TURING 图灵程序设计丛书 微软技术系列

Addison Wesley

More Effective C#中文版

改善C#程序的50个具体办法

[美] Bill Wagner 著
陈黎夫 译

MVP Microsoft Most Valuable Professional

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 微软技术系列

More Effective C#中文版

改善C#程序的50个具体办法

[美] Bill Wagner 著
陈黎夫 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

More Effective C# 中文版：改善 C# 程序的 50 个具体办法 /（美）瓦格纳（Wagner, B.）著；陈黎夫译．—北京：人民邮电出版社，2010.1
（图灵程序设计丛书）
书名原文：More Effective C#: 50 Specific Ways to Improve Your C#
ISBN 978-7-115-21570-3

Ⅰ.①M… Ⅱ.①瓦…②陈… Ⅲ.①C 语言－程序设计 Ⅳ.①TP312

中国版本图书馆CIP数据核字（2009）第178752号

内 容 提 要

本书延续了 *Effective* 系列图书的风格，针对 C# 2.0 和 C# 3.0 中添加的新特性给出了实用的建议。书中的 50 个条目自成一体且又丝丝相扣，这些条目按照泛型、多线程开发、C# 设计模式、C# 3.0 语言增强、LINQ 以及杂项等主题分为 6 类，将特定的代码和深入的分析有机地结合了起来，能够帮助你以最佳的方式从 C# 1.x 切换至 C# 3.0。当你通读全书之后，会发现不只得到了一条条独立的建议，还学到了如何以优雅的方式用 C# 进行程序设计。

本书适合具有 C# 编程经验的 .NET 开发人员阅读。

图灵程序设计丛书

More Effective C#中文版：改善C#程序的50个具体办法

◆ 著 [美] Bill Wagner
译 陈黎夫
责任编辑 傅志红
执行编辑 陈兴璐

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：19.25
字数：315千字 2010年1月第1版
印数：1－3 000册 2010年1月北京第1次印刷
著作权合同登记号 图字：01-2009-5718号
ISBN 978-7-115-21570-3

定价：49.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

本书赞誉

“本书就像一盏明灯，照亮了C# 3.0中很多不为人知的角落。它不仅介绍了如何做，还解释了这样做的原因，让读者学习到很多经过实践检验的语言新特性用法，包括LINQ、泛型以及多线程等。若你确实需要使用C#语言开发程序，那么本书是必不可少的。”

——Bill Craun，Ambassador Solutions公司首席咨询师

“本书创造了一个让你能够和Bill Wagner并肩思考、工作的机会。Bill在本书中充分展示了他在C#上的造诣，给出了很多编程方面的有效建议，值得每个Visual C#开发者去学习。本书并没有停留在泛泛描述C#语法上，而是真正教会你使用C#语言。”

——Peter Ritchie，微软公司MVP: Visual C#

“本书是Bill Wagner前一本书的很好续作。其中对C# 3.0和LINQ的介绍非常及时！”

——Tomas Restrepo，微软公司MVP: Visual C++, .NET和Biztalk Server

“作为C#设计组的成员，很少有书能够让我从中学到什么新东西。本书则是个例外。它很好地将特定的代码和深入的分析结合了起来。本书提供了一系列非常有用的建议。当你通读全书之后，会发现不只得到了一条条独立的建议，还学到了如何能够以优雅的方式用C#进行程序设计。虽然你可以根据需要挑选

某个条目阅读，但我仍强烈建议你通读全书——至少不要跳过每一章前面的介绍部分。这一富有洞察力的、充满远见的内容会对你日后的C#学习给予很大的启发和帮助。”

——Mads Torgersen，微软公司Visual C#项目经理

“Bill Wagner为C#开发人员撰写了一本精彩绝伦的图书，其中介绍了大量C#最佳实践。本书再次确立了他在C#社区中的重要地位。我们大都知道如何使用C#，同时也期待有人能给出提高的建议，让我们更上一层楼。若想成为C#开发的顶级高手，那么没有什么资料要比Bill Wagner的这本书更好了。Bill非常智慧、深刻，富有经验和技巧。若能将这本书中给出的建议应用到你的代码中，定会大大提高你的工作质量。”

