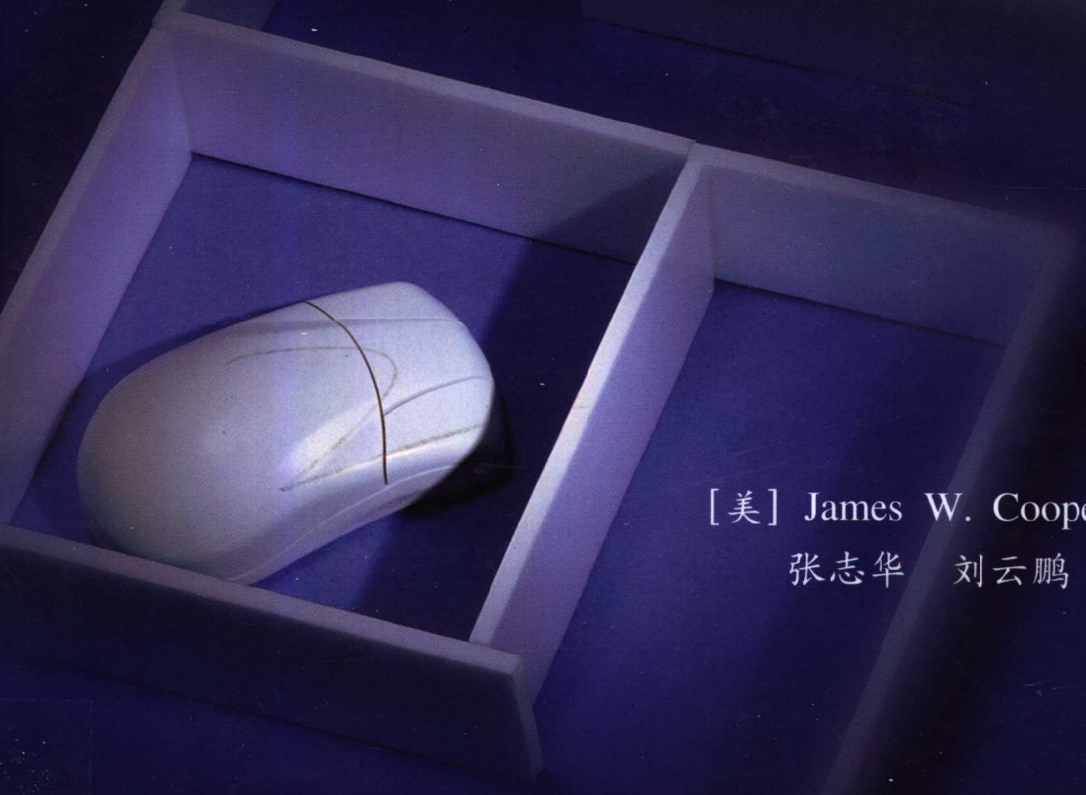
C#设计模式

C# Design Patterns: A Tutorial



[美] James W. Cooper 著
张志华 刘云鹏 等译

PEARSON Addison Wesley
PHEI
電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

C# 设计模式

C# Design Patterns: A Tutorial

[美] James W. Cooper 著

张志华 刘云鹏 等译

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内容简介

本书主要介绍如何用最常见的设计模式编写C#程序。全书分为四个部分，首先介绍了C#语言和面向对象程序设计的一般原则，可作为C#程序设计的快速入门教程；然后分别讲述了创建型模式、结构型模式和行为型模式。每一类设计模式又包括若干种具体模式，共有23种。在介绍每种模式时，给出了一个或多个应用该模式的示例，以便于理解，且这些示例都是能完全运行的程序，包含在随书附带的光盘中。此外，每一章还提供了UML图，用以说明类之间的关系。

本书适用于计算机及相关专业的本科生和研究生，对于软件开发人员也是一本很好的参考书。

版权贸易合同登记号：图字：01-2002-6038

图书在版编目（CIP）数据

C#设计模式/（美）库珀（Cooper, J. W.）著；张志华等译. -北京：电子工业出版社，2003.8
书名原文：C# Design Patterns: A Tutorial
ISBN 7-5053-8979-3

Ⅰ.C... Ⅱ.①库... ②张... Ⅲ.C语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2003）第067673号

