
Broadview
www.broadview.com.cn
C#
二维三维图形绘制
工程实例宝典
C#图形设计的技术宝典
C#图形处理的良师益友
伍逸 著
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

C#
二维三维图形绘制
工程实例宝典

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

内 容 简 介

本书全面详细地阐述了C#图形设计技术，专门列举了许多C#二维三维图形绘制的工程实例，可称为C#图形处理方面的一本宝典。

本书分为五个部分共10章，第一部分介绍C#基本的数据类型和图形基础技术，第二部分讲述二维图形的基本算法，第三部分介绍三维图形的相关知识及各种三维图形的实现，第四部分介绍C#中应用微软Office的Excel实现各种二维及三维图形，第五部分讲述实现文件的相关知识。

本书适用于从事图形图像处理的工程技术人员，也可作为高等院校计算机和计算机应用相关专业的教学参考用书。

图书在版编目（CIP）数据

C#二维三维图形绘制工程实例宝典 / 伍逸著. —北京：电子工业出版社，2010.12
ISBN 978-7-121-12273-6

Ⅰ.①C… Ⅱ.①伍… Ⅲ.①C语言—程序设计 Ⅳ.①TP312

中国版本图书馆CIP数据核字（2010）第222560号

策划编辑：郭 立 袁金敏
责任编辑：郭 立
特约编辑：顾慧芳
印 刷：北京天宇星印刷厂
装 订：三河市皇庄路通装订厂
出版发行：电子工业出版社
北京市海淀区万寿路173信箱 邮编100036
开 本：787×1092 1/16 印张：41.5 字数：760千字
印 次：2010年12月第1次印刷
印 数：3500册 定价：89.00元（含光盘1张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前言

由于实际工作的需要，大量地用到C#图形设计方面的技术，但在查寻了许多有关C#图形设计方面的资料后，发现竟然没有一本书籍或一份资料，详细而清晰地介绍相关的知识和技术，而是散布于不同的书籍和资料中，且大多语焉不详，一笔带过。于是产生了写这样一本书的想法，经过这段时间的日夜辛劳，终于有所回报。

内容

本书层次是这样安排的，全书分五个部分共10章，第一部分为第1章到第3章，介绍了C#的基本数据类型，数组类型，图形基础包括画笔、刷子、位图及双缓存技术等，坐标体系的定义，坐标体系的分类。这些知识是学习其他各章节的必备基础知识。

第二部分为第4章到第6章，讲述了二维图形的基本算法，包括矩阵、转换等，二维折线图形及特效二维图形包括面积图、股票图等的绘制。

第三部分为第7章和第8章，介绍了三维图形的相关知识及各种三维图形如网格图、曲面图等的实现。

第四部分为第9章，介绍了C#中应用微软Office的Excel实现各种二维及三维图形。

第五部分为第10章，讲述了实现文件的相关知识。

特点

本书的每一部分都是从实际例子入手来讲述图形的实现，对涉及的类及函数给出了详细的解释。具体特点如下：

- 讲述的所有技术，即可在Visual Studio 2005中实现，也可在Visual Studio 2008中实现，无须任何更改；
- 从实际的软件模块入手，几乎所有的示例程序读者都可仅做细微的修改或不做修改即可应用于自己的图形软件中；
- 所有的示例程序均应用纯粹的C#语言实现，没有用到任何第三方控件或函数库，

也没有用到 DirectX 或 OpenGL；

- 对于书中相关的数学知识给出了详细的解释。

致谢

在本书的写作过程中，始终得到了笔者家人的支持，没有你们的支持和爱，我将很难完成。另外，本书的出版得到了电子工业出版社的大力支持，袁金敏和顾慧芳两位编辑付出了大量的劳动，在此一并衷心感谢！

如果你也曾经在 C#图形处理方面遭遇到一些瓶颈，或是想加快学习的速度，或是解决工作的困难，那我想这本书可以对你有所帮助。有些时候，只是朋友间的一个简单提示却豁然开朗的感觉真的很好。

作　者

2010 年 8 月

目录

第一部分　C#的基本数据类型、数组类型和图形基础

第二部分　二维图形的基本算法

第三部分 三维图形的相关知识及三维图形的实现

第四部分 C#中应用微软 Office 的 Excel 实现各种二维及三维图形

第五部分 实现文件的相关知识

第一部分　C#的基本数据类型、数组类型和图形基础

第 1 章　C#语言基础

【本章导读】

任何一门语言，其语言基础知识都是非常重要的，只有掌握了其基础知识才能灵活运用语言解决实际问题。本章主要介绍：

- C#数据类型及转换；
- 类的创建与使用；
- 类的继承和多态；
- 接口创建与实现；
- 委托与事件。

1.1　数据类型

对于程序中的每一个用于保存信息的量，使用时都必须声明它的数据类型，以便编译器为它分配内存空间。C#的数据类型分为值类型（Value Type）、引用类型（Reference Type）和指针类型（Pointer Type）三大类。值类型包括简单类型（Simple Type）、结构类型（Structure Type）和枚举类型（Enum Type）。引用类型包括类类型（Class Type）、接口类型（Interface Type）、委托类型（Delegate Type）和数组类型（Array Type）。指针类型只能用于不安全模式。

