PROGRAMMER TO PROGRAMMER™

Visual Basic .NET Class Design Handbook

Coding Effective Classes

Visual Basic .NET 类设计手册

Andy Olsen
Damon Allison
James Speer 著
袁勤勇
吴 静 等译

清华大学出版社
http://www.tup.com.cn

Visual Basic .NET 类设计手册

Andy Olsen
Damon Allison
James Speer
著

袁勤勇　吴静　等译

清华大学出版社

(京) 新登字 158 号
北京市版权局著作权合同登记号：01-2002-0943

内 容 简 介

本书主要讲解了用 VB.NET 设计类时的一些具体细节。首先介绍什么是类型，接下来具体介绍组成类型的类型成员(如方法、构造函数、属性和事件等)，最后研究如何将类型集中在一起组成程序集。本书适用于希望深入理解.NET 平台的 VB.NET 开发人员，通过对本书的学习，可以进一步理解 VB.NET 为设计类而提供的各种机制。

书　　名：Visual Basic.NET 类设计手册
作　　者：Andy Olsen，Damon Allison，James Speer　著　袁勤勇　吴静　等译
出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
　　http://www.tup.com.cn
责任编辑：徐燕萍
封面设计：康博
版式设计：康博
印 刷 者：北京通州区大中印刷厂
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　印张：21.5　字数：550 千字
版　　次：2002 年 11 月第 1 版　2002 年 11 月第 1 次印刷
书　　号：ISBN 7-302-06038-X/TP・3603
印　　数：0001～4000
定　　价：43.00 元

出版者的话

近年来，国内计算机类图书出版业得到了空前的发展，面向初级用户的应用类软件图书铺天盖地，但是真正有深度和内涵的讲解专业开发知识的图书不多。已经掌握计算机和网络基础知识的人们，尤其是IT专业人士迫切需要“阳春白雪”。IT图书市场呼唤精品！

为了满足这种市场需求，清华大学出版社从世界出版业知名品牌Wrox出版公司引进了受到无数IT专业人士青睐，被奉为IT出版界经典之作的Handbook系列丛书。这套讲述专项开发技能的编程丛书，从头到尾都贯穿了Wrox出版公司“由程序员为程序员而著(Programmer to Programmer)”的出版理念，每一本书无不是出自软件大师之手。实际上，Wrox公司的图书作者都是世界顶级IT公司(如Microsoft，IBM，Oracle以及HP等)的资深程序员，他们的作品既深入研究编程机理，传授最新编程技术，又站在程序员的角度，指导程序员拓展编程思路，学习实用开发技巧，从而风靡世界各地，被IT专业人士和程序员视为职业生涯中的必读之作。

为了保证该系列丛书的质量，清华大学出版社迅速组织了一批位于IT开发领域前沿的专家学者进行翻译，经过编辑人员的进一步加工整理后，现陆续奉献给广大读者。

读者可以从www.wrox.com网站下载所需的源代码并获得相关的技术支持。同时，也欢迎广大读者参与p2p.wrox.com网站上的在线讨论，与世界各地的编程人员交流读书感受和编程体验。

前　言

想请问一下听说过 Visual Basic .NET 的人，VB.NET 超越 Visual Basic 6 的最大改变是什么，很可能他们会说是“面向对象”。但是这表示什么意思呢？在某种程度上，Visual Basic 从 VB5 开始就已经是面向对象的了，那么.NET 本身的创新之处又在哪里呢？大多数人认为应该是继承，这样可以利用代码重用和类层次结构的多态性。那确实是在我们的代码中使用面向对象技术的一种可见方式，但是还有一个至关重要的基本转变，即我们在任何时候编写的所有代码最后都属于一个类，VB.NET 程序员必须习惯这一点，虽然这在 VB.NET 中并没有得到广泛的重视。

