成输出程序，然后运行该程序。此命令既可用于快速迭代开发，也可用于运行源分布式程序（例如网站）。

此命令依赖于 dotnet build，以便在启动程序前生成.NET 程序集的源输入。

输出的文件被写到 bin 文件夹，如果不存在则创建它。根据需要，文件将被覆盖。

临时文件被写入到 obj 文件夹。

对于具有多个指定框架的项目情况下，dotnet run 将首先选择.NET Core 框架。如果不存在，则会输出错误。若要指定其他框架，需要使用--framework 参数。

dotnet run 命令必须在项目上下文中使用。如果你想执行一个 DLL 作为替换，那么你应该使用不带任何参数的 dotnet 命令，就像下面的例子。

```
dotnet myapp.dll
```

直接运行生成后的 DLL。

dotnet run 选项：将参数分隔到正在运行的应用程序的参数的 dotnet run。在此参数后的所有参数均会被传递给正在运行的应用程序。

```
-f, --framework [FID]
```

运行指定框架标识符（FID）的应用程序。

```
-c, --configuration [Debug|Release]
```

发布时要使用的配置。默认值是“Debug”。

```
-p, --project [PATH]
```

指定要运行的项目。它可以是 project.json 文件或包含 project.json 文件的目录的路径。如果没有指定，则默认为当前目录。

可以使用 dotnet run –h 来查看命令帮助。

例如：

```
dotnet run
```

运行当前目录中的项目。

```
dotnet run --project /projects/proj/project.json
```

运行指定的项目。

```
dotnet run --configuration Release -- --help
```

运行当前目录中的项目。由于使用了参数一，因此上面的--help 将作为参数被传递到正在运行的应用程序。