值类型和引用类型有区别，值类型变量直接存储它的数据内容，当把一个值赋给一个值类型时，该值实际上被拷贝了。引用类型变量不存储实际数据内容，而是存储对实际数据的引用，当把一个值赋给一个引用类型时，仅仅是拷贝引用，实际的值仍然保留在原来的内存位置，只是赋值后有两个不同的变量指向这个实际的值。

1.1.1　简单类型

C#提供了称为简单类型的预定义结构类型集，简单类型通过保留字标识，而这些保留字只是 System 命名空间中预定义结构类型的别名。C#简单类型的保留字与预定义结构类型对应关系如表 1-1 所示。C#简单类型分为整数类型、布尔类型、字符类型、浮点类型和 decimal 类型。

表 1-1 保留字与预定义结构类型对应关系

保 留 字	预定义结构类型
sbyte	System.SByte
byte	System.Byte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.Uint64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

预定义结构类型和简单类型是等价的。例如，可用下列两种声明中的一种来声明一个整数变量：

```
int a = 10;
System.Int32 a = 10;
```

由于预定义结构类型是对象，因此两者都具有成员。如，int 具有在 System.Int32 中声明的成员及从 System.Object 继承的成员，允许使用下面的语句：

```
 int I = int.MaxValue;          // 得到整型量的最大值，是 Int32 的属性之一
string s = I.ToString();        // 把整型量转换成字符串，是 Int32 的一个实例方法
string t = 123.ToString();
```

1．整数类型

C#有 9 种整数类型：sbyte、byte、short、ushort、int、uint、long、ulong 和 char。取值范围如表 2-2 所示。

表 1-2 整数类型的含义和取值范围

类 型	含 义	取 值 范 围
sbyte	有符号 8 位整数	–128～127
byte	无符号 8 位整数	0～255
short	有符号 16 位整数	–32 768～32 767
ushort	无符号 16 位整数	0～65 535
int	有符号 32 位整数	–2 147 483 648～2 147 483 647
uint	无符号 32 位整数	0～4 294 967 295
long	有符号 64 位整数	–9 223 372 036 854 775 808～9 223 372 036 854 775 807
ulong	无符号 64 位整数	0～18 446 744 073 709 551 615
char	无符号 16 位整数	0～65 535

2．浮点类型

C#支持两种浮点类型：float 和 double。它们用 32 位单精度和 64 位双精度 IEEE 754 格式来表示。Float 类型取值范围为 $1.5\times10^{-45}\sim3.4\times10^{38}$，精度为 7 位。double 取值范围为 $5.0\times10^{-324}\sim1.7\times10^{308}$，精度为 15 位或 16 位。

3．decimal 类型

decimal 类型是用于财务和货币计算的 128 位数据类型。decimal 类型比浮点类型具有更高的精度和更小的范围。decimal 类型的范围是 $1.0\times10^{-28}\sim7.9\times10^{28}$,精度为 28 或 29 个有效数字。当给 decimal 变量赋值时，使用 m 后缀来表明它是一个 decimal 类型。例如：

```
decimal myMoney = 100.3m;
```

整型可以被隐式转换为 decimal 类型，其计算结果为 decimal 类型。可用整型初始化 decimal 型变量。如

```
decimal myMoney = 1000;
```

浮点类型和 decimal 类型的精度和表示范围均不相同，它们之间的转换可能会产生溢出异常或精度损失，因此浮点类型和 decimal 类型之间不存在隐式转换。例如：

```
decimal myMoney = 100.34m;
double x = (double) myMoney;
myMoney = (decimal) x;
```

4．布尔类型

布尔类型表示布尔逻辑量。在 C#中，布尔逻辑量只有 true 和 false 两个值。可以将布尔值赋给布尔形变量，也可以将值为布尔类型的表达式赋给布尔形变量，例如：

```
bool var = true;
bool var = (v > 0 && v < 10);
bool var = (20 > 30);
bool var = (c == s);
```

在 C 和 C++中，布尔类型的值和 int 类型的值可相互转换，即 false 等效于零值，true 等效于非零值。在 C#中，布尔型和其他类型之间不存在标准转换。布尔类型与整型截然不同，不能用布尔值代替整型值，反之亦然。例如，下列 if 语句在 C#中是非法的，而在 C++中是合法的：

```
int b = 30;
if(b)
  cout << "The value of b is true";
```