——Charlie Calvert，微软公司Visual C#社区项目经理

前　言

自从Anders Hejlsberg在2005年专业开发者大会上第一次演示LINQ（Language-Integrated Query，语言集成查询）以来，C#编程世界就被彻底地改变了。LINQ的出现为C#语言带来了几个令人着迷的新特性：扩展方法、局部变量类型推断、lambda表达式、匿名类型、对象初始化器以及集合初始化器。C# 2.0也为LINQ的出现打下了坚实的基础，添加了包括泛型、迭代器、静态类、可空类型、属性访问器权限以及匿名委托等新功能。但即使在非LINQ的使用环境中，这些语言特性也有大显身手之处——毕竟还有很多非数据访问的编程任务。

本书针对C# 2.0和C# 3.0中添加的新特性给出了实用的建议，也包含了在我的上本图书*Effective C#: 50 Specific Ways to Improve Your C#*（Addison-Wesley，2004）[①]中没有提到的高级特性。本书中的条目主要针对那些正在使用C# 3.0编写程序的开发人员。书中着重介绍了泛型技术，这是C# 2.0和C# 3.0中众多新特性的基石。本书并没有将条目按照语言特性组织起来，而是根据新特性最善于解决的编程问题来编排条目的。

与Effective Software Development丛书中的其他图书一样，本书中的每个条目的建议都自成一体，针对使用C#时的某个特定问题。这些条目能够帮助你以最佳的方式从C# 1.x切换至C# 3.0。

泛型是C# 3.0中所有新特性的基础。虽然只有第1章专门介绍了泛型，但你会发现泛型技术也是几乎每个条目中的不可分割的一部分。在阅读完本书之后，你定会熟悉并喜欢上泛型以及元编程（metaprogramming）概念。

① 中译本《Effective C#中文版》，李建忠译，人民邮电出版社2007年出版。——编者注

当然，本书中的很大一部分篇幅都用来讨论了如何使用C# 3.0以及LINQ查询语法。不过不管你是否将其用在查询数据源上，C# 3.0所添加的众多语言新特性均非常有用。语言上的改变非常巨大，LINQ又是引起改变的主要原因，它们都需要专门的章进行介绍。LINQ和C# 3.0将深刻影响你编写C#代码的方式，而本书则会让这个过渡更加平稳简单。

读者对象

本书是为那些使用C#进行软件设计的专业开发人员所编写的。本书假定你已对C# 2.0和C# 3.0有了一定的了解。Scott Meyers告诉我说，*Effective*系列图书应该作为开发人员针对某一主题学习的进阶参考资料。因此，本书并没有泛泛介绍任何有关语言的新特性，而是着重阐述如何将这些新特性应用到正在开发的软件中。你将会学到，何时该在开发中使用这些新语言特性，以及如何避免误用所造成的问题。

除了对C#语言新特性有一定了解之外，你还应该对组成.NET Framework的主要组件有所了解，包括.NET CLR（Common Language Runtime）、.NET BCL（Base Class Library）以及JIT（Just In Time）编译器等。本书并没有涉及.NET 3.0组件，例如WCF（Windows Communication Foundation）、WPF（Windows Presentation Foundation）以及WF（Windows Workflow Foundation）等。不过其中介绍的各种用法同样适用于上述各个组件以及其他的.NET Framework组件。

内容介绍

泛型是自C# 1.1以来所有C#语言新功能的基础。第1章首先介绍了如何用泛型替代`System.Object`和类型强制转换，随后讨论了一些高级主题，包括约束、泛型的特化、方法约束以及向后兼容性等。其中介绍的几种技术都使用泛型让你更清晰地表达出设计意图。

多核处理器已经普及，同时计算机的核心数量也在不停增加。这也就意味着每个C#开发人员都需要对C#多线程编程有着足够的理解。即便一章的篇幅

不足以让你变成专家，但第2章中的建议仍旧会对你开发多线程应用程序有所帮助。

第3章介绍了如何用C#语言实现常用的设计。其中将介绍用C#语言特性表达意图的最好方法。你将学到如何使用延迟求值，如何创建可组合的接口，以及如何避免由于公共接口中各种语言元素所带来的混乱。

第4章讨论了如何借助C# 3.0的语言增强来解决编程中遇到的困难，包括如何使用扩展方法来分离契约和实现，如何有效地使用C#闭包，以及如何使用匿名类型等内容。

第5章介绍了LINQ及其查询语法，包含了编译器如何将查询关键字映射到方法调用的方法，如何区分委托和表达式树（以及需要时在二者之间进行转换），以及如何在需要单一值（scalar value）时处理查询等。

