Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™

.NET开发经典名著

Professional C# 6 and .NET Core 1.0

C#高级编程（第10版）

C# 6 & .NET Core 1.0

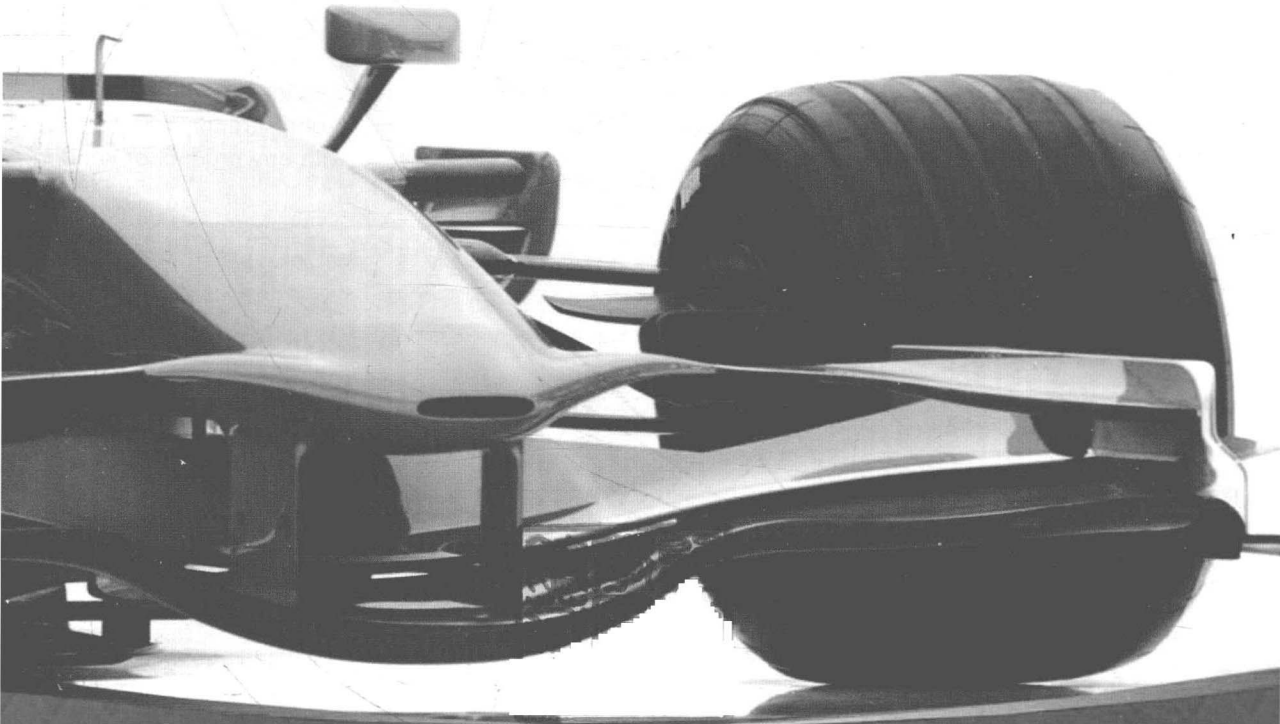
两度荣获全行业优秀畅销品种

[美] Christian Nagel 著
李 铭 译
黄 静 审校

清华大学出版社

.NET 开发经典名著

wrox
A Wiley Brand



C#高级编程(第 10 版)

C# 6 & .NET Core 1.0

[美] Christian Nagel 著
李 铭 译

清华大学出版社
北 京

Christian Nagel
Professional C# 6 and .NET Core 1.0
EISBN：978-1-119-09660-3

北京市版权局著作权合同登记号 图字：01-2016-5204

图书在版编目(CIP)数据

C#高级编程(第 10 版) C# 6 & .NET Core 1.0/(美) 克里斯琴·内格尔 (Christian Nagel) 著；李铭 译. —北京：清华大学出版社，2017
(.NET 开发经典名著)
书名原文：Professional C# 6.0 and .NET Core 1.0
ISBN 978-7-302-46196-8

