TURING 图灵程序设计丛书 .NET系列

Addison Wesley

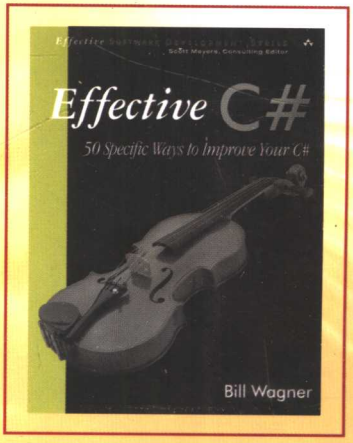
Effective C#
50 Specific Ways to Improve Your C#

Effective C# 中文版

改善C#程序的50种方法

[美] Bill Wagner 著
李建忠 译

- 业界专家经验荟萃
- 讲述从优秀到卓越的秘诀
- 涵盖 C# 2.0



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 .NET系列

Effective C# 中文版
改善C#程序的50种方法

Effective C#
50 Specific Ways to Improve Your C#

[美] Bill Wagner 著

李建忠 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

Effective C#中文版：改善 C#程序的 50 种方法 /（美）瓦格纳（Wagner, B.）著；李建忠译.
—北京：人民邮电出版社，2007.5
（图灵程序设计丛书）
ISBN 978-7-115-15888-8

Ⅰ.E... Ⅱ.①瓦...②李... Ⅲ. C 语言—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2007）第 025036 号

内 容 提 要

本书围绕一些关于 C#和.NET 的重要主题，包括 C#语言元素、.NET 资源管理、使用 C#表达设计、创建二进制组件和使用框架等，讲述了最常见的 50 个问题的解决方案，为程序员提供了改善 C#和.NET 程序的方法。本书通过将每个条款构建在之前的条款之上，并合理地利用之前的条款，来让读者最大限度地学习书中的内容，为其在不同情况下使用最佳构造提供指导。

本书适合各层次的 C#程序员阅读，同时可以推荐给高校教师（尤其是软件学院教授 C#/.NET 课程的老师），作为 C#双语教学的参考书。

图灵程序设计丛书

Effective C# 中文版：改善 C# 程序的 50 种方法

◆ 著 [美] Bill Wagner
译 李建忠
责任编辑 傅志红

◆ 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
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：800 × 1000 1/16
印张：20
字数：382 千字 2007 年 5 月第 1 版
印数：1–5 000 册 2007 年 5 月北京第 1 次印刷

著作权合同登记号 图字：01-2005-3579 号
ISBN 978-7-115-15888-8/TP

定价：49.00 元

读者服务热线：(010)88593802 印装质量热线：(010)67129223

版 权 声 明

前　言

本书就如何高效使用C#语言和.NET库，为程序员们提供了一些实用的建议。本书由50个关键条款（也可看作是50个小主题）组成，这些主题反映了我（及其他C#顾问）和C#开发人员共事时遇到的最常见问题。

与很多C#开发人员一样，我是在从事10多年C++开发之后开始使用C#的。在本书中，讨论了哪些情况下遵循C++实践可能会在使用C#时引发的问题。有一些使用C#的开发人员有着深厚的Java背景，他们可能会发现有些变化相当明显。因为从Java到C#，一些最佳实践发生了改变，我建议Java开发者要格外注意有关值类型的论述（参见第1章）。此外，.NET垃圾收集器和JVM垃圾收集器的行为方式也不尽相同（参见第2章）。

本书中的条款汇集了我最常提供给开发者的建议。虽然并非所有条款都是通用的，但大多数条款都可以很容易地应用到日常的编程工作中。这些条款涵盖了对属性（条款1）、条件编译（条款4）、常量性类型（条款7）、相等判断（条款9）、ICloneable（条款27）和new修饰符（条款29）的论述。我的经验是，在大多数情况下，减少开发时间和编写出色的代码应该是程序员的主要目标。某些科学和工程应用程序最重视的可能是系统的整体性能。对其他应用程序而言，凡事都应该围绕可伸缩性展开。对于不同的目标，可能会找到某些情况下比较重要（或不太重要）的信息。针对这一问题，我设法对各种目标进行了详细的解释说明。书中关于readonly和const（条款2）、可序列化的类型（条款25）、CLS兼容（条款30）、Web方法（条款34）和DataSet（条款41）的讨论针对某些特定的设计目标。这些目标在相应的条款中有清楚的说明，这样读者就可以在特定的情况下决定最适用的做法。