责任编辑：陶淑毅
印 刷 者：北京兴华印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路173信箱 邮编：100036
经 销：各地新华书店
开 本：787 × 1092 1/16 印张：16.75 字数：482千字 附光盘一张
版 次：2003年8月第1版 2003年8月第1次印刷
定 价：33.00元（含光盘）

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077

译 者 序

本书介绍了C#设计模式，其中C#是Microsoft公司推出的新一代软件开发环境Microsoft.NET的核心语言，它的出现代表了程序设计语言下一步的发展方向；而设计模式是系统地获取一些软件开发专家的成功经验，提供“在一定的环境中解决某一问题的方案”。本书主要讲述的是，如何使用最常见的设计模式编写C#程序，以提高程序的可读性、可重用性和可维护性。

本书分为四个部分：第一部分介绍了C#语言和面向对象程序设计的一般原则；第二部分讲解了创建型模式（Creational Pattern），这一模式规定了创建对象的方式；第三部分介绍了结构型模式（Structural Pattern），该模式规定了如何组织类和对象；第四部分讲述了行为型模式（Behavioral Pattern），这一模式规定了对象之间交互的方式。全书一共介绍了23种设计模式，从第8章到第31章每章讨论一种模式，并给出一个或多个应用该模式的示例，这些示例都是能完全运行的程序。另外，每一章还包括一个UML图，用以说明类之间的关系。

本书的第一部分可作为C#程序设计的快速入门教程，另外在这一部分还介绍了UML，可供不熟悉UML的读者参考。对于第二部分至第四部分介绍的各种类型的设计模式，读者可以根据需要有选择地阅读。学习设计模式时，不要急于求成，对于某些难以理解的模式，多研究一下示例程序会很有帮助。

本书用C#示例程序讲解抽象的设计模式，使读者理解起来更容易。无论是有经验的软件开发人员，还是初学者，这都是一本理想的教材和参考用书。

全书主要由张志华、刘云鹏翻译，其他参与本书翻译工作的同志还有施运梅、管萍、李文虎、沈志刚、李树玲、刘广、张驰等。在此，也由衷地感谢所有支持我们、给我们提供帮助的人。

由于译者水平有限，难免有不当之处，恳请广大读者批评指正。

前　言

这是一本很实用的书，告诉读者如何用最常见的设计模式编写C#程序，同时本书也可以作为C#程序设计的快速入门教程。设计模式分成若干较短的章节来讨论，每章讨论一种设计模式，并给出一个或多个应用该模式的示例，这些示例都是能完全运行的可视化程序。每一章还包括了用以说明类之间关系的UML图。

本书并非“Gang of Four”所著的很有影响的“Design Patterns”（《设计模式》）一书的姊妹篇。而对于那些想要了解设计模式是什么以及如何把设计模式应用于工作中的人来说，此书却是一本不错的指南。读者没有必要先阅读“Design Patterns”，再阅读本书，但是，如果你事先读过那本书，也完全可以再读一读本书以获得更多的知识。

在本书中，读者会发现设计模式经常用在程序中对象的组织方式上，以使程序更加容易编写和修改。另外，通过熟悉设计模式，也会掌握一些用于讨论程序如何构建的有用词汇。

人们从不同方面开始欣赏设计模式，从高深的理论研究到日常的实践操作。最终，人们看到了这些模式的巨大力量，体会到了使用模式带来的诸多便利，于是设计模式融入到了工作中的方方面面。

全书以尽可能多的方式来描述模式，帮助读者理解抽象的概念。该书分成如下几个主要部分：概述，C#介绍和模式的讨论（这一部分又分创建型模式、结构型模式和行为型模式三部分内容）。

对于每一种模式，先给出简洁的描述，然后构建几个简单的程序示例。每一个示例都是可视化的程序，读者可以运行、检验程序，以使该模式的概念尽可能地具体化。所有的程序示例及其变化形式都在随书附带的光盘中，读者可以运行、修改这些程序，并查看改动所产生的效果。

示例中用到的每一个类都放在C#文件中，由于一个示例由多个C#文件组成，因此我们为每个示例提供一个C#项目文件，并把每个示例都放在一个单独的子目录中，从而避免了混乱。本书假设读者具备Visual Studio.NET环境，这个软件有各种版本，这里的示例代码是使用专业版开发的。

