HZ BOOKS 华章IT

EFFECTIVE 系列丛书

Pearson

经典畅销书全新升级，世界知名C#专家Bill Wagner倾力撰写，C#程序员必备参考

紧贴C#语言的设计理念，既从正面阐释如何编写高效代码，又从反面入手指出容易出错之处，涵盖C#语言的各个方面

Effective C#

50 Specific Ways to Improve Your C#

Third Edition

Effective C#

改善C#代码的50个有效方法

（原书第3版）

[美] 比尔·瓦格纳（Bill Wagner） 著

爱飞翔 译

机械工业出版社
China Machine Press

EFFECTIVE
系列丛书

Effective C#
50 Specific Ways to Improve Your C#
Third Edition

Effective C#

改善C#代码的50个有效方法

（原书第3版）

[美] 比尔·瓦格纳（Bill Wagner） 著
爱飞翔 译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Effective C#：改善C#代码的50个有效方法（原书第3版）/（美）比尔·瓦格纳（Bill Wagner）著；爱飞翔译．—北京：机械工业出版社，2018.4
（Effective系列丛书）
书名原文：Effective C#: 50 Specific Ways to Improve Your C#, Third Edition

ISBN 978-7-111-59719-3

I. E… II. ①比… ②爱… III. C语言—程序设计 IV. TP312.8

中国版本图书馆CIP数据核字（2018）第080957号

本书版权登记号：图字 01-2017-0904

Effective C#：改善C#代码的50个有效方法（原书第3版）

出版发行：机械工业出版社（北京市西城区百万庄大街22号 邮政编码：100037）
责任编辑：张志铭 责任校对：殷 虹
印 刷：三河市宏图印务有限公司 版 次：2018年5月第1版第1次印刷
开 本：186mm×240mm 1/16 印 张：16.75
书 号：ISBN 978-7-111-59719-3 定 价：79.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzit@hzbook.com

Praise 本书赞誉

“要想成为高效率的 .NET 开发者，就必须透彻地理解自己所选的编程语言。Wagner 先生的这本书通过合理的论证，提出了一些见解，能够帮助读者更为透彻地了解 C# 语言。无论是新手还是具备数年经验的人，都可以从这本书中学到新的知识。”

——Jason Bock，Magenic 首席顾问

“如果你和我一样把这本书看完，那么你就可以收集到很多 C# 语言的技巧，这些技巧能够令你更好地发挥自身能力，从而成为更加专业的开发者。在综述 C# 语言技巧的书里面，这本书很可能是最棒的，Bill Wagner 这次所写的新版《Effective C#》依然是杰作，没有令我失望。”

——Bill Craun，Ambassador Solutions 首席顾问

“想要构建高性能与高扩展性应用程序的开发者都应该阅读本书。这本书写得很好，作者 Bill 把相当复杂的问题拆分成读者容易理解、容易吸收的小知识点。”

——Josh Holmes，微软架构布道师

“这本书依然很好，浓缩了许多对 C# 开发者极为有用的技巧。每天学习一项技巧，50 天之后，C# 开发水平就会大有长进。”

——Claudio Lassala，EPS Software / CODE Magazine 首席开发者

“这本书包含很多 C# 语言的知识以及作者对这门语言的理解。Bill 揭示了 .NET 运行时平台的底层机制，使读者能够明白构建于该平台上的 C# 代码是如何运行的。此外，他还告诉大家应该怎样写出清晰流畅、易于理解的代码。书中包含丰富的技巧与深刻的见解……每一位 C# 开发者都应该阅读。”

——Brian Noyes，IDesign Inc.（www.idesign.net）首席架构师

“这是一本每一位 C# 开发者必读的书，里面给出了很多实用的编程建议。”

——Shawn Wildermuth，微软 MVP（C#）、写作者、培训者、演讲者

“Bill Wagner 在这本书中切实地阐述了怎样使用 C# 语言中最为重要的特性。他具备丰富的知识与流畅的表达能力，可以清楚地讲解 C# 语言的新功能，使读者明白怎样运用这些功能写出精练而易于维护的代码。”