因此，每次当您使用 Visual Studio .NET 或记事本开始编写 VB.NET 代码时，您实际上正在设计一个类。当您编写一个 Sub 时，您是在编写一个方法；当您声明一个 Event 时，您是在创建字段和其他类。理解这一点后，您就可以知道我们能把什么放入一个类中，以及它对.NET 运行时的真正意义，它能完全利用 Visual Basic .NET 的基础。本系列丛书的目的是解释如何使用语言来最有效地使用 .NET 平台，而本书的目的是解释如何使用 Visual Basic .NET 来最有效地使用.NET 类型系统。

0.1　本书面向对象

本书面向希望探索.NET 平台全部功能的 VB.NET 开发人员。虽然编写 VB.NET 程序可能只需在 Visual Studio 中拖放，然后右击组件，将代码添加到事件处理程序中，但本书面向的是有兴趣在这些约束之外编写代码的开发人员。如果希望定义自己的数据类型，建立自己的类层次结构，或建立带有健壮接口的类，那么就需要深入理解 VB.NET 为设计类提供的机制。这也是本书的主题。

本书假定您已经使用 VB.NET 编写过代码，熟悉其基本语法。您应该熟悉所选择的开发工具，知道如何编译和运行 VB.NET 代码。

您应该知道.NET 基本的面向对象机制——例如，哪些对象是类的实例，如何实例化对象，如何访问对象的方法和属性。不过，我们会在讨论时，扼要讲述一下其中关键字的含义和语法。

0.2　本书主要内容

每次利用 VB.NET 编写代码时，我们都是在编写一个类—— 这是无法避免的。本

书讨论程序员在这个环境中所做的一些决定，把这些决定放入它们真正的上下文中：有关类设计的决定。因此，当我们编写一个 Sub， 并决定是否使它 Shared，Public 还是 Private，以及它应该接受什么参数等时，本书帮助我们在如何影响类的设计的上下文中研究这些决定。

本书并不讨论我们每天都在编写的那些代码，而是提出问题“它真正在做什么？”要求您不要只在效果方面考虑每个 VB.NET 关键字，而是要考虑它如何完成那个效果。在本书中，我们会介绍所有这些代码是如何被编译成.NET 类型的，如何定义类型成员，如何继承类型成员，如何把类型聚合到程序集中，如何能控制类型实例的创建，以及有效地编写类方面的更多内容。

注意：

在本书中，所附代码段顶部一般都省掉了“Imports System”。这样的代码段在 vbc.exe 编译器中是不能编译的，因此您在使用时应添加“Imports System”。

0.3 本书不包括的内容

本书不是一本关于面向对象分析和设计、UML 建模或设计模式的书。它不解决如何抓住业务难题的问题，决定应该编写哪些类来解决这些问题。它集中在实现的问题上：如何编写一个类来提供特定的行为。

它也不是对面向对象的基本介绍，因为任何 VB 程序员都应该已经熟悉这样的思想了，即一个对象对应了一个实例，调用它的方法并访问属性，即使没有自定义类型的过程，同样可以这样做。如果您使用对象很轻松，那么本书不一定适合于您。

0.4 本书具体内容

本书将介绍在.NET 中究竟由什么组成一个类。我们从描述类型是什么，以及类如何关联到.NET 类型框架开始，接下来讨论了组成类型的类型成员。本书大部分内容主要介绍 VB.NET 为设计类型成员(方法、构造函数、属性和事件)提供的不同机制，最后研究类型如何集中在一起组成程序集。

以下是各章的主要内容：

第 1 章——定义类型

该章阐述究竟什么是类型，类型在.NET 中发挥什么作用，以及存在哪些类型。我们还要分析可以在 VB.NET 中声明的不同类型，以及它们如何映射为.NET 类型。

第 2 章——类型成员

在第 2 章中，我们介绍类型成员：什么是类型成员，如何定义类型成员，以及如何

用 VB.NET 关键字定义类型成员。我们还要介绍由从.NET 根类 System.Object 派生出来的每个类型所继承的类型成员。

第 3 章——方法

方法是.NET 应用程序中广为应用的工具，包括所有程序逻辑。该章研究对所有方法通用的行为，以及在 VB.NET 中如何定义简单方法。

第 4 章——构造函数和对象生命周期

构造函数是有些特殊的方法，调用它可以初始化类型的新实例。在该章中，我们介绍如何编写这些特殊的方法，以及如何使用它们来控制可以创建类型实例的代码。

第 5 章——属性

属性(标量属性和索引属性)是一种机制，它允许我们创建指定方法来访问属于类型的数据。该章研究如何实现属性，索引属性如何工作，以及在 VB.NET 中默认属性所发挥的作用。