Ⅰ. ①C… Ⅱ. ①克… ②李… Ⅲ. ①C 语言—程序设计 ②计算机网络—程序设计 Ⅳ. ①TP312C②TP393

中国版本图书馆 CIP 数据核字(2017)第 019996 号

责任编辑：王　军　于　平
装帧设计：牛静敏
责任校对：成凤进
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市中晟雅豪印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　印　　张：92.5　　插　　页：1　　字　　数：2482 千字
版　　次：2017 年 3 月第 1 版　　印　　次：2017 年 3 月第 1 次印刷
印　　数：1~5000
定　　价：168.00 元

产品编号：067811-01

译 者 序

C#是微软公司在2000年6月发布的一种新的编程语言，由Delphi语言的设计者Hejlsberg带领微软公司的开发团队开发，是一种安全的、稳定的、简单的、优雅的、由C和C++衍生出来的面向对象的编程语言。它在继承C和C++强大功能的同时，去掉了它们的一些复杂特性(例如没有宏以及不允许多重继承)。C#综合了Visual Basic简单的可视化操作和C++的高运行效率，以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程支持，成为.NET开发的首选语言。

自.NET推出以来，大约每两年就推出一个新的主要版本。从.NET Core的特性可以看出，自.NET第1版以来，这个技术在.NET历史上给.NET带来的变化最大：

- .NET Framework要求把开发过程中使用的.NET运行库版本安装到目标系统上。而在.NET Core 1.0中，框架(包括运行库)是与应用程序一起交付的。即使更新运行库，也不影响现有的应用程序。
- Visual Studio 2013附带着C# 5和.NET Framework 4.5。.NET Framework 4.5很大，有20 000多个类。只要添加新功能，.NET Framework就会变得越来越大。目前.NET Core的框架与.NET Framework 4.6一样巨大，但.NET Core 1.0以模块化的方法设计。该框架分成数量众多的NuGet包。根据应用程序决定需要什么包。
- NuGet包可以独立于.NET Framework发布，所以.NET Core可以很快更新，发布周期更短。
- .NET Core是开源的。
- .NET Core支持多个平台。新版本的.NET Core不仅运行在Windows上，还运行在Linux和Mac系统上。
- .NET Core可以编译为本地代码，得到更大的性能提升。

本书在第I部分阐述C#语言的背景知识。首先介绍C#的基本语法和数据类型，再介绍C#的面向对象功能，之后是C#中的一些高级编程主题。第II部分首先介绍Visual Studio 2015，接着论述C# 6新增的.NET编译器平台、应用程序的测试，之后介绍了独立于应用程序类型的.NET Core和Windows运行库主题。第III部分的主题是构建应用程序与XAML——Universal Windows应用程序和WPF。先介绍XAML的基础，给基于XAML的应用程序指定样式，再关注MVVM(Model-View-View Model)模式。在UWP应用程序和WPF应用程序的介绍性章节后，有两章的内容讨论UWP应用程序的具体特征，另外两章讨论WPF应用程序。本部分的最后，使用ClickOnce部署WPF应用程序。第IV部分阐述Web应用程序和服务，还包含关于ADO.NET的两章。先论述了ADO.NET和Entity Framework，接着介绍如何创建自己的Windows服务，然后学习ASP.NET的新版本ASP.NET Core 1.0，以及ASP.NET MVC 6的特点。接下来讨论ASP.NET Web API，使用ASP.NET技术WebHooks和SignalR的形式发布和订阅Web应用程序，这部分的最后讨论部署。

本书对上一版做了全面更新，使C#代码适用于最新版本的.NET Core 1.0。本书由.NET专家Christian Nagel编写，书中包含开发人员使用C#所需要的所有内容。本书适合希望提高编程技巧、有经验的C#程序员使用，也适用于刚开始使用C#的专业开发人员。