——Charlie Calvert，微软 C# 社区项目经理

The Translator's Words 译 者 序

自经典畅销书《 Effective C++》面世以来，Effective 图书系列成了软件开发界的传统图书。很多语言都有对应的 Effective 书籍，这些书会把该语言中的中、高级技巧按照其所属门类系统地组织起来，使这些技巧既能自成一体，又能与同一门类中的其他技巧相互联系，从而形成一套知识体系。有了这样的体系，读者就可以把工作与学习中所总结的心得以及所联想到的思路安排进来，从而清晰地感知它们在整个体系中所处的位置。

与该书系的其他作品一样，这本《 Effective C# 》也表现出了这种风格。作者把 C# 语言在各个方面的用法系统地整理出来。除了从正面讲述应该怎样编写高效的代码之外，还从反面入手，告诉大家 C# 及 .NET 中都有哪些可能出错或遭到误用的地方。作者不仅指出了问题，而且详细解释了这些问题的产生原理及应对方案。考虑到作者在 .NET 及 C# 开发界的经历以及在 Microsoft 公司与其他组织中所从事的工作，这些讲解是很有分量的。

虽说这本书是整个 Effective 系列的一部分，但并没有单纯按照某个固定的套路去复刻，而是有着自身的创见。书中的很多建议都是从系统自带的类库中寻找灵感，并提倡将相关的设计模式运用到自己所要编写的类库或应用程序上，这或许能够提醒大家：编写程序库与编写客户代码时所用的思路未必是毫无关联的，而是有可能在某种意义上是相通的。作者针对程序库的设计者所提出的一些建议其实同样适用于客户端的开发者，反过来说，客户端的开发者调用程序库的方式也可以给库的设计者提供参考，提醒他们多考虑用户的实际用法，而不是一味固守某种教条。

在具体实现层面上，作者的思路同样开阔。他没有直接重复业界已有的编程习惯，

而是辨析了这些写法的优点及缺点，并建议大家要在适当的情境中合理地加以运用，而不能过于盲目，同时还告诉读者，应该理解并熟悉 C# 语言所添加的新特性，以改掉从前那些不好的或已经过时的习惯。

总之，书中的 50 条建议都是紧贴着 C# 语言自身的设计理念而写的，在介绍新特性以及与其他语言相对比的过程中，也充分考虑到了程序库设计者与客户端开发者实际使用该语言的方式。这种在沿承中有所创新的做法，令语言本身及其用户都显得更有活力。

翻译本书的过程中，我得到了机械工业出版社华章公司诸位编辑与工作人员的帮助，在此深表感谢。

书中的术语参考了 Microsoft 的语言门户网站（www.microsoft.com/Language/zh-cn/Search.aspx）以及其他一些技术文章，书末附有“中英文词汇对照表”，以供查阅。

由于译者水平有限，错误与疏漏之处，请大家发邮件至 eastarstormlee@gmail.com，或访问 github.com/jeffreybaoshenlee/ecs3-errata/issues 留言，给我以批评和指教。

爱飞翔

Preface 前　言

本书第 1 版于 2004 年出版，到了 2016 年，C# 开发社群的情况已经有了很大的变化。使用这门语言编写程序的人越来越多，很多人现在都把 C# 当作首选的工作语言，并且不会再按照使用其他语言时所形成的那些习惯来使用这门语言。此外，C# 开发者所具备的经验各不相同，从刚毕业的学生到拥有数十年经验的专业开发者，都有人在用 C# 写程序，而且 C# 所支持的平台也比原来更加广泛。你可以用它架设服务器或制作网站，也可以为各种操作系统开发桌面版本或移动版本的应用程序。

这次升级的第 3 版既考虑到 C# 语言本身的变化，也考虑到使用这门语言的人（或者说 C# 开发社群）所发生的变化。笔者并不打算讲述 C# 语言的演变历程，而是关注怎样用好当前版本的 C# 语言。旧版的某些条目已经与当今的 C# 语言或 C# 应用程序脱节了，这些内容不会出现在新版中。新版中会添加一些条目，以讲述 C# 语言的新特性与 .NET 框架的新功能，这些内容是从软件产品的开发过程中提炼出来的，许多 C# 开发者采用这些特性开发了多个版本的软件。看过《 More Effective C# 》第 1 版的读者稍后可能会发现，那本书里的某些内容已经移到了本书中。在本书第 3 版中，笔者重新编排《 More Effective C# 》的内容，删除了原有的许多条目，以便在那本书的第 2 版中添加其他一些条目。总之，这本书里的 50 个条目都是一些编程建议，可以帮助你更为高效地使用 C# 语言，从而成为更加专业的开发者。