在 C#中，正确的用法是通过显式地将整数或浮点值与零进行比较；或者是显式地将对象引用与 null 进行比较来完成的，例如：

```
int b = 30;
if (b != 0)
   System.Console.WriteLine("The value of b is true");
```

或

```
if (0 != b)
    System.Console.WriteLine("The value of b is true");
```

5. 字符类型

除了数字以外，计算机处理的信息主要是字符。C#字符类型采用 Unicode 字符集，一个 Unicode 标准字符长度为 16 位，它允许用单个编码方案表示世界上使用的所有字符。

可以按如下方法直接给一个字符变量赋值：

```
char c = 'c';
```

此外，也可以通过十六进制转义符（前缀“\x”加十六进制数字）或 Unicode 转义符（前缀“\u”加十六进制数字）给字符变量赋值，例如：

```
char c = '\x0067';
char c = '\u0067';
```

不存在从其他类型到 char 类型的隐式转换。对字符变量使用整数进行赋值和运算是不允许的。但是，字符型和整型之间可以进行显式转换，例如：

```
char c = (char)10;
int d = (int) 'j';
```

C#使用 Unicode 转义符，表 1-3 列出了各种转义符。

表 1-3 转义符

转 义 符	含 义	Unicode 编码
\'	单引号	0x0027
\"	双引号	0x0022
\\	反斜杠	0x005C
\0	空	0x0000
\a	警报	0x0007
\b	退格符	0x0008
\f	换页符	0x000C
\n	换行符	0x000A
\r	回车	0x000D
\t	水平制表符	0x0009
\v	垂直制表符	0x000B

1.1.2 结构类型

上面介绍的都是一些简单类型，只有这些数据类型还不够，有时还需要更复杂的数据类型，把多个不同类型的数据组合到一起，以便使用。结构就是这些相关联的不同类型数据的组合，它可以构建复杂的数据结构，可以声明常数、字段、方法、属性、索引器、运

算符、实例构造函数、静态构造函数和嵌套类型。这看起来与类很相似，都表示可以包含数据成员和函数成员的数据结构。但是它们是有区别的，结构类型是一种值类型，而类类型是一种引用类型。

结构主要用于创建小型的对象以节省内存，如：复数、坐标系中的点或字典中的“键-值”对都是结构的典型示例。这些数据结构的关键之处在于，它们只有少量数据成员，不要求使用继承或引用标识。示例 Example1_1_1,定义一个表示矩形的 C#结构。

[示例 Example1_1_1] 定义一个矩形结构的代码如下：

```
 using System;
//Define a struct
struct Rectangle
{
 public int x,y;                                    // 定义矩形的左上角坐标
 public int width,height;                           // 定义矩形的宽和高
 public Rectangle(int a,int b, int w, int h)
 {
      x = a;
      y = b;
      width = w;
      height = h;
 }
}
class TestStruct
{
  public static void Main()
 {
      Rectangle myRect;

      myRect.x = 20;
      myRect.y = 30;
      myRect.width = 200;
      myRect.height = 300;

      Console.WriteLine("My Rectangle:");
      Console.WriteLine("x = {0}, y = {1}, width = {2}, height = {3} ",myRect.x, myRect.y, myRect.width,
myRect.height);
  }
}
```

输出结果为

```
My Rectangle:
x = 20, y = 30, width = 200, height = 300
```

1.1.3 枚举类型

枚举类型是自 System.Enum 派生的一种独特的值类型，它用于声明一组命名的常数。每种枚举类型均有一种基础类型，此基础类型可以是除 char 类型以外的任何类型。

枚举元素的默认基础类型为 int。默认情况下，第一个元素的值为 0，后面每个枚举元素的值依次递增 1。例如：

```
enum WeekDay{ Sun , Mon , Tue , Wed , Thu , Fri , Sat }
```

在此枚举中，Sun 的值为 0，Mon 为 1，Tue 为 2，依此类推。也可以直接给枚举元素赋值来改变这种默认情况，而且不同元素的值可以相同。如：

```
enum WeekDay{ Sun = 1 , Mon , Tue , Wed = Sun , Thu , Fri , Sat }
```

在此枚举中，强制第一个枚举元素 Sun 的值为 1，Mon 为 2，Tue 为 3，而 Wed 又强制为 1，Thu 为 2，依此类推。枚举元素 Sun 和 Wed、Mon 和 Thu、Tue 和 Fri 的值相同。

如果枚举元素的数据类型不是 int 型，则可以如下方式进行声明。

```
enum Color : long { Red , Green , Blye };
```