虽然本书中的每个条款都是独立的，但是这些条款是围绕一些重要的主题（如C#语法、资源管理和对象及组件设计）组织起来的，理解这一点非常重要。这并非无心之举。我的目的就是通过将每个条款构建在之前的条款之上，并合理地利用之前的条款，来让读者最大限度地学习书中的内容。尽管如此，大家仍然不要忘了举一反三。对于特定的问题，本书也可

以作为一个理想的查询工具。

要记住的是，本书并不是C#语言的教程或指南，也不是为了教授大家C#语法或结构。我的目标是为大家在不同的情况下使用什么语言构造最好提供指导。

本书面向的读者

本书是为专业的开发人员，也就是那些在日常工作中使用C#的程序员们编写的。本书的阅读前提是读者有面向对象的编程经验，并且至少用过一种C系列语言（C、C++、C#或Java）。有Visual Basic 6背景的开发人员在阅读本书之前，应该先熟悉C#语法和面向对象设计。

另外，读者应该在.NET的重要领域有一些经验：Web Services、ADO.NET、Web Forms和Windows Forms。

为了充分利用本书，大家应该理解.NET环境处理程序集的方式、微软中间语言（MSIL）和可执行代码。C#编译器生成的程序集会包含MSIL，我经常将其简写为IL。加载程序集的时候， JIT（Just In Time）编译器会将MSIL转换为机器可执行的代码。C#编译器确实会执行一些优化，但是JIT编译器会负责处理很多更高效的优化，如内联。在书中，我对各种优化所涉及的过程进行了说明。这种两阶段的编译过程对于在不同情形下哪种构造的表现最佳有着很重要的影响。

本书内容

第1章“C#语言元素”讨论的是C#语法元素和`System.Object`的核心方法，`System.Object`是编写每一个类型都要涉及的。声明、语句、算法和`System.Object`接口，这些都是编写C#代码时必须时刻记住的主题。此外，与值类型和引用类型之间的区别直接相关的条款也都在本章。根据使用的是引用类型（类）还是值类型（结构），很多条款内容都有一些不同。在深入阅读本书之前，我强烈建议大家先阅读有关值类型和引用类型的讨论（条款6~8）。

第2章“.NET资源管理”涵盖了C#和.NET的资源管理问题。大家会学习如何针对.NET管理的执行环境优化资源分配和使用模式。是的，.NET垃圾收集器使我们的工作简单了很

多。内存管理是环境的职责，而非开发人员的职责。但是，我们的行为对垃圾收集器在应用程序中的执行效果会产生大的影响。而且，尽管内存不是我们的问题，但管理非内存资源仍然是我们的职责，后者可以通过IDisposable进行处理。在这里，大家可以学习.NET中资源管理的最佳做法。

第3章“使用C#表达设计”从C#的角度讲解了面向对象设计。C#提供了丰富的工具供我们使用。有时候，相同的问题可以用很多不同的方法解决：使用接口、委托、事件或者特性和反射。选用哪一种方式，对系统将来的可维护性会产生很大的影响。选择最佳的设计表示可以帮助程序员们更容易地使用类型。最自然的表示会使我们的意图更加清晰。这样，类型就会比较容易使用，而且不容易误用。第3章中的条款集中讲解了我们所做的设计决定，以及各种C#惯用法最适用的场合。

第4章“创建二进制组件”讲解了组件和语言互操作性。大家将学习如何在不牺牲C#功能的情况下，编写可被其他.NET语言使用的组件。还将学习如何将类细分成组件，来升级应用程序的某些部分。我们应该能在不重新发布整个应用程序的情况下发布组件的新版本。

第5章“使用框架”讲解了.NET框架未充分使用的部分。我看到很多开发人员非常希望创建自己的软件，而不是使用已经构建好的软件。这可能是由.NET框架的体积造成的，也可能因为框架是全新的。这些条款涵盖了框架中那些我曾见过开发人员做重复劳动、而非使用业已存在的功能的部分。通过学习更高效地使用框架，大家可以节省宝贵的时间。

第6章“杂项讨论”以不适合其他分类的条款以及对未来的展望作为全书的结尾。有关C# 2.0、标准、异常安全（exception-safe）的代码、安全和互操作的信息，都可以在这里找到。