在这里要感谢清华大学出版社的编辑们，他们为本书的出版投入了巨大的热情并付出了很多心血。没有他们的帮助和鼓励，本书不可能顺利付梓。本书全部章节由李铭翻译、黄静审校。参与本

书翻译的还有孔祥亮、陈跃华、杜思明、熊晓磊、曹汉鸣、陶晓云、王通、方峻、李小凤、曹晓松、蒋晓冬、邱培强、洪妍、李亮辉、高娟妮、曹小震、陈笑。在此，一并表示感谢！

对于这本经典之作，译者在翻译过程中力求忠于原文，做到“信、达、雅”，但是鉴于译者水平有限，错误和失误在所难免。如有任何意见和建议，请不吝指正，感激不尽！

译　者

作 者 简 介

Christian Nagel 是 Visual Studio 和开发技术方向的 Microsoft MVP，担任微软开发技术代言人(Microsoft Regional Director)已经超过 15 年。Christian 是 thinktecture 的合伙人，并创办了 CN innovation，通过这两家公司为如何使用 Microsoft 平台开发解决方案提供培训和咨询。他拥有 25 年以上的软件开发经验。

Christian Nagel 最初在 Digital Equipment 公司通过 PDP 11 和 VAX / VMS 系统开始他的计算机职业生涯，接触过各种语言和平台。在 2000 年，.NET 只有一个技术概览版时，他就开始使用各种技术建立.NET 解决方案。目前，他主要使用几个 Microsoft Azure 服务产品，讲授 Universal Windows Platform 应用和 ASP.NET MVC 的开发。

在软件开发领域工作多年以后，Christian 仍然热爱学习和使用新技术，并通过多种形式教别人如何使用新技术。他的 Microsoft 技术知识非常渊博，编写了很多书，拥有微软认证培训师(MCT)和微软认证解决方案开发专家(MCSD)认证。Christian 在国际会议上(如 TechEd、BASTA !和 TechDays)经常发言。他创立了 INETA Europe 来支持.NET 用户组。他的联系网站是 www.cninnovation.com，其 Twitter 账号是@christiannagel。

技术编辑简介

István Novák 是 SoftwArt 的合伙人和首席技术顾问，SoftwArt 是匈牙利的一家小型 IT 咨询公司。István 是一名软件架构师和社区的传教士。在过去的 25 年里，他参加了 50 多个企业软件开发项目。2002 年，他在匈牙利与他人合作出版了第一本关于.NET 开发的图书。2007 年，他获得微软最有价值专家(MVP)头衔。2011 年，他成为微软开发技术代言人(Microsoft Regional Director)。István 与他人合作出版了 *Visual Studio 2010 and .NET 4 Six-in-One*(Wiley, 2010)和《Windows 8 应用开发入门经典》，独立编写了《Visual Studio 2010 LightSwitch 开发入门经典》。

István 从匈牙利的布达佩斯技术大学获得硕士学位和软件技术的博士学位。他与妻子和两个女儿居住在匈牙利的 Dunakeszi。他是一个充满激情的初级潜水员，常常在一年的任何季节，到红海潜水。

致　谢

我要感谢 Charlotte Kughen，他让本书的文本更具可读性。我经常在深夜写作，而.NET Core 在不断演变。Charlotte 为我提供了巨大的帮助，使我写出的文字易于阅读。可以肯定，Charlotte 现在精通编程的许多知识。也特别感谢 István Novák，他撰写了一些好书。

尽管.NET Core 的飞速发展和我在书中使用的临时构建还存在一些问题，但 István 向我挑战，改进了代码示例，让读者更容易理解。谢谢你们：Charlotte 和 István，你们让本书的质量上了一个大台阶。

我也要感谢 Kenyon Brown、Jim Minatel，以及 Wiley 出版社帮助出版本书的其他人。我还想感谢我的妻子和孩子，为了编写本书，我花费了大量的时间，包括晚上和周末，但你们很理解并支持我。Angela、Stephanie 和 Matthias，你们是我深爱的人。没有你们，本书不可能顺利出版。