本书预设的语境是 6.0 版本的 C#，然而笔者并不会把该版本的功能全都拿出来讲。与 Effective Software Development 系列的其他书一样，这本书所关注的也是怎样用语言特性来解决日常工作中可能遇到的问题，并提供实用的建议。在 C# 6.0 版的这些特性

中，笔者会特意挑出一些来讲，因为其中的某些特性能够使开发者以更好的方式来编写常用的代码。网上搜到的写法可能是针对许多年前的 C# 版本而写的，有了新版 C# 所引入的特性之后，开发者就可以用更好的写法来完成那些任务了，对于此类情况，笔者会专门指出。

书中的很多建议都可以用 Roslyn 平台的 Analyzer 及 Code Fix 加以体现，从而验证开发者所写的代码是否符合这些建议。笔者把相关的 Analyzer 放在了这里：https://github.com/BillWagner/EffectiveCSharpAnalyzers。你可以提交 issue，以表达自己的看法，或是发送 pull request 为项目添加新的内容。

读者对象

本书面向的是那些使用 C# 来完成日常工作的职业开发者。由于本书假设读者已经熟悉了 C# 的语法及语言特性，因此，并不会按部就班地讲解这些特性，而是会告诉你应该怎样把当前这一版 C# 语言所拥有的各种特性融入日常的开发工作中。

除了要熟悉语言本身的特性之外，还应该对 CLR（Common Language Runtime，公共语言运行时）及 JIT（Just-In-Time，即时）编译器有所了解。

内容提要

有一些语言结构是每次写 C# 程序时几乎都会用到的，这些常见的写法出现在本书的第 1 章中，它们是开发者手头必备的工具，无论创建什么样的类型与算法，都离不开这些工具。

尽管 C# 程序运行在托管环境中，但并不是说开发者什么事情都不用操心。要想令程序的性能满足需求，就必须编写出能够与托管环境相协调的代码，这不是单靠性能测试与性能调整就可以实现的。因此，第 2 章会介绍一些设计习惯，告诉你应该怎样把代码写得与托管环境相协调。以良好的设计风格为基础，可以更加有效地优化细节问题。

自 C# 2.0 以来所引入的很多新技术都是以泛型为依托的。第 3 章讲解怎样用泛型取代 `System.Object` 以及强制类型转换，然后，笔者会讨论一些高级技术，例如约束、

泛型特化、方法约束以及向后兼容等。读完本章之后，你会学到很多泛型技巧，从而能够更加顺畅地表达出自己的设计思路。

第 4 章会讲解 LINQ、查询语法以及与之相关的语言特性。你会了解到在哪些情况下应该运用扩展方法把协定与实现相分离，还会学到应该怎样有效地使用闭包以及如何编写匿名类型。此外，笔者还会解释编译器怎样把查询关键字映射成方法调用、如何区分委托与表达式树以及必要时怎样在二者之间转换，以及如何对查询做出转义以获取纯量形式的结果。

第 5 章会指引你把 C# 程序中的异常与错误处理好。笔者要讲解怎样确保程序中的错误能够得到适当的汇报，以及如何令程序的状态在出错之后依然保持稳定，甚至与出错之前一样。此外，你还会学到怎样给使用代码的人提供便利，令他们能够更加顺畅地调试你所编写的程序。

代码约定

要想把范例代码印在书中，就必须在保持清晰的前提下顾及篇幅。笔者尽量把代码写得简短，以凸显其中最关键的部分，并把类或方法中的其他部分省掉。有时为了节省篇幅，还会把错误恢复代码也省掉。public 方法自然应该验证其参数以及外界输入给它的数据，但考虑到篇幅，笔者通常会把这些代码去掉。此外，很多复杂的算法还会对方法调用做出核查，而且会包含 `try/finally` 子句，这些代码也因同样的理由而删去。