关于条款

我写这些条款是为了向大家提供编写C#软件的简洁明了的建议。书中有一些指导方针是通用的，因为它们会影响程序的正确性，如正确初始化数据成员（参见第2章）。有一些指导方针不是很容易理解，并且在.NET社区中引发过很多争论，如是否使用ADO.NET DataSet。我个人认为使用它们可以节省很多时间（参见条款41），其他一些专业的程序员，同时也是我非常尊敬的程序员，对此并不同意。它其实取决于我们正在构建的软件性质。我的立场是尽量节省时间。如果是编写大量在基于.NET和基于Java的系统之间传输信息的软件，DataSet就是个糟糕的主意。在整本书中，我为所做的全部建议都给出了理由。如果其理由

并不适用于你碰到的情况，那就不要采纳书中的建议。当建议是普遍适用时，我通常会省略其显而易见的理由：如果不这样做，程序就起不了作用。

版式和代码约定

写编程语言图书的一个困难之处在于，语言设计者用一些英文单词表示非常特殊的新含义，这就导致了一些很难理解的句子。“Develop interfaces with interfaces”就是一个例子。因此，我在使用语言关键字时，都采用了代码体。

本书使用了很多相关的C#术语。在提到类型的成员时，它是指可以成为类型的一部分的任何定义：方法、属性、字段、索引器、事件、枚举或者委托。当应用的只是一种定义时，我会使用一个更加具体的术语。对于书中的许多术语，大家可能熟悉，也可能还不熟悉。当这些术语第一次在正文中出现时，它们会以楷体形式表现，并给出定义。

本书的范例都是简短、专注的代码段，以示范特定条款中的建议。列出它们是为了强调遵循建议的好处。它们并不是可以加入到读者当前程序中的完整范例。大家不能简单地复制代码清单，然后编译它们。所有代码清单我都省略了很多细节。在所有情况下，我们都预设已存在如下常用的using语句：

```
using System;
using System.IO;
using System.Collections;
using System.Data;
```

当使用不太常见的命名空间时，我会确保让读者看到相关的命名空间。简短的范例会使用完全限定类名称，而长的范例则会包含不太常用的using语句。

范例中的代码也比较随意。例如，当显示下列代码时：

```
string s1 = GetMessage( );
```

如果和论述的内容无关，我可能不会显示GetMessage()例程的主体。当我省略代码时，读者可以假定缺失的方法做的是一些明显且合理的事情。我这样做的目的是为了让我们把焦点聚在特殊的主题上。通过省略和主题无关的代码，我们的注意力就不会分散。这样还能让各个条款保持简短，以使大家能够在短时间内完成学习。

关于 C# 2.0

我之所以对新的C# 2.0版本所言甚少，有两个原因。首先，本书中的大部分建议也同样适用于C# 2.0。虽然C# 2.0是一个非常重大的升级版本，但它是建立在C# 1.0基础之上的，且并没有让现如今的建议失效。对于最佳实践有可能发生变化的地方，我已经在文中给出了说明。

第二个原因是现在编写新的C# 2.0功能的高效用法还为时过早。本书的内容是基于我以及我的同事使用C# 1.0的已有经验。我们对于C#2.0中的新功能还没有足够的经验，因而也就不了解能够应用到日常任务中的最佳做法。当在书中编写C# 2.0新功能的高效用法的时机还未成熟时，我并不想误导读者。

建议、反馈及获取本书的更新内容

本书内容基于我的经验以及和同事们的交流。如果读者有不同的经验，或者有任何疑问或意见，我愿洗耳恭听。请通过电子邮件和我联系：wwagner@srtsolutions.com。我会把这些意见放在网上，作为本书的延伸。登录www.srtsolutions.com/EffectiveCSharp，可以看到当前的讨论。