前　言

对于开发人员，把 C#语言和.NET 描述为最重要的新技术一点都不夸张。.NET 提供了一种环境。在这种环境中，可以开发在 Windows 上运行的几乎所有应用程序。在 Windows 上运行的是.NET Framework 以前的版本，新版本.NET Core 1.0 不仅在 Windows 上运行，还在 Linux 和 Mac 系统上运行。C#是专门用于.NET 的编程语言。例如，使用 C#可以编写 Web 页面、Windows Presentation Foundation(WPF)应用程序、REST Web 服务、分布式应用程序的组件、数据库访问组件、传统的 Windows 桌面应用程序，以及可以联机/脱机运行的 Universal Windows Platform (UWP)应用程序。本书介绍.NET Core 1.0 和完整的.NET Framework，即.NET Framework 4.6。如果读者使用以前的版本编写代码，本书的一些章节就不适用。

在可能的情况下，本书的示例都使用了.NET Core 1.0。本书的代码在 Windows 系统上创建，但也可以在其他平台上运行。可能需要对示例进行较小的改变，才能使它们在 Linux 上运行。阅读第 1 章可以了解如何构建用于 Linux 平台的应用程序，什么程序不能在 Linux 上运行？WPF 应用程序仍然需要完整的.NET Framework，仅在 Windows 上运行。UWP 应用程序使用.NET Core，但还需要 Windows 运行库。这些应用程序也需要 Windows。这些 UI 技术都包含在本书的第III部分中。

那么，.NET 和 C#有什么优点？

0.1　.NET Core 的重要性

为了理解.NET Core 的重要性，就一定要考虑.NET Framework。.NET Framework 1.0 在 2002 年发布，此后大约每两年就推出一个新的主要版本。Visual Studio 2013 附带着 C# 5 和.NET 4.5。.NET Framework 4.5 十分巨大，有 20 000 多个类。

注意：第 1 章详细介绍了.NET Framework 和 C#的版本。

这个巨大的框架有什么问题？.NET Core 是如何解决的?

对于新的开发人员来说，掌握这个巨大的框架并不容易。其中保留了旧应用程序很重要的许多内容，但它们对新的应用程序并不重要。对于有经验的开发人员来说，在这些技术中选择一个最好的是不容易的。必须为 Web 应用程序选择使用 ASP.NET Web Forms 还是 ASP.NET MVC，为客户端应用程序选择使用 Windows Forms 和 WPF 还是 Universal Windows Platform，为数据访问选择 Entity Framework 还是 LINQ to SQL，为存储集合选择使用 Array List 还是

List<T>。这对于一些有经验的开发人员而言，选择是显而易见的，但对于大多数开发人员来说，选择并不是那么容易。刚开始接触.NET 的开发人员就更困难了。

.NET Core 基于较小的单元——小型 NuGet 包。Console 类只用于控制台应用程序。在.NET Framework 中，Console 类可用于 mscorlib，mscorlib 是每个.NET 应用程序都引用的程序集。使用.NET Core，必须显式地决定使用 NuGet 包 System.Console；否则，Console 类就不可用。

较小的包更容易摆脱框架的某些部分。如果需要给遗留应用程序使用旧的集合类，它们就可以通过 NuGet 包 System.Collections.NonGeneric 来使用。对于新的应用程序，可以定义能使用的软件包列表，System.Collections.NonGeneric 可以排除在这个列表之外。

如今，开发要快得多。在许多产品中，客户会收到产品的持续更新，而不是每两年接收一次新版本。甚至 Windows 10 都具备这么快的步伐。客户在每次更新时都收到较小的新特性，但收到新特性的速度更快。.NET Framework 目前的发布周期是两年，还不够快。一些技术，如 Entity Framework，已经绕过了这个问题，它可以通过 NuGet 包提供新功能，而 NuGet 包可以独立于.NET Framework 来发布。