第 6 章——事件和委托

VB.NET 中最复杂的类型成员是事件。该章阐述委托如何工作，以及.NET 如何通过委托字段和指定方法来提供它的事件基础结构。

第 7 章——继承和多态性

类型并不仅仅是它的成员总和，它还包括从超类中继承的所有成员。本章研究.NET 类型的继承工作机制，成员何时被继承或不被继承，以及我们如何使用 VB.NET 控制和利用继承。

第 8 章——代码组织和元数据

当我们在 VB.NET 中编写类时，我们必须决定在何处准确地放置这个类，除了在逻辑上要放置在命名空间结构里，在物理上还要放置在源文件里，以及最终放置在.NET 程序集里。该章将讨论这些问题。我们还要使用.NET 元数据把数据添加到我们的类中，这个类也许对其他使用它的程序员有用。

0.5　前提条件

在学习本书前，您应该能够编译和执行在 Visual Basic .NET 中编写的代码。这意味着您将需要：

- .NET Framework SDK，可从 Microsoft 的 MSDN 站点 (http://msdn.microsoft.com) 的 Software Development Kits 类别中获得。在本书出版时，下载页面可通过下面的 URL 获得：

  http://msdn.microsoft.com/downloads/sample.asp?
  url=/msdn-files/027/000/976/msdncompositedoc.xml

- Visual Studio .NET，它结合了 Visual Basic .NET。包括 2002 版的 Visual

Basic .NET IDE 和下面的 Microsoft 产品：

Microsoft Visual Basic .NET Standard

Microsoft Visual Studio .NET Enterprise Architect

Microsoft Visual Studio NET Enterprise Developer

Microsoft Visual Studio .NET Professional

产品主页是 http://msdn microsoft.com/vstudio/。

还有几种支持其他平台的.NET 产品正在开发中，但是在本书出版时，都不支持 VB.NET 编译。

目　　录

第1章 定义类型

我们通常把 Visual Basic .NET 描述为一种面向对象的编程语言。这促使我们以对象的方式进行编程。但是在面向对象的分析和设计中，我们要确定系统中最重要的是哪些对象，并考虑它们之间的相关性。但在面向对象的编程中，我们不编写“对象”，而是定义类来表示对象的行为和属性。实际上，在 Visual Basic .NET 中，类仅仅是我们用来定义对象行为的所有机制中的一种，这些对象是指运行时的程序中的对象。在 Visual Basic .NET 中编写代码时，我们实际编写的是类型(type)。类型表示行为和数据存储需求的结合。当程序运行时，将创建类型实例(用以分配所需的数据存储空间)，并且使我们可以使用类型行为。

选择我们需要的类型，以及给予它们的行为和数据存储空间，这就是在.NET 中编程的所有内容。

本书打算帮助 Visual Basic .NET 开发人员了解应该如何把编程问题拆分成几个分离的行为单元——类、模块、子例程、函数、结构、委托等等。在编写 Visual Basic .NET 应用程序的过程中，我们要确定需要编写哪些类型，它们应该是哪种类型，它们应该有哪些成员，以及哪些代码该放在哪里。本书将让您知道这些决定的结果，这些都是您每次坐下编写 Visual Basic .NET 应用程序代码时所要下的决定。

我们将从究竟什么是类型来开始介绍本书。在本章中，我们将研究.NET 的类型系统，以及开发人员可以使用的类型种类。

1.1 类型

在编程中，我们使用术语“类型”来描述特殊的值。例如，Visual Basic 6 程序员比较熟悉如 Integer、Date 和 String 这样的类型。对于每种类型，编译器都必须了解下面的信息：