致谢

虽然写作似乎是一件孤独的事情，但本书却是一大群人的成果。我非常幸运，认识两位出色的编辑Stephane Nakib和Joan Murray。Stephane Nakib第一次联系我为Addison Wesley写作是在一年多以前。我当时心存疑虑，因为书店里到处都是.NET和C#方面的书籍。在C# 2.0面世了足够的时间，可以大书特书之前，我一直看不出为C#和.NET再写一本参考书、教程或编程书籍的必要性。我们讨论过好几个想法，话题总是回到写一本有关C#最佳实践的图书。在进行这些讨论的过程中，Stephane告诉我，Scott Meyers开始着手主编一个Effective系列，其风格延续了他的*Effective C++*系列图书。我买的Scott的三本书都被我翻得非常破旧。我还将它们推荐给了我认识的每一位专业C++程序员。他的写作风格清晰而简明。每一个建议条款都有过硬的理由。*Effective*丛书是很好的资源，而且其体例使得读者很容易就能记住其中的建议。我认识很多C++开发人员，他们复印了书的目录，并把它钉在书房的墙上，不断提醒自己。Stephane一提到写作*Effective C#*的想法，我就欣然接受了这个机会。这本书把我曾

为C#开发人员给出的所有建议收录在一起。我很荣幸能成为该系列图书的一个作者。和Scott一起工作让我学到了很多东西。我真心希望本书能够像Scott的书提高了我的C++技巧那样，帮助大家提高C#应用技巧。

Stephane帮助落实了写作*Effective C#*的想法，她审读了提纲和草稿，并在该书的早期写作过程中给予了充分的支持。当她抽身离开的时候，Joan Murray接管了这个项目，并毫无倦怠地负责了原稿的写作管理。Ebony Haight作为编辑助理，在整个过程中提供了不间断的帮助。Krista Hansing完成了所有编辑和转换编程行话的工作。Christy Hackerd完成了所有把Word文档转变为成书的工作。

书中如有错误，应由我来负责。出色的审稿团队修改了绝大多数的错误、冗长和表述不清的问题。最值得一提的是，Brian Noyes、Rob Steel、Josh Holmes和Doug Holland使得最终的正文比初期的草稿更加正确和有用。另外，还要感谢安阿伯计算机学会、大湖区.NET用户组、Greater Lansing用户组和西密歇根州.NET用户组的所有成员，他们听取了有关这些条款的议论，并提供了出色的反馈。

特别要提到的是，Scott Meyers的参与对本书的最终版本有着巨大的积极影响。和他讨论本书的早期草稿，使得我更加明白为什么自己会把*Effective C++*丛书用得破旧不堪。再小的问题，也逃不过他的眼睛。

我要感谢MyST Technology Partners（`myst-technology.com`）的Andy Seidl和Bill French。我使用了一个基于MyST的安全博客网址向审稿人公布了各个条款的早期草稿。之后我们向公众公开了部分站点，以便大家能够以在线的格式看到本书的部分内容。登录www.srtsolutions.com/EffectiveCSharp，可以阅读在线版本。

到目前为止，我已经为杂志写了好几年的文章，我要在此感谢那个将我引进门的人：Richard Hale Shaw。他在自己参与创办的杂志《Visual C++开发者》上邀请我这个未经检验的作者开设了一个专栏。如果没有他的帮助，我不会发现自己对写作的热爱。没有他最初给我的帮助，我也不会有机会为*Visual Studio*杂志、*C# Pro*或*ASP.NET Pro*撰稿。

一路走来，我幸运地和不同杂志的很多出色的编辑共过事。我想把他们的名字全都列在这里，但空间不允许。有一个人非提不可，那就是Elden Nelson。我享受与他共事的所有时光，他对我的写作风格产生了很大的积极影响。

我的业务伙伴Josh Holmes和Dianne Marsh，他们容忍了我对公司业务的有限参与，而让我把时间用来写作本书。他们还帮助审阅了我的原稿、想法和条款中的思想。

在整个漫长的写作过程中，我的父母Bill和Alice Wagner“做事要有始有终”的忠告，成为我最终完成本书的唯一原因。

最后也是最重要的，我要感谢我的家人Marlene、Lara、Sarah和Scott。写书会占用大量的业余时间。在我为本书付出了所有时间之后，他们表现出来的却是始终如一的耐心。

目 录

第1章 C#语言元素

为什么程序已经可以正常工作了，我们还要改变它们呢？答案就是我们可以让它们变得更好。我们常常会改变所使用的工具或者语言，因为新的工具或者语言更富生产力。如果固守旧有的习惯，我们将得不到期望的结果。对于C#这种和我们已经熟悉的语言（如C++或Java）有诸多共通之处的新语言，情况更是如此。人们很容易回到旧的习惯中去。当然，这些旧的习惯绝大多数都很好，C#语言的设计者们也确实希望我们能够利用这些旧习惯下所获取的知识。但是，为了让C#和公共语言运行库（Common Language Runtime，CLR）能够更好地集成在一起，从而为面向组件的软件开发提供更好的支持，这些设计者们不可避免地需要添加或者改变某些元素。本章将讨论那些在C#中应该改变的旧习惯，以及对应的新的推荐做法。