书中还穿插了许多用来说明设计模式的程序片段，帮助读者进一步理解这些模式。另外，还附加了这些程序的UML图，以说明类之间的关系。UML图说明了类及其继承结构，由框和箭头组成，其中箭头指向父类，后接虚线的箭头指向接口。如果读者还不熟悉UML，可以参看第6章提供的UML的简单介绍。书中所有的图表都是用WithClass 2000制作的，该软件的演示版包含在随书附带的光盘中。

学习完本书后，读者会熟练掌握设计模式的基础知识，并能将其运用于日常的C#程序设计工作中。

目　录

第一部分　C# 面向对象程序设计

第二部分 创建型模式

第三部分 结构型模式

第四部分 行为型模式

第一部分

C# 面向对象程序设计

本书的第一部分介绍了设计模式的概念、C#语言和面向对象程序设计的一般原则。具体讲解了C#语法和如何创建简单的程序，并讨论了类、对象、继承、接口及UML图的基础知识。学习完这一部分后，可以熟悉C#和面向对象程序设计，接下来就可以学习如何将设计模式运用于C#程序设计中了。

第1章　设计模式概述

也许读者正坐在电脑桌旁，凝视着天花板，勾画着如何编写一个新程序的功能部件。凭直觉，你也许知道必须做什么，要用到哪些数据和对象，但是潜意识里总觉得，会有一种更好的、更通用的方式来编写这个程序。

实际上，直到读者在脑子里建立了一个代码要完成什么操作和代码段之间如何相互作用的框架后，才有可能进行编码。框架建立得越完善，对问题的解决方案会越满意。如果没有掌握这种正确的方法，那么即使问题的解决方案很明显，恐怕也还要茫然一段时间。

我们凭感觉就知道，越是一流的解决方案就越容易复用，也越容易维护。即使读者是惟一可靠的程序员，一旦设计完一个相对优雅、没有暴露太多内部异常的解决方案后，也会觉得很安心。

计算机科学的研究人员开始欣赏设计模式的主要原因之一是，设计模式能够使解决方案既优雅简单，又可复用。术语“设计模式”对缺乏专业知识和经验的人来说有点儿正式，第一次遇到时会有些无所适从。事实上，设计模式仅仅是一些在项目之间和程序员之间复用面向对象代码的简便方法。设计模式背后的思想很简单：对通用的对象间的相互作用方式进行记录和编目（程序员经常发现这些对象间的相互作用方式很有用）。

一个常被引用的模式是 Smalltalk（Krasner and Pope 1988）的模型 - 视图 - 控制器（Model-View-Controller）框架，它出自关于程序设计框架的早期文献。这个模式把用户接口问题分成三个部分，如图 1.1 所示。图中“数据模型”包括程序的计算部分，“视图”表示用户界面，“控制器”定义用户和视图的交互方式。

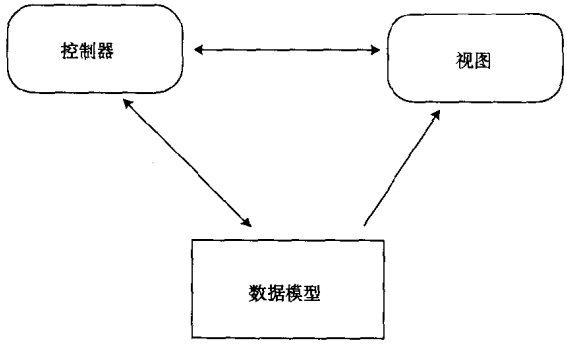


图 1.1　模型 - 视图 - 控制器框架

其中的每个部分都是一个独立的对象，每个对象用自己的规则处理数据。用户、GUI（图形用户界面）和数据间的通信应该认真控制，而这种功能的分离恰恰能很好地满足这一点。三个对象使用这样一组约束的连接相互通信，这就是一个功能强大的设计模式示例。

换言之，设计模式描述了对象如何进行通信才能不牵涉相互的数据模型和方法。保持这种独立性一直是一个好的面向对象（Object Oriented, OO）程序设计的目标，如果读者一直在努力使对象只涉及各自的业务，那么你很可能已经在使用某些通用的设计模式了。

设计模式开始变得越来越流行是在 20 世纪 90 年代初期，当时 Eric Gamma（1992）把模式合并在了 GUI 应用程序框架 ET++ 中。各种讨论和学术会议发展的巅峰是一本喻为“设计模式鼻祖”的