- 当我们创建这种类型的值时，应该分配多少内存
- 使用这个值，我们可以执行什么操作

类型的概念是“强类型”编程语言中的基础，这种编程语言包括所有.NET 语言。在强类型语言中，存储在每个变量里的值的类型在编译时都是已经知道的，因此编译器可以预测将要如何使用每个变量，这样，如果弄错了，编译器可以通知我们。

类型是一个契约。特定类型的变量可以保证，它将包含您希望给定类型变量所拥有

的所有数据，而且能以希望处理这种类型变量的所有方法来处理它。

在谈到类型时，要牢记的是，对计算机来说，所有数据都只是 1 和 0 字符串。当我们在程序中拥有一个变量时，这个变量保存的只是某个二进制数字而已。因此，当我们要计算机把这个变量显示到屏幕上，对它执行计算，或检索变量的属性时，计算机需要知道变量包含什么类型，这样才能知道如何解释它的值，并响应我们的请求。例如，整型可以存储在 4 个字节的二进制数据中。类似地，单精度浮点数以 4 个字节存储。看一下下面的 4 个字节：

```
00110110 11011011 10001010 01110100
```

如果这个值被解释为整型，它将表示数字 920357492。解释为单精度浮点值，它的近似值为 6.5428267E-6。因此，如果一个变量包含这个二进制数字，我们要求.NET 为它加 1 时，最后的结果不仅依赖于变量里的值，还依赖这个变量的类型。

类型给内存中的 1 和 0 字符串赋予语义上的意义。它也把特定行为与数据关联——举个例子，我们可以比较值，判断一个值是否大于另一个值，获取表示值的字符串，或以特定方式修改值。

我们应该认识到组成程序的所有逻辑都被封装在类型的行为里，程序在任何给定时刻的状态由这些类型的实例表示。这是学习如何利用.NET 的基础。

1.2 .NET 类型系统

.NET Framework 定义通用语言规范(CLS) 来促进无缝的语言互操作性。因此我们可以选择任何.NET Framework 语言来实现面向对象的设计。虽然我们是 Visual Basic .NET 开发人员，但是理解.NET 语言的无关性其意义也是很重要的。

CLS 使得在同一个应用程序中使用几种不同的编程语言变得非常简单。它定义了一组要求所有 CLS 兼容语言都支持的语言构造。当我们编译使用 CLS 兼容语言编写的源代码时，它被编译成叫作 Microsoft Intermediate Language (MSIL)的标准.NET Framework 字节代码格式。这意味着我们可以很愉快地在同一个应用程序中混合使用 Visual Basic .NET、Visual C#、Managed Extensions for C++、Visual J#或 JScript。还有来自第三方开发商的 许多 CLS 兼容语言，如 COBOL(Fujitsu)、Perl 和 Python (ActiveState)、Smalltalk (Quasar Knowledge Systems)以及更多其他语言。Microsoft 认为语言的选择是一种“生活方式选择”，就像您更喜欢咖啡还是茶一样。

支持所有这些语言之间平滑互操作的基础系统是通用类型系统(Common Type System)。这个系统规定如何声明新类型，如何创建这些类型的实例，以及公共语言运行时如何管理这些实例的生存期。通用类型系统是.NET Framework 的基础，因为它支持跨.NET

Framework 语言的语言互操作性。

图 1-1 显示了通用类型系统的结构。

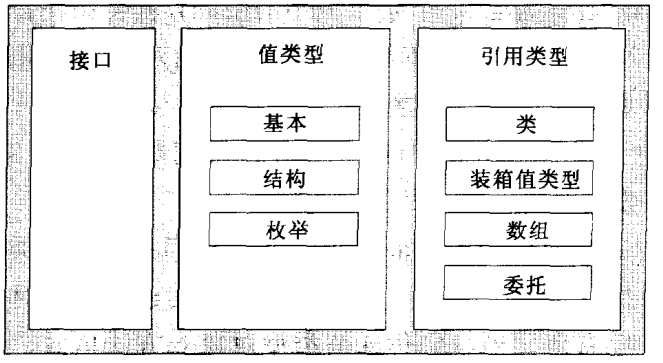

图 1-1

正如图 1-1 所显示的，通用类型系统清楚区别了值类型和引用类型。它允许我们定义纯粹的接口，接口只是一个契约的简单定义，然后由另一个类型(或一些类型)来实现——它把契约的定义和契约的实现分离开。所有其他类型都是由这两部分组成。