条款1：使用属性代替可访问的数据成员

C#将属性从其他语言中的一种特殊约定提升成为一种第一等（first-class）的语言特性。如果大家还在类型中定义公有的数据成员，或者还在手工添加`get`和`set`方法，请赶快停下来。属性在使我们可以将数据成员暴露为公有接口的同时，还为我们提供了在面向对象环境中所期望的封装。在C#中，*属性*（property）是这样一种语言元素：它们在被访问的时候看起来好像是数据成员，但是它们却是用方法实现的。

有时候，一些类型成员最好的表示形式就是数据，例如一个客户的名字、一个点的*x*/*y*坐标，或者上一年的收入。使用属性我们可以创建一种特殊的接口——这种接口在行为上像数据访问，但却仍能获得函数的全部好处。客户代码[1]对属性的访问就像访问公有变量一样。但实际的实现采用的却是方法，这些方法内部定义了属性访问器的行为。

1. 即使用属性的代码。——译者注

.NET框架假定我们会使用属性来表达公有数据成员。事实上，.NET框架中的数据绑定类只支持属性，而不支持公有数据成员。这些数据绑定类会将对象的属性关联到用户界面控件（Web控件或者Windows Forms控件）上。其数据绑定机制事实上是使用反射来查找一个类型中具有特定名称的属性。例如下面的代码：

```
textBoxCity.DataBindings.Add("Text",
  address, "City");
```

便是将textBoxCity控件的Text属性和address对象的City属性绑定在一起。（有关数据绑定的细节，参见条款38。）如果City是一个公有数据成员，这样的数据绑定就不能正常工作。.NET框架类库（Framework Class Library）的设计者们之所以不支持这样的做法，是因为将数据成员直接暴露给外界不符合面向对象的设计原则。.NET框架类库这样的设计策略从某种意义上讲也是在推动我们遵循面向对象的设计原则。对于C++和Java编程老手，我想特别指出的是这些数据绑定代码并不会去查找get和set函数。在C#中，我们应该忘掉get_和set_这些旧式的约定，而全面采用属性。

当然，数据绑定所应用的类一般都要和用户界面打交道。但这并不意味着属性只在UI（用户界面）逻辑中有用武之地。对于其他类和结构，我们也需要使用属性。随着时间的推移，新的需求或行为往往会影响原来类型的实现，采用属性比较容易能够应对这些变化。例如，我们可能很快就会发现Customer类型不能有一个空的Name。如果我们使用一个公用属性来实现Name，那么只需要在一个地方做更改即可：

```
public class Customer
{
  private string _name;
  public string Name
  {
    get
    {
      return _name;
    }
    set
    {
      if (( value == null ) ||
```

```
            ( value.Length == 0 ))
            throw new ArgumentException( "Name cannot be blank",
              "Name" );
          _name = value;
        }
      }

      // ……
    }
```

如果使用的是公有数据成员，我们就要寻找并修改所有设置Customer的Name的代码，那将花费大量的时间。

另外，由于属性是采用方法来实现的，因此为它们添加多线程支持就更加容易——直接在get和set方法中提供同步数据访问控制即可：

```
public string Name
{
  get
  {
    lock( this )
    {
      return _name;
    }
  }
  set
  {
    lock( this )
    {
      _name = value;
    }
  }
}
```

既然是采用方法来实现的，那么属性也就具有了方法所具有的全部功能。比如，属性可以实现为虚属性：

```
public class Customer
{
  private string _name;
```

```
  public virtual string Name
  {
    get
    {
      return _name;
    }
    set
    {
      _name = value;
    }
  }

  // 忽略其他实现代码。
}
```

自然，属性也可以实现为抽象属性，或者作为接口定义的一部分：

```
public interface INameValuePair
{
  object Name
  {
    get;
  }

  object Value
  {
    get;
    set;
  }
}
```

最后，我们还可以借助属性的特点来创建const和非const版本的接口：

```
public interface IConstNameValuePair
{
  object Name
  {
    get;
  }
```