书的出版，即由 Gamma 等人（1995）创作的“Design Patterns：Elements of Reusable Software”。这本书通常被称为“Gang of Four”或“GoF”，它对想弄清楚如何使用设计模式的人产生了深远的影响，并成为销量空前的畅销书。书中描述了经常遇到又通用的23种模式，并对如何使用和何时使用这些模式加以论述。我们在本书中把这本具有开创性的书称为“Design Patterns”（《设计模式》）。

自从“Design Patterns”一书出版以来，相继又有许多其他有价值的书出版。其中一本关系较密切的书是“The Design Patterns Smalltalk Companion”（Alpert，Brown，and Woolf 1998），它以Smalltalk的观点介绍了同样的23种模式。我们在本书中称之为“Smalltalk Companion”。最近又出版了“Java Design Patterns：A Tutorial”和“Visual Basic Design Patterns：VB6 and VB.NET”，这两本书使用各自的语言阐明了同样的23种模式（Cooper 2000，2002）。

1.1　定义设计模式

我们总在谈论做各种事情的方法：工作、爱好和家庭生活等各个方面，而且在不断地重复一些模式。

- 发粘的小圆面包与晚餐上的小圆面包看起来很像，但我在上面加了红糖和坚果。
- 她门前的花园和我的很像，但我在花园里种了玫瑰。
- 这张茶几和那张样式上很像，但这张茶几用门代替了抽屉。

在程序设计中也会有同样的事情：告诉一个同事我们怎样完成了一个包含相当多技巧的程序，这样他就不用从头开始再做一次。通过维持对象的独立性，我们找到了对象通信的有效方式。

随着该领域文献的增加，出现了一些有用的设计模式定义。

- 设计模式是对读者经常遇到的设计问题的可再现的解决方案（The Smalltalk Companion）。
- 设计模式建立了一系列描述如何完成软件开发领域中特定任务的规则（Pree 1995）。
- 设计模式更关注于复用可重复出现的结构设计方案，而框架注重于具体设计和实现（Coplien and Schmidt 1995）。
- 模式提出了一个发生在特定设计环境中的可重复出现的设计问题，并提供了解决方案（Buschmann et al. 1996）。
- 模式识别并确定类和实例层次上或组件层次上的抽象关系（Gamma，Johnson，and Vlissides，1993）。

尽管找到与建筑学、家具制造和逻辑学的相似之处是很有帮助的，但设计模式不仅仅局限于对象的设计，还涉及到对象之间的相互作用。有一种观点就是将某些模式看做通信模式。

另外一些模式不但涉及到对象通信，还涉及到针对对象继承和包含的策略。正是简单而优雅的设计及相互作用的方式使众多的设计模式变得如此重要。

设计模式可存在于多个层次中，从最底层的专门解决方案到很普遍的系统问题。目前已提出了数百个模式，在文献中和各级会议上都有讨论。某些模式应用很广，也有些作者将模式行为归成几类，每类适用一种问题（Kurata 1998）。

很明显，读者不需要记住这些模式。实际上，大部分模式是被发现而不是被设计出来的。寻找这些模式的过程称为“模式挖掘”，它本身就值得写一本书。

“Design Patterns”一书中包含的23种设计模式都具有几个已知的应用，并且有一定的普遍性，很容易跨越应用领域解决不同的问题。

本书作者把这些模式分成三类：创建型、结构型和行为型。

- 创建型模式是创建对象而不是直接实例化对象，这会使程序在判断给定情况下创建哪一个对象时更为灵活。
- 结构型模式可以将一组对象组合成更大的结构，例如复杂的用户界面或报表数据。
- 行为型模式定义系统内对象间的通信，以及复杂程序中的流程控制。

在后面的章节里，会看到用C#描述的这些模式，对23种模式中的每一种，我们都至少提供一个完整的C#程序。读者可以分析提供的代码片段，也可以运行、编辑和修改所附光盘上的完整程序。在每一种模式描述完之后，都会给出光盘中相应程序的列表。

1.2 学习过程

我们发现，不管使用哪种语言，学习设计模式都需要三个阶段：

1. 接受。
2. 认可。
3. 领会。

首先，你要接受设计模式对自己的工作很重要这一前提；接下来，会认识到需要学习设计模式，这样才能知道什么时候需要使用设计模式；最后，要充分消化有关模式的细节，这样才能知道哪一种模式能帮助自己解决给定的问题。