更新较小的包，就允许更快的创新。.NET Core 基于许多小型 NuGet 包，所以更容易改变。.NET Core 和 ASP.NET 现在是开源的。.NET Core 的源代码在 http://www.github.com/dotnet 上，ASP.NET 的源代码在 http://www.github.com/aspnet 上。

发布.NET 时，Windows 在客户端和服务器上都有很大的市场份额。现在，世界更加碎片化。许多公司决定不通过 ASP.NET 运行服务器端代码，因为它不在 Linux 上运行。而 ASP.NET Core 1.0 和.NET Core 可以在 Linux 上运行。

.NET Core 独立于平台，支持 Windows、Linux 和 Mac 系统。对于客户端应用程序，可以在 iPhone 和 Android 上使用.NET 和 Xamarin。

.NET Framework 要求把开发过程中使用的.NET 运行库版本安装到目标系统上。基于客户需求，许多应用程序的开发都受到要使用的.NET Framework 版本的限制。这不仅是基于客户端的应用程序开发的问题，也是基于服务器的应用程序开发的问题。我们不得不转回旧的.NET 运行库版本，因为供应商不支持最新的版本。而有了.NET Core，运行库会和应用程序一起交付给客户。

建立 ASP.NET 时，与 Active Server Pages(ASP，使用运行在服务器上的 JavaScript 或 VBScript 代码建立)的兼容是一个重要的方面。现在不再需要了。建立 ASP.NET Web Forms 时，开发人员不需要知道任何关于 JavaScript 和 HTML 的内容，一切都可以用服务器端代码完成。现在，因为有了大量的 JavaScript 框架和 HTML 的增强，所以需要对 JavaScript 和 HTML 进行更多的控制。

在新版本的 ASP.NET 中，性能在框架体系结构中有很大的作用。只有真正需要的东西才施加性能影响。如果 Web 应用程序没有静态文件，就必须显式地决定是否使用它，否则就不会对它有性能影响。通过细粒度的控制，可以决定需要什么特性。

为了得到更大的性能提升，.NET Core 可以构建为本地代码。这不仅在 Windows 上是可能的，在 Linux 和 Mac 系统上也是可行的。这样，在程序启动时可以得到特别的性能改进，而且使用更少的内存。

现在，遗留的应用程序有一个问题。大多数应用程序都不能轻松地切换到.NET Core 上。完整的.NET Framework(仅运行在 Windows 上)也在进化。它进化的步伐没有.NET Core 那么大，

因为它是一个成熟的框架。在撰写本书时，发布了.NET 4.6.1，与之前的版本相比，其更新比较小。使用 Windows Forms 或 ASP.NET Web Forms 编写的应用程序仍然需要使用完整的框架，但它们可以利用.NET 4.6.1 的增强功能。通过.NET 4.6.1，还可以使用为.NET Core 建立的 NuGet 包。许多新的 NuGet 包采用便携的方式建立。通过 ASP.NET MVC 5 Web 应用程序，还可以决定改为运行在 ASP.NET Core 1.0 上的 ASP.NET MVC 6。ASP.NET Core 1.0 允许使用.NET Core 或.NET 4.6。这可以简化转换过程。然而，要在 Linux 上运行 ASP.NET MVC，则需要迁移 ASP.NET MVC 应用程序来使用.NET Core，但之前也不能在 Linux 上运行。

下面总结.NET Core 的一些特性：

- .NET Core 是开源的。
- NuGet 包较小，允许更快的创新。
- .NET Core 支持多个平台。
- .NET Core 可以编译为本地代码。
- ASP.NET 可以在 Windows 和 Linux 上运行。

从.NET Core 的特性可以看出，自.NET 第 1 版以来，这个技术在.NET 历史上给.NET 带来的变化最大。这是一个新的开始，我们可以用更快的步伐继续新的开发旅程。

0.2 C#的重要性

C#在 2002 年发布时，是一个用于.NET Framework 的开发语言。C#的设计思想来自于 C++、Java 和 Pascal。Anders Hejlsberg 从 Borland 来到微软公司，带来了开发 Delphi 语言的经验。Hejlsberg 在微软公司开发了 Java 的 Microsoft 版本 J++，之后创建了 C#。

