



C#2010 编程基础及在运动控制卡上的应用

左 力

2014.4.29.完稿

目 录

第一讲：C#2010 概述及基础.....	2
第二讲：C#2010 编程语言.....	20
第三讲：C#2010 在运动控制卡上的应用.....	26
第四讲：C#2010 绘图、多线程编程方法.....	36
第五讲：C#2010 高速采样的方法.....	49
第六讲：C#2010 程序生成安装软件.....	55

第一讲：C#2010 概述及基础

一. 为什么要用C#?

1. Windows XP将会在2014年4月8日正式“退休”，届时微软将不会再为该系统提供每周的安全补丁更新和其他的技术支持。即继续用XP系统有安全隐患。
2. 微软建议大家使用Win8、Win7。（2013年10月17日，微软正式推出Windows 8.1。Win8太新，对硬件要求也高，只有64位系统。现在用Win8为时尚早。）

在Win7下，使用什么编程软件好？

继续用VB6.0。但VB6.0在Win7下兼容性不好、慢，且太老（1998年上市）、功能差；
改用VB2010。但VB2010和VB6.0差别很大，要学新东西。

用C#、VC。会不会太难？

VB2010、C#、VC，选用哪一个为好？先调查一下，大家都在用什么软件。

2012年底的Tiobe编程语言排行榜图1所示：