对某些幸运的人来说，设计模式是显而易见的工具，这些人只需阅读模式的概要部分就能抓住它们的本质。而对于大多数像我们这样的人来说，了解一种模式后会有一个较慢的感悟阶段，然后当看到能将这些模式用在工作中时，才会恍然大悟。本书通过提供一系列完整的、可运行且读者可以进行试验的程序，带你走到最后一步的领会阶段。

“Design Patterns”中的示例很简洁，多采用C++描述，有时候是用Smalltalk描述。如果选用另外一种语言工作，那么采用选择的语言来描述模式的示例也很有帮助。本书力图满足C#程序员的这一需求。

1.3 学习设计模式

熟悉设计模式有几种不同的方法。在每种方法中，都应该按顺序阅读本书和“Design Patterns”一书，或者以相反顺序阅读。为了完整性，我们还强烈推荐读者阅读“Smalltalk Companion”一书，因为它对每种模式都有不同的描述。并且，还有许多学习和讨论设计模式的网站，读者可以去浏览学习。

1.4 关于面向对象方法

使用设计模式的根本原因是为了保持类之间的隔离，防止相互之间了解太多的内容；另一个重要的原因是，使用这些模式可以帮助读者避免重复开发，能用其他程序员容易理解的术语描述你的程序设计方法。

有很多策略能让OO程序员达到分离类的目的，其中包括封装和继承。几乎所有具备OO能力的语言都支持继承。一个继承父类的类能访问父类中的所有方法和所有非私有变量，然而，如

果以一个功能完整的类开始继承层次结构，可能会过多地限制住自己，也会给特定方法的实现带去累赘。因此，“Design Patterns”建议应遵循下列设计原则：针对接口编程，而不是针对实现编程。

用更简洁的形式来表示这一原则，就是在任何类层次结构的顶端，定义的是一个抽象类或一个接口，它没有实现方法，但定义了该类需要支持的方法。这样在所有的派生类中，就会有更大的自由度去实现这些方法，能最大限度地满足你的要求。

另外一个需要考虑的重要概念是对象组合，这是一种可包含其他对象的对象结构，即把几个对象封装在另一个对象中。许多初级OO程序员使用继承去解决每个问题，而随着你开始编写比较复杂的程序，就会体会到对象组合的优点。新的对象拥有一个最适合于要完成的目标的接口，而不是拥有父类中的所有方法，这就导出了“Design Patterns”一书提出的第二个重要原则：优先使用对象组合，而不是继承。

1.5　C# 设计模式

本书讨论了“Design Patterns”一书提出的23种模式，每种模式至少给出了一个可运行的程序示例。所有的程序都有可视化界面，以使程序理解起来更容易，并且都使用了类、接口和对象组合，但程序本身保持了必要的简单性，这样编码就不会掩盖所描述模式的优雅特性。

尽管C#是我们的目标语言，但这并非一本专门讲解C#语言的书。有许多的C#特性书中都没有涉及，但本书覆盖了C#的大部分重要内容。读者会发现，对于采用C#进行面向对象程序设计的人来说，这是一本相当有用的指导书，书中对如何使用C#进行编程也给出了很好的概述。

1.6　本书的组织结构

第1章到第7章介绍了设计模式的概念、C#语言的基础知识，以及面向对象程序设计和继承的思想，还讲解了如何用UML类图表示对象。

然后研究了23种设计模式，将它们分类进行编目：创建型模式、结构型模式和行为型模式。许多模式都是在某种程度上独立存在的，但我们也会不时地用到已经讨论过的模式。例如，在介绍完工厂和命令模式后，会广泛地使用这两种模式；在介绍了中介者模式后也会多次使用该模式。我们还将备忘录模式用到了状态模式中；将职责链模式用到了解释器模式的讨论中；将单件模式用到了享元模式的讲解中，但我们在正式介绍某种模式之前决不会去使用该模式。

本书利用当前讨论的模式来介绍C#的新特性。例如，我们在适配器模式和桥接模式中介绍了ListBox，DataGrid和TreeView；在抽象工厂中讲解了如何绘制图形对象；在解释器模式和组合模式里介绍了Enumeration界面，并研究了格式转换；在单件模式中用到了异常；在外观模式中讨论了ADO.NET数据库连接；在代理模式中，教授读者如何使用C#计时器。