常见的命名空间就不再写出了。你可以认为每一份范例代码前面都写有下面几条 `using` 语句：

```
using System;
using static System.Console;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

提供反馈意见

笔者与本书的审阅者都尽力确保书中的内容正确无误，尽管如此，本书与范例代码里面可能还是会有一些错误，读者如果发现某个地方写错了，请发邮件至 bill@

thebillwagner.com，或通过 Twitter 号码 @billwagner 联系我。勘误表将会发布至 http://thebillwagner.com/Resources/Effectivecs。书中的很多条目是笔者在与其他 C# 开发者通过电子邮件及 Twitter 讨论之后写出的，读者若对这些编程建议有疑问或意见，也请联系笔者。更为一般的话题可参见笔者博客：http://thebillwagner.com/blog。

致谢

我要感谢为本书做出贡献的诸多人士。很荣幸能在这些年里与大家一起使用 C# 语言。C# Insiders 邮件列表中的每位朋友（无论身处 Microsoft 公司之内或之外）都为本书提供了创意，并且愿意与我交流，使我能把这本书写得更好。

必须特别感谢下面这几位 C# 开发者：Jon Skeet、Dustin Campbell、Kevin Pilch-Bisson、Jared Parsons、Scott Allen 以及 Mads Torgersen。感谢你们与我沟通、向我提供意见，并将其转变为具体的成果。这一版的很多新想法都是根据诸位的意见而形成的。

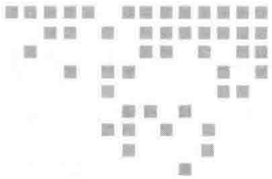
这一版的技术评审团队同样很出色。Jason Bock、Mark Michaelis 与 Eric Lippert 仔细阅读了文稿与范例代码，以确保读者能拿到一本优质的书籍。他们的水平相当高，不仅全面而彻底地审阅了本书，而且还提供了一些建议，帮助我把其中的很多话题解释得更为清楚。

我与 Addison-Wesley 出版社的编辑团队合作得相当愉快。Trina Macdonald 是一位优秀的编辑，总能督促我把书写好。Mark Renfro 与 Olivia Basegio 是她的得力帮手，我依靠他们完成了很多工作，这本书的定稿能够达到现在这样的质量，与他们的努力有很大关系，从头到尾的每一页内容都是如此。Curt Johnson 致力于发售这本技术图书，无论是哪种格式都有他的一份心力在里面。

感谢 Scott Meyers 再度将本书收入 Effective 书系，他阅读了整部文稿，并提出了一些改进建议。Meyers 虽然不是做 C# 的，但却有着丰富的软件开发经验，能够把文稿中没有解释清楚的地方找出来，而且能指出其中有哪些技巧还不足以总结成心得推荐给大家使用。他的意见，给我带来了很大的帮助。

感谢家人留出时间，令我可以写完这本书。我花了很长时间撰写书稿并制作范例代码，妻子 Marlene 总是给予我支持。有她的鼓励，我才能把这本书和其他的书写好。

Contents 目　录



第 1 章 Chapter 1

C# 语言的编程习惯

能用的东西为什么要改？因为改了之后效果更好。开发者换用其他工具或语言来编程也是这个道理，因为换了之后工作效率更高。如果不肯改变现有的习惯，那么就体会不到新技术的好处，但如果这种新的技术与你熟悉的技术看上去很像，那么改起来就特别困难。例如 C# 语言就与 C++ 或 Java 语言相似，由于它们都用一对花括号来表示代码块，因此，开发者即便切换到了 C# 语言，也总是会把使用那两门语言时所养成的习惯直接带过来，这样做其实并不能发挥出 C# 的优势。这门语言的首个商用版本发布于 2001 年，经过这些年的演变，当前这一版 C# 语言与 C++ 或 Java 之间的差别已经远远大于那个年代。如果你是从其他语言转入 C# 的，那么需要学习 C# 语言自己的编程习惯，使得这门语言能够促进你的工作，而不是阻碍你的工作。本章会提醒大家把那些与 C# 编程风格不符的做法改掉，并培养正确的编程习惯。

第 1 条：优先使用隐式类型的局部变量

隐式类型的局部变量是为了支持匿名类型机制而加入 C# 语言的。之所以要添加这种机制，还有一个原因在于：某些查询操作所获得的结果是 `IQueryable<T>`，而其他一些则返回 `IEnumerable<T>`。如果硬要把前者当成后者来对待，那就无法使用由 `IQueryProvider` 所提供的很多增强功能了（参见第 42 条）。用 `var` 来声明变量而不

指明其类型，可以令开发者把注意力更多地集中在名称上面，从而更好地了解其含义。例如，jobsQueuedByRegion 这个变量名本身就已经把该变量的用途说清楚了，即便将它的类型 Dictionary<int, Queue<string>> 写出来，也不会给人提供多少帮助。

对于很多局部变量，笔者都喜欢用 var 来声明，因为这可以令人把注意力放在最为重要的部分，也就是变量的语义上面，而不用分心去考虑其类型。如果代码使用了不合适的类型，那么编译器会提醒你，而不用你提前去操心。变量的类型安全与开发者有没有把变量的类型写出来并不是同一回事。在很多场合，即便你费心去区分 IQueryable 与 IEnumerable 之间的差别，开发者也无法由此获得有用的信息。如果你非要把类型明确地告诉编译器，那么有时可能会改变代码的执行方式（参见第 42 条）。在很多情况下，完全可以使用 var 来声明隐式类型的局部变量，因为编译器会自动选择合适的类型。但是不能滥用这种方式，因为那样会令代码难于阅读，甚至可能产生微妙的类型转换 bug。

局部变量的类型推断机制并不影响 C# 的静态类型检查。这是为什么呢？首先必须了解局部变量的类型推断不等于动态类型检查。用 var 来声明的变量不是动态变量，它的类型会根据赋值符号右侧那个值的类型来确定。var 的意义在于，你不用把变量的类型告诉编译器，编译器会替你判断。

笔者现在从代码是否易读的角度讲解隐式类型的局部变量所带来的好处和问题。其实在很多情况下，局部变量的类型完全可以从初始化语句中看出来：