C#一开始不仅作为一种面向对象的通用编程语言，而且是一种基于组件的编程语言，支持属性、事件、特性(注解)和构建程序集(包括元数据的二进制文件)。

随着时间的推移，C#增强了泛型、语言集成查询(Language Integrated Query，LINQ)、lambda 表达式、动态特性和更简单的异步编程。C#编程语言并不简单，因为它提供了很多特性，但它的实际使用的功能不断进化着。因此，C#不仅仅是面向对象或基于组件的语言，它还包括函数式编程的理念，开发各种应用程序的通用语言会实际应用这些理念。

0.3 C# 6 的新特性

在 C# 6 中有一个新的 C#编译器。它完成源代码的清理。现在自定义程序也可以使用编译器管道的特性，并使用 Visual Studio 的许多特性。

这个新的 C#编译器平台允许用许多新特性改进 C#。虽然这些特性都没有 LINQ 或 async 关键字的影响大，但许多改进都提高了开发人员的效率。C# 6 有哪些变化?

0.3.1 静态的 using 声明

静态的 using 声明允许调用静态方法时不使用类名。

C# 5

```
using System;
// etc.
Console.WriteLine("Hello, World!");
```

C# 6

```
using static System.Console;
// etc.
WriteLine("Hello, World");
```

using static 关键字参见第 2 章。

0.3.2 表达式体方法

表达式体方法只包括一个可以用 lambda 语法编写的语句：

C# 5

```
public bool IsSquare(Rectangle rect)
{
  return rect.Height == rect.Width;
}
```

C# 6

```
public bool IsSquare(Rectangle rect) => rect.Height == rect.Width;
```

表达式体方法参见第 3 章。

0.3.3 表达式体属性

与表达式体方法类似，只有 get 存取器的单行属性可以用 lambda 语法编写：

C# 5

```
public string FullName
{
  get
  {
    return FirstName + " " + LastName;
  }
}
```

C# 6

```
public string FullName => FirstName + " " + LastName;
```

表达式体属性参见第 3 章。

0.3.4 自动实现的属性初始化器

自动实现的属性可以用属性初始化器来初始化：

C# 5

```
public class Person
{
  public Person()
  {
    Age = 24;
  }
  public int Age {get; set;}
}
```

C# 6

```
public class Person
{
  public int Age {get; set;} = 42;
}
```

自动实现的属性初始化器参见第 3 章。

0.3.5 只读的自动属性

为了实现只读属性，C# 5 需要使用完整的属性语法；而在 C# 6 中，可以使用自动实现的属性：

C# 5

```
private readonly int _bookId;

public BookId
{
  get
  {
    return _bookId;
  }
}
```

C# 6

```
public BookId {get;}
```

只读的自动属性参见第 3 章。

0.3.6 nameof 运算符

使用新的 nameof 运算符，可以访问字段名、属性名、方法名或类型名。这样，在重构时，就不会遗漏名称的改变：

C# 5

```
public void Method(object o)
{
  if (o == null) throw new ArgumentNullException("o");
```

C# 6

```
public void Method(object o)
{
  if (o == null) throw new ArgumentNullException(nameof(o));
```

nameof 运算符参见第 8 章。

0.3.7　空值传播运算符

空值传播运算符简化了空值的检查：

C# 5

```
int? age = p == null ? null : p.Age;
```

C# 6

```
int? age = p?.Age;
```

新语法也有触发事件的优点：

C# 5

```
var handler = Event;
if (handler != null)
{
  handler(source, e);
}
```

C# 6

```
handler?.Invoke(source, e);
```

空值传播运算符参见第 8 章。

0.3.8　字符串插值

字符串插值删除了对 string.Format 的调用，它不在字符串中使用编号的格式占位符，占位符可以包含表达式：

C# 5

```
public override ToString()
{
  return string.Format("{0}, {1}", Title, Publisher);
```

```
}
```