[示例 Example1_1_2] 使用枚举

```
using System;
public class TestEnum
{
    enum Range : long{ Max = 2147483648L , Min = 255L }
    public static void Main()
    {
        long a = (long) Range.Max;
        long b = (long) Range.Min;
        Console.WriteLine("Max = {0} , Min = {1}", a , b);
    }
}
```

下面再看一个例子：

```
enum Color { Red = Green , Green , Blue }
```

枚举元素 Red 的值由 Green 决定，而枚举元素 Green 的值又由 Red 决定，从而形成一个循环，这将产生错误。

1.1.4 数组类型

数组类型是由抽象基类 System.Array 派生的引用类型，它代表一组相同类型变量的集合，其中的每一个变量称为数组的元素。数组元素可以为任意类型，包括数组类型。对数组元素的访问是通过数组下标来实现的。C#数组的下标从 0 开始，即第一个元素对应的下标是 0，以后元素下标逐个递增。C#的数组定义要注意如下的事项。

- 在声明一个数组时，方括号必须跟在类型后面，而不能跟在变量名后面，例如："int[] color;" 不能写成 "int color[];"。
- 可以不指定数组的大小，这样可以指定任意长度的数组，例如：

```
int[] color;。
```

当然，也可以指定数组的长度，如：

```
int[5] color;
```

- 可以声明包含数组的数组，即交错数组。

在 C#中，支持的数组包括：一维数组、多维数组（矩形数组）和数组的数组（交错数组）。

1．一维数组和多维数组

声明一个由 3 个整型元素组成的一维数组如下。

```
int[] a = new int[3];
```

为每个数组元素赋值，以完成初始化。

```
a[0] = 1;
a[1] = 2;
a[2] = 3;
```

声明一个 2 行 2 列的二维数组如下。

```
int[,] a = new int[2,2];
```

为每个数组元素赋值，以完成初始化。

```
a[0,0] = 1;
a[0,1] = 2;
a[1,0] = 3;
a[1,1] = 4;
```

声明一个三维（3、2 和 2）数组如下。

```
int[,,] b = new int[3,2,2];
```

可以在声明数组时直接将其初始化，这时不需要指明数组的长度，也可以指明数组的大小，如：

```
// 声明一个含 3 个元素的一维整型数组
int[] b = new int[]{1 , 2 , 3};
int[] b = new int[3]{1 , 2 , 3};
int[] b = {1 , 2 , 3};
// 含 3 个元素的一维字符串数组
string[] c = {"one" , "two" , "three"};
string[] c = new string[3] {"one" , "two" , "three"};
// 含 6 个元素的二维整型数组
int[,] c = new int[,]{{1 , 2} , {3 , 4} , {5 , 6}};
// 含 6 个元素的二维字符串数组
string[,] d = new string[,]{{"one","two"},{"three","four"},{"five","six"}};
string[,] d = {{"one","two"},{"three","four"},{"five","six"}};
```

也可以先声明一个数组变量，然后再初始化，这时必须使用 new 运算符。例如：

```
int[] d;                          // 先声明一维数组变量
d = new int[]{1, 2, 3};           // 再初始化
d = {1, 2, 3};                    // 错误，没有使用 new 运算符
int[,] e;                         // 先声明二维数组变量
```

```
e = new int[,]{{1,2},{3,4},{5,6}};          //再初始化
```

[示例 Example1_1_3] 使用多维数组

```
using System;
class TestMutiArray
{
    public static void Main()
    {
        string[] course = {"C#","Data Structure","Software engineering"};
        Disp(course);
    }
    static void Disp(string[] arr)
    {
        for(int i=0; i<arr.length; i++)
        {
            Console.WriteLine("course[{0}] = {1}", i, arr[i]);
        }

    }
}
```

输出结果是

```
course[0] = C#
course[1] = Data Structure
course[2] = Software engineering
```

2. 交错数组

交错数组是指数组元素又是一个数组，即数组的数组。

声明一个由 3 个元素组成的一维数组，其中每个元素又是一个一维整型数组，如下：

```
int[][] a = new int[3][];
```

必须初始化 a 的元素后才可以使用它。初始化元素如下所示：

```
a[0] = new int[3];
a[1] = new int[2];
a[2] = new int[2];
```

每个元素又是一个一维整型数组。第一个元素 a[0]是由 3 个整数组成的一维数组，第二个元素 a[1]是由 2 个整数组成的一维数组，而第三个元素 a[2]是由 2 个整数组成的一维数组。

也可以直接初始化数组元素，在这种情况下不需要数组大小，例如：

```
a[0] = new int[] {1, 3, 5};
a[1] = new int[] {0, 2};
a[2] = new int[] {10, 20};
```

也可以在声明数组时将其初始化，例如：

```
int[][] a = new int[][]
```