第6章介绍了如何定义部分类，使用可空类型，以及在使用数组参数时避免协变和逆变问题等内容。

示例代码

本书中给出的示例代码不是完整的程序，而是小块的代码片断，但已足够说明问题。在有些示例中，方法名称就表明了该方法要完成的任务，例如`AllocateExpensiveResource()`。这样你无需阅读整篇的代码即可快速领悟到其表达的含义，从而应用到你的开发中。在省略了方法实现时，方法名称就表明了该方法的用途。

在所有的代码片断中，均假设引入了如下命名空间：

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

若是使用到了其他命名空间中的类型，我将在类型中显式给出命名空间。

在本书的前3章中，我会尽可能地使用C# 2.0和C# 3.0中的新语法，即使这些新语法并不是必需的。在第4章和第5章中，则假设你已经熟悉了3.0的新语法。

建议和反馈

我已经仔细审校过本书，但若是你确信找到了错误，请通过电子邮件联系我：bill.wagner@srtsolutions.com。本书勘误将发布至http://srtsolutions.com/blogs/MoreEffectiveCSharp之上。

致谢

最近，一位同事问起我写完一本书之后的感觉。我说有一种类似发布了一个软件产品一样的满足和轻松。尽管写书工作量很大，但它确实也带来很多的成就感。与成功发布软件产品一样，完成一本书需要很多人的共同努力，这些人理应得到感谢。

在2004年撰写*Effective C#*时，我很荣幸成为Effective Software Development丛书的一名作者。现在这本涉及大量C#语言变化的*More Effective C#*继续成为该丛书中的一员，则更让我受宠若惊。撰写本书的想法源于我在2005年的PDC大会上与Curt Johnson和Joan Murray的一顿晚餐，那时我已被Hejlsberg和其他C#团队成员的演示介绍深深震撼。从那时开始，我就开始关注C#语言的变化，并仔细思考这些变化将为C#开发者带来怎样的影响。

当然，在我有能力和自信给出关于所有这些新特性的建议之前，仍旧需要花费时间来使用这些功能，并与同事、客户以及社区中的其他开发者讨论各种不同的使用方式。在万事俱备之后，我便正式开始撰写本书了。

我非常幸运拥有一支优秀的技术审校队伍。他们向我介绍新的主题，修改现有的建议，并挑出了早期草稿中出现的很多错误。Bill Craun、Wes Dyer、Nick Paldino、Tomas Restrepo和Peter Ritchie都提供了详细的技术反馈意见，让本书更加实用。Pavin Podila还审校了本书的WPF部分，保证了其正确性。

在写作的过程中，我和社区以及C#开发团队的成员讨论了很多想法。安阿伯.NET开发组、大湖区.NET用户组、大兰辛地区用户组、西密歇根.NET用户组和托莱多.NET用户组中的成员都充当了本书各个条目的早期审校者。此外，CodeMash的参会者也帮我决定了各个条目的取舍。特别是Dustin Campbell、Jay Wren和Mike Woelmer，他们和我讨论了很多想法。Mads Torgersen、Charlie Calvert和Eric Lippert也曾帮我澄清了本书中的很多条目。特别值得一提的是，Charlie Calvert帮我更好地将工程师的思维用写作者的表达方法变成文字。若是没有这些讨论，本书将远远无法清晰明了，且可能会错过很多重要的概念。

从头到尾两次经历了Scott Meyers的完整审校流程之后，我可以闭着眼推荐他经手的每一本书。Scott Meyers虽不是C#专家，但他的天分和对图书质量的认真负责让我非常佩服。他对本书的审读意见非常多，逐一解答确认也花费了很长时间，但这保证了书稿的质量。

在本书出版的整个过程中，Joan Murray的作用无可替代。作为编辑，她总能帮我想到所有的一切。在我需要监督的时候给出提醒，为我找到优秀的审校者，并帮助我把本书的想法变为大纲、再到草稿、直到现在你手中的成书。还有Curt Johnson，他们让我与Addison-Wesley出版社的合作非常愉快。

最后一步是与文字编辑的配合。Betsy Hardinger帮助将工程师写出的枯燥文字转变成了生动的书面英语，而并没有损害技术上的准确性。在她编辑之后，本书的语言变得更加地流畅。