C# 6

```
public override ToString() => $"{Title} {Publisher}";
```

与 C# 5 语法相比，C# 6 示例的代码减少了许多，不仅因为它使用了字符串插值，还使用了表达式体方法。

字符串插值还可以使用字符串格式，把它分配给 FormattableString 时会获得特殊的功能。字符串插值参见第 10 章。

0.3.9 字典初始化器

字典现在可以用字典初始化器来初始化，类似于集合初始化器。

C# 5

```
var dict = new Dictionary<int, string>();
dict.Add(3, "three");
dict.Add(7, "seven");
```

C# 6

```
var dict = new Dictionary<int, string>()
{
  [3] = "three",
  [7] = "seven"
};
```

字典初始化器参见第 11 章。

0.3.10 异常过滤器

异常过滤器允许在捕获异常之前过滤它们。

C# 5

```
try
{
  //etc.
}
catch (MyException ex)
{
  if (ex.ErrorCode != 405) throw;
  // etc.
}
```

C# 6

```
try
{
```

```
  //etc.
}
catch (MyException ex) when (ex.ErrorCode == 405)
{
  // etc.
}
```

新语法的一大优势是，它不仅减少了代码的长度，而且没有改变堆栈跟踪——在 C# 5 中会改变堆栈跟踪。异常过滤器参见第 14 章。

0.3.11　Catch 中的 await

await 现在可以在 catch 子句中使用。C# 5 需要一种变通方法：

C# 5

```
bool hasError = false;
string errorMessage = null;
try
{
  //etc.
}
catch (MyException ex)
{
  hasError = true;
  errorMessage = ex.Message;
}
if (hasError)
{
  await new MessageDialog().ShowAsync(errorMessage);
}
```

C# 6

```
try
{
  //etc.
}
catch (MyException ex)
{
  await new MessageDialog().ShowAsync(ex.Message);
}
```

这个功能不需要增强的 C#语法，现在就能工作。这个增强需要微软公司大量的投资才能实现，但是否使用这个平台真的没有关系。对用户来说，这意味着需要更少的代码——仅仅是比较两个版本。

注意：新的 C# 6 语言特性参见后面的章节，本书的所有章节都使用了新的C#语法。

0.4 UWP 的新内容

Windows 8 引入一种新的编程 API：Windows 运行库(Windows Runtime)。使用 Windows 运行库的应用程序可以通过 Microsoft Store 来使用，且有许多不同的名称。最初称它为 Metro 应用程序或 Metro 样式的应用程序，也称为 Modern 应用程序、Windows Store 应用程序(尽管它们也可以通过 PowerShell 脚本来安装，不使用商店)和 Universal 应用程序。这里可能遗漏了一些名字。如今，这些都只是 Windows 应用程序，运行在 UWP(Universal Windows Platform，通用 Windows 平台)上。

这些应用程序的理念是，让最终用户很容易通过 Microsoft 商店找到它们，提供便于触屏的环境，这个环境是现代的用户界面，看起来很干净、光滑，并允许流畅地交互，而且应用程序是可以信任的。更重要的是，已经了解 Windows 用户界面的用户应该被新的环境所吸引。

第 1 版的设计准则有诸多限制，有一些缺陷。如何在应用程序中寻找东西？许多用户在右边找不到工具栏，却发现它允许搜索许多应用程序。Windows 8.1 把搜索功能搬到桌面的一个搜索框中。同时，如果用户不在屏幕上从上扫到下或从下扫到上，就经常找不到位于顶部或底部的应用栏。

Windows 10 使设计更加开放。可以使用对应用程序有用的东西，在用户界面上做出决定，因为它与用户和应用程序最匹配。当然，它仍然最适合创建出好看的、光滑的、流畅的设计。最好让用户与应用程序愉快地交互，确定如何完成任务应该不是很难。