值类型和引用类型

值类型表示本质上是数字的实体，如日期和时间、颜色、屏幕坐标等等。这种实体可以用规定了长度的简单二进制数字表示——惟一复杂的是对这些数字含义的解释。例如，一个 8 字节长的整数可以用来表示一个很大范围的日期和时间。这个数字可以被解释为距离固定时间的时间间隔偏移量。.NET 有 64 位 DateTime 类型，它表示依照 Gregorian 日历从公元 1 世纪 1 月 1 日 00:00:00 以来的 tick 数。tick 是时间单位，它是 100 个纳秒，或微秒的十分之一。因此，DateTime 值类型可以很容易地存储为 8 字节数字。然后类型只需要提供这样的功能，允许我们从简单二进制值中以我们可以使用的格式提取年、月、日、时、分、秒等等。

由于像这样的值类型实际上由非常少的数据构成，对我们来说，在程序中传递 DateTime 信息很简单。如果我们创建一个变量 a 保存一个 DateTime ，它将需要 8 个字节的存储空间。如果我们创建另一个变量 b 来保存 DateTime，它也可以获得 8 个字节的存储空间。现在，如果我们写 b = a ，那么 a 中的 8 个字节的数据可以很快复制并放入 b 中。由于值类型的数据大小固定，因此它们可以被很快地复制，并正确存储到变量的分配内存中，使得我们能够快速访问这些数据。

同样的事情对较复杂的类型，如数组，就不一定正确了。数组没有固定的大小。当我们创建变量 c 保存整型数组时，它将需要多少字节的存储空间我们并不清楚——这得依赖于我们放入其中的数组大小。但是，当我们创建一个变量时，.NET 需要知道要

分配给它多少空间。因此，.NET 使用称为引用的数据来解决这个问题，而引用的大小是固定的。引用值必须是相同数量的字节，并指向系统内存中的托管堆。如果我们创建一个有 8 个整数的数组(将需要 32 个字节大小的空间)，就会在托管堆中分配 32 字节。当我们把这个数组放入变量 c 中，托管堆中的数组位置(对数组的引用)被放入变量中。如果我们创建另一个变量 d 保存一个整型数组，然后让 d = c，这个引用值被复制到 d 中，这样 d 指向的整型数组就和 c 指向的完全相同。任何数组，因为它的大小是可变的，所以不能是值类型。数组像类一样，是引用类型。

值类型展示了按值复制(copy-by-value)语义。当我们将一个值类型对象复制到另一个时，编译器对对象执行逐字节复制。当不需要了解创建了多少对象的副本而只关心它的值时，值类型会很有用。当我们传递值类型对象到方法里时，编译器创建对象的本地副本。当方法返回时，值类型对象使用的内存会被自动回收。就是说在我们使用值类型时还有一些重要限制。

引用类型表示拥有惟一身份的实体——它们存在于托管堆的某个地方，由变量引用。例如，假定我们定义了一个单独的 BankAccount 类，然后在应用程序中创建这个类的对象。.NET Framework 公共语言运行时将内存分配给这个对象，然后返回一个对这个新对象的引用。

当我们将 BankAccount 对象复制到另一个变量时，这个变量会引用同一个 BankAccount 对象。我们仍然只有一个 BankAccount 对象，但是现在有两个变量会引用它。引用类型展示了按引用复制的语义。同样地，当我们传递 BankAccount 对象到一个方法中，方法接收对象引用的副本，而不是值的副本。