当然，撰写图书也要花费大量的时间。在这段时间中，Dianne Marsh（SRT Solutions的另一个所有人）让公司保持正常运转。而最大的牺牲则来自于我的家庭，在撰写本书时，我无暇悉心顾及他们的感受。最真诚的感谢送给Marlene、Lara、Sarah和Scott，感谢你们再次全力支持我的投入。

目　录

第1章 使用泛型

毋庸置疑，C# 2.0所添加的泛型（generic）特性极大地影响了开发人员编写代码的方法和方式。很多文章和论文均详细分析过泛型相对于从前版本中C#集合类的优势，这些分析都是正确的。与使用传统的弱类型集合（依赖于System.Object）相比，使用泛型集合能够保证编译期类型安全，并提高应用程序的执行效率。

不过，有些文章和论文可能会让你有这样的想法：泛型仅仅在集合这个上下文中才有用武之地。但事实并非如此。泛型还能够用于很多其他场合中，例如创建接口、事件处理程序以及常用算法等。

也有一些讨论将C#的泛型和C++的模板进行对比，这类讨论通常的结论是某一种要好于另一种。虽然将C#泛型和C++模板比较能够帮助你理解泛型的语法，不过其作用也就仅此而已。有些使用方法在C++模板中显得更加自然，而有些则在C#泛型中略胜一筹。不过，若一味沉浸在探究哪一种“更好”之中，那么最终只会让你迷失于这无谓的争辩中，失去了比较的本意（可以参考稍后将介绍的条目2）。为C#语言添加泛型支持需要修改C#编译器、JIT（Just In Time）编译器以及CLR（Common Language Runtime，公共语言运行时）。其中，C#编译器根据C#代码生成以微软中间语言（Microsoft Intermediate Language，MSIL或IL）表示的泛型类型定义。而JIT编译器则会把泛型类型定义与一系列的类型参数组合起来，从而创建出封闭的泛型类型。CLR将在运行时同时支持上述两种概念。

泛型类型定义有利也有弊。有些时候，将代码转换为泛型写法可以减少程序的大小。而有些时候反而会事与愿违。这取决于程序中使用的类型参数

的个数，以及创建出的封闭泛型类型的个数。

泛型类定义能够完整地编译为MSIL类型。对于任何满足约束的类型参数，泛型类型中包含的代码必须保证完全合法。所有类型参数已经明确给出的泛型类型叫做*封闭泛型类型*（closed generic type），而若是仅给出了部分类型参数，那么这种泛型叫做*开放泛型类型*（open generic type）。

IL中的泛型可以看作是某个实际类型定义的一部分。IL为初始化某个完整的泛型类型实例预留了占位符。JIT编译器将在运行时生成机器代码时补全该封闭泛型类型的定义。这样的处理方法自然带来了一个矛盾：其劣势在于多个封闭泛型类型会增大处理代码的开销，而优势则体现在存储数据的时间/空间开销会减少。

不同的封闭泛型类型可能会导致代码生成不同的最终运行时形式。在创建多个封闭泛型类型时，JIT编译器和CLR均会对过程进行优化，以便降低对内存的压力。IL形式的程序集将被加载至内存中的数据页。只有在JIT编译器将IL转换成机器指令之后，生成的机器码才会被放置于只读的代码页中。

无论是否泛型，每个类型都会执行上述过程。对于非泛型类型，类的IL和其生成的机器码之间是一对一的关系。而泛型的出现则让转换的过程变得略加复杂。在JIT对泛型类进行转换时，JIT编译器将检查当前的类型参数，并根据该信息生成特定的指令。JIT编译器将会对该过程进行一系列的优化，以便让不同类型参数能够使用同样的机器码。首先也是最重要的，JIT编译器将专门为所有引用类型生成泛型类的一个机器码版本。

如下的实例化代码在运行时的机器码均完全相同。

```
List <string> stringList = new List<string>();
List<Stream> OpenFiles = new List<Stream>();
List<MyClassType> anotherList = new List<MyClassType>();
```

C#编译器将在编译期保证类型安全，有了这种保证之后，JIT编译器即可生成更加优化的机器码。

而若是某个封闭泛型类型中使用了至少一个值类型的类型参数，那么JIT编译器将采用不同的策略。在这种情况下，JIT编译器将为不同的类型参数创建不

同版本的机器指令。因此，下面的三种封闭泛型类型所生成的机器码将各不相同。

```
List<double> doubleList = new List<double>();
List<int> markers = new List<int>();
List<MyStruct> values = new List<MyStruct>();
```