新的 Windows 运行库是 Windows 运行库 3.0，它基于以前的版本，定义了 XAML 用户界面，实现了应用程序的生命周期，支持后台功能，在应用程序之间共享数据等。事实上，新版本的运行库在所有区域都提供了更多的功能。

Windows 应用程序现在使用.NET Core。通过 NuGet 包与 Windows 应用程序可以使用相同的.NET 库。最后，本地代码编译后，应用程序启动更快，消耗的内存更少。

与提供的附加功能相比，可能更重要的是现在可用的普遍性。Visual Studio 2013 的第一次更新为 Windows 8 应用程序包括了一个新的项目类型：通用(Universal)应用程序。在这里，通用应用程序用 3 个项目实现：一个项目用于 Windows 应用程序，一个项目用于 Windows Phone 应用程序，另一个是共享的代码项目。甚至可以在这些平台之间共享 XAML 代码。新的通用项目模板包括一个项目。相同的二进制代码不仅可以用于 Windows 和 Windows Phone，还可以用于 Xbox、物联网(Internet of Things，IoT)设备和 HoloLens 等。当然，这些不同的平台所提供的功能不可能用于所有地方，但是使用这个不同的功能，仍然可以创建二进制图像，在每个 Windows 10 设备上运行。

0.5　编写和运行 C#代码的环境

.NET Core运行在Windows、Linux和Mac操作系统上。使用Visual Studio Code(https://code.visualstudio.com)，可以在任何操作系统上创建和构建程序。最好用的开发工具是 Visual Studio 2015，也是本书使用的工具。可以使用 Visual Studio Community 2015 版(https://www.visualstudio.com)，但本书介绍的一些功能只有 Visual Studio 的企业版提供。需要企业版时会提到。Visual Studio 2015 需要 Windows 操作系统，要求使用 Windows 8.1 或更高版本。

要构建和运行本书中的 WPF 应用程序，需要 Windows 平台。Windows 7 仍支持运行 WPF 应用程序。

要构建通用的 Windows 应用程序，可以使用 Windows 8.1 和 Visual Studio，但要测试和运行这些应用程序，就需要 Windows 10 设备。

0.6　本书的内容

本书首先在第 1 章介绍.NET 的整体体系结构，给出编写托管代码所需要的背景知识。此后概述不同的应用程序类型，学习如何用新的开发环境 CLI 编译程序。之后本书分几部分介绍C#语言及其在各个领域中的应用。

第 I 部分——C#语言

该部分给出 C#语言的良好背景知识。尽管这一部分假定读者是有经验的编程人员，但它没有假设读者拥有任何特定语言的知识。首先介绍 C#的基本语法和数据类型，再介绍 C#的面向对象特性，之后介绍 C#中的一些高级编程主题，如委托、lambda 表达式、语言集成查询(LINQ)、反射和异步编程。

第 II 部分——.NET Core 与 Windows Runtime

该部分首先介绍全世界 C#开发人员都使用的主要 IDE：Visual Studio 2015。第 17 章学习 Visual Studio 企业版可用的工具。

第 18 章还要学习 C#编译器的工作原理，以及如何使用.NET 编译器平台以编程方式修改代码。

用 C#代码创建功能时，不要跳过创建单元测试的步骤。一开始需要更多的时间，但随着时间的推移，添加功能和维护代码时，就会看到其优势。第 19 章介绍如何创建单元测试、网络测试和编码的 UI 测试。

第 20～28 章介绍独立于应用程序类型的.NET Core 和 Windows 运行库。第 20 章学习如何从应用程序中写出也可以在生产环境中使用的诊断信息。第 21 章和第 22 章介绍了使用任务并行库(Task Parallel Library，TPL)进行并行编程，以及用于同步的各种对象。第 23 章学习如何访问文件系统，读取文件和目录，了解如何使用 System.IO 名称空间中的流和 Windows 运行库中的流来编写 Windows 应用程序。第 24 章利用流来了解安全性，以及如何加密数据，允许进行安全的转换。还将学习使用套接字和使用更高级别的抽象(如 HttpClient，见第 25 章)