整个.NET系统的设计目标是，使基于Web的客户－服务器交互操作更加灵活，而本书则关注于一般的面向对象程序设计问题，而不是如何编写基于Web的系统。我们涵盖了C#程序设计的核心内容，并给出了一些简单的示例，说明如何在设计模式帮助下编写更好的程序。

第2章　C#语言的语法

C#具有功能强大的现代语言的所有特性。如果读者熟悉Java，C或C++，就会发现多数C#的语法都是非常熟悉的。如果使用过Visual Basic或类似语言，那么应该阅读本章来看看C#与VB的区别，你会很快发现在VB.NET上执行的主要操作都和C#上的相似。

C#和VB的两个主要区别是：(1) C#是区别大小写的（大多数语法使用小写）；(2) C#中的每一条语句都以分号（;）结束，这样，C#语句就不必限制在一行内，而且不需要续行符。

在VB中，我们可以这样写：

```
y = m * x + b       'compute y for given x
```

也可以这样写：

```
Y = M *X + b       'compute y for given x
```

可以将它们看成是一样的。不管大写还是小写，对于变量Y，M和X都是一样的。然而，在C#中，大小写至关重要，如果我们这样写：

```
y = m * x + b;         // all  lowercase
```

或

```
Y = m * x + b;        // Y differs from y
```

则意味着Y和y是两个不同的变量。尽管刚开始看来这似乎有些笨拙，但以后会发现区别大小写是很有用的。例如，程序员经常用大写的符号表示常数：

```
Const PI = 3.1416  As Single in VB
const float PI = 3.1416;     //in C#
```

C#中的修饰符const的含义是：被命名的值是一个常数，不能更改。

程序员有时也用大小写混合的形式定义数据类型，用小写形式定义这种数据类型的变量。

```
class Temperature {     // begin definition of
                        // new data type
Temperature temp;       // temp is of this new type
```

我们在后面的章节中会更具体地介绍类的知识。

2.1　数据类型

表2.1给出了C#的主要数据类型。注意，基本类型的长度与计算机或操作系统的类型无关。C#中的字符是16位的宽度，可以表示非拉丁语言中的所有字符。它使用一种叫双字节码的字符编码系统，其中定义了绝大多数可书写语言的数以千计的字符。可以使用通常的方式在不同的变量类型之间转换。

表2.1　C#中的数据类型

bool	true或false
byte	8位无符号整数
short	16位整数
int	32位整数
long	64位整数
float	32位浮点数
double	64位浮点数
char	16位字符
string	多个16位字符

- 宽度窄（字节数少）的数据类型可以直接赋给较宽的数据类型，并能自动转换为新类型。如果y是浮点类型而j是整数类型，则可以用下列方式把一个整数类型转换成浮点类型。

```
float y = 7.0f;          //y is of type float
int j = 5;               //j is of type int
y = j;                   //convert int to float
```

- 可以通过强制转换把较宽的类型（字节较多）缩减成较窄的类型。这时，需要把数据类型的名称放在圆括号内，并把它放在要转换的数据前面。

```
j = (int)y;          //convert float to integer
```

也可以写出这样的合法语句，它包含的强制转换可能会失败。

```
float x = 1.0E45;
int k = (int) x;
```

如果强制转换失败，程序执行时会出现一个异常错误。

布尔型变量只能接受保留字true和false所表示的值。布尔型变量通常接受比较操作和其他逻辑操作的结果作为它的值。

```
int k;
boolean gtnum;

gtnum = (k > 6 );          //true if k is greater than 6
```

与C或C++不同，不能将数值型数据赋值给布尔型变量，也不能在布尔类型和其他数据类型之间转换。

2.2　数值与字符串间的转换

可以使用Convert方法将数值转换成字符串或将字符串转换成数值。在开发环境中只要键入“Convert”和一个圆点，系统就会提供一系列可用的方法，在其中能找到正确的那一个。

```
string s = Convert.ToString (x);
float y = Convert.ToSingle (s);
```

注意，“Single”代表一个单精度浮点数。

数值型对象也提供了各种格式化方法来指定小数位数。

```
float x = 12.341514325f;
string s = x.ToString ("###.###");        //gives 12.342
```