虽然看上去比较有意思，不过这和我们开发者又有什么关系呢？答案就是，使用多个引用类型参数的泛型类型并不会影响程序的内存占用，因为其被JIT编译后只生成一份代码。不过若是封闭泛型类型中包含值类型作为参数，那么其JIT编译后的代码则会各不相同。我们再来深入挖掘一下上述过程，看看其带来的影响。

当运行时需要JIT编译一个泛型定义（泛型方法或泛型类），且至少有一个类型参数为值类型时，那么该过程可以分为两个步骤。首先，编译器将创建一个新的IL类，用来表示该封闭泛型类型。例如，在泛型定义中将T用int或其他某种值类型替换。随后，JIT将把该代码编译成x86指令。这两个步骤非常有必要，因为JIT并不是在某个类加载时就为其生成完整的x86指令，而是仅在类中的每个方法被第一次调用时才开始编译的。这样，框架有必要在IL代码上先执行一个替换的步骤，随后再像普通类定义一样按需编译。

这也就意味着运行时的额外内存占用将分为如下两个部分：一是为每种用值类型作为参数的封闭泛型类型保存一份IL定义的副本，二是为每种用值类型作为参数的封闭类型保存一份所调用方法的机器码的副本。

不过这个使用值类型作为泛型参数的做法也有它的好处：避免了对于值类型的装箱和拆箱操作，这样也就降低了值类型的代码/数据所占用的空间。此外，类型安全可以由编译器保证，也就让框架不必忙于进行运行时检查，进一步降低了代码量并提高了程序的性能。不仅如此，与创建泛型类相比，创建泛型方法将有助于降低为支持不同实例而需要额外生成的IL代码量（将在条目8中介绍）。只有实际用到的方法才会被实例化。非泛型类中定义的泛型方法将不会被JIT编译。

本章将介绍泛型的种种使用方法，以及创建泛型类型或方法的一些技巧，从而节约时间并有助于创建具有更高复用性的组件。本章也将介绍如何把.NET 1.x的类型（使用System.Object）迁移至使用泛型的.NET 2.0类型。

条目1：使用1.x框架API的泛型版本

.NET平台的头两个版本并不支持泛型。因此只能基于System.Object来编码，然后通过必要的运行时检查保证程序的正确性（一般来讲，将会检查该对象运行时类型是不是System.Object的某个特定的子类型）。甚至在.NET Framework中也大量存在着这种设计，因为框架本身也需要为开发人员提供一系列的底层组件库。

毫无疑问，System.Object是所有类型的最终基类。因此，使用它就意味着程序可以在此处使用任意的类型。不过编译器对类型的了解也就到此为止了，我们必须小心翼翼地对待每一行代码，使用我们组件的其他开发人员也需要非常仔细。无论是把System.Object作为参数，还是作为返回值，都难免在运行时得到意外的类型输入，随即不可避免地导致运行时错误。

在泛型出现之后，这样的日子就一去不复返了。若是你有过使用从前版本.NET的经历，那么定会非常赞同用最新的泛型版本来替代原有的传统类和接口。将System.Object用泛型的类型参数替代能够极大地提高代码质量。为什么呢？因为泛型类型的约束让人很难传入错误类型的参数。

如果上述程序健壮性的优势无法说服你从原始的System.Object迁移到泛型代码，那么性能则会是第二个理由。.NET 1.1强制使用最基础的System.Object类型，只有在使用之前才将其动态强制转换为需要的类型。1.1版本的任何类/接口都需要在进行值类型和System.Object转换的过程中执行装箱/拆箱的操作。根据实际情况，这个操作有可能会给性能带来巨大的影响。虽然这仅仅影响到了值类型，不过正如前面提到过的那样，1.1版本的弱类型系统需要我们在程序中添加很多检查性质的代码，以便保证参数和返回值类型的正确。哪怕检查结果没有任何问题，这些冗繁的步骤也会带来额外的性能开销。而在失败时的开销会更大：包括栈审核（stack walk）和转换异常时的栈解退（stack unwinding）、运行时定位catch语句等。总体说来，弱类型系统将带来各种各样的麻烦，从性能低下直至程序异常终止等。