公共语言运行时跟踪分配给托管堆中所有对象的内存。这些对象经常被称之为托管对象。在应用程序中当一个托管对象不再被引用时，对象就自动被垃圾收集器回收。垃圾收集过程定期运行，回收不再被引用的托管对象所使用的资源。我们会在第 4 章详细介绍这个过程。

在 VB.NET 文献中经常会看到，语句“值类型被存储在栈中”，这会令人误解。如果我们有一个含有几个值类型实体的数组，这些值会存储在数组中，但是数组存储在托管堆中。按那样理解的话，值类型也存储在堆中。但是当我们从数组取回这些值时，我们没有获得对堆中值位置的引用，从堆中获得的是值的副本，我们可以本地使用这个副本。值类型和引用类型的区别是：引用类型存储在托管堆的惟一位置中，而值类型存储在使用它们的地方。

创建了任何类型的变量后，变量都有一个默认值，即 0 值。在值类型情况下，这个值有一个由类型的语义所指定的特定含义，它可能表示数字 0，Boolean 状态下的 False，公元 1 世纪 1 月 1 日 00:00:00 这样的时间值等等。在引用类型情况下，它是一个不指向托管堆中有效对象的引用。这样的引用叫作空(null)引用，这在 VB.NET 中由关键字 Nothing 表示。您可以给包含了对象引用的变量赋值 Nothing，有效引用就会被空引用

代替。这很有用，这样一旦托管堆上的对象不再由存储在别处的引用进行访问，它可以由.NET 垃圾收集器清除。

.NET 值类型有 3 种。我们将在本章的剩余部分深入讨论每一种值类型：

- 基本类型

所有编程语言都定义基本类型，如整型、浮点型等等。在.NET 中 这些类型是值类型。稍后，我们将讨论 Visual Basic .NET 中的基本类型，并介绍这些类型如何映射 Microsoft Intermediate Language (MSIL) 的数据类型。

- 用户自定义的值类型

我们可以定义自己的值类型，在应用程序中表示小段数据。在 Visual Basic .NET 中，结构就是用户定义的值类型 —— 使用 Structure 关键字定义的。.NET Framework 可以使用以类似方式定义的自定义值类型，如 System.DateTime 和 System.Drawing.Point。

- 枚举类型

枚举类型是一种特殊的用户自定义的值类型，它表示一个拥有一小列允许值的类型。例如，枚举类型可能允许值为 Yes、No 和 Maybe。这些值中每个都由一个整数表示，但是定义枚举类型允许我们为特定的一组整数值指派含义。这样，枚举帮助我们避免“莫名其妙的值”出现在代码中，它使我们的代码更加容易阅读和编写——也使进行维护的程序员更容易理解我们所写的内容！例如，我们用一个枚举类型表示错误条件，可以使用这些值报告为什么操作失败，而不是使用含义模糊的错误代码。

枚举类型还帮助我们把相关常量组集合到一个单独的地方，这样能帮助我们理解这些常量的含义，以及防止在不同枚举类型中定义的常量之间的名称冲突。例如，我们可能有一个 HairColor 枚举类型，它允许 Blonde、Red、Brown 和 Black，还有一个 EyeColor 枚举类型，可以是 Blue、Green 或 Brown。这样可以确保我们不会无意中把某人描绘成蓝色的头发和红色的眼睛，但允许我们对这二者使用棕色。

下面是通用类型系统中的引用类型。我们将会在本章的后面详细说明这些类型：

- 类类型

我们在一般的应用程序中定义的大部分类型都是类类型。类为应用程序中的重要实体规定了数据和方法。

- 委托

委托在应用程序中表示指向方法的指针。我们可以用委托来指定回调方法，并为图形用户界面(GUI)注册处理程序方法，如按钮单击。

- 数组

数组被分配在托管堆上，由引用访问。数组可以是一维的，或多维的。您也可以使数组再包含数组，即允许锯齿状数组结构。

- 装箱值类型

在某些环境下，可能必须把值类型视为引用类型。我们可能希望在只能使用引用的