```
var foo = new MyType();
```

懂 C# 的开发者只要看到这条语句，立刻就能明白 foo 变量是什么类型。此外，如果用工厂方法的返回值来初始化某个变量，那么其类型通常也是显而易见的：

```
var thing = AccountFactory.CreateSavingsAccount();
```

某些方法的名称没有清晰地指出返回值的类型，例如

```
var result = someObject.DoSomeWork(anotherParameter);
```

这个例子当然是笔者刻意构造的，大家在编写代码的时候应该把方法的名字起好，使得调用方可以据此推断出返回值的类型。对于刚才那个例子来说，其实只需要修改变

量的名称，就能令代码变得清晰：

```
var HighestSellingProduct = someObject
    .DoSomeWork(anotherParameter);
```

尽管方法名本身没有指出返回值的类型，但是像这样修改之后，很多开发者就可以通过变量的名称推断出该变量的类型应该是 Product。

HighestSellingProduct 变量的真实类型当然要由 DoSomeWork 方法的签名来决定，因此，它的类型可能并不是 Product 本身，而是继承自 Product 的类，或是 Product 所实现的接口。总之，编译器会根据 DoSomeWork 方法的签名来认定 HighestSellingProduct 变量的类型。无论它在运行期的实际类型是不是 Product，只要没有明确执行类型转换操作，那么一律以编译器判断的类型为准。

用 var 来声明变量可能会令阅读代码的人感到困惑。比方说，如果像刚才那样用方法的返回值来给这样的变量做初始化，那么就会造成此类问题。查看代码的人会按照自己的理解来认定这个变量的类型，而他所认定的类型可能恰好与变量在运行期的真实类型相符。但是编译器却不会像人那样去考虑该对象在运行期的类型，而是会根据声明判定其在编译期的类型。如果声明变量的时候直接指出它的类型，那么编译器与其他开发者就都会看到这个类型，并且会以该类型为准，反之，若用 var 来声明，则编译器会自行推断其类型，而其他开发者却看不到编译器所推断出的类型。因此，他们所认定的类型可能与编译器推断出的类型不符。这会令代码在维护过程中遭到错误地修改，并产生一些本来可以避免的 bug。

如果隐式类型的局部变量的类型是 C# 内置的数值类型，那么还会产生另外一些问题，因为在使用这样的数值时，可能会触发各种形式的转换。有些转换是宽化转换（widening conversion），这种转换肯定是安全的，例如从 float 到 double 就是如此，但还有一些转换是窄化转换（narrowing conversion），这种转换会令精确度下降，例如从 long 到 int 的转换就会产生这个问题。如果明确地写出数值变量所应具备的类型，那么就可以更好地加以控制，而且编译器也会把有可能因转换而丢失精度的地方给你指出来。

现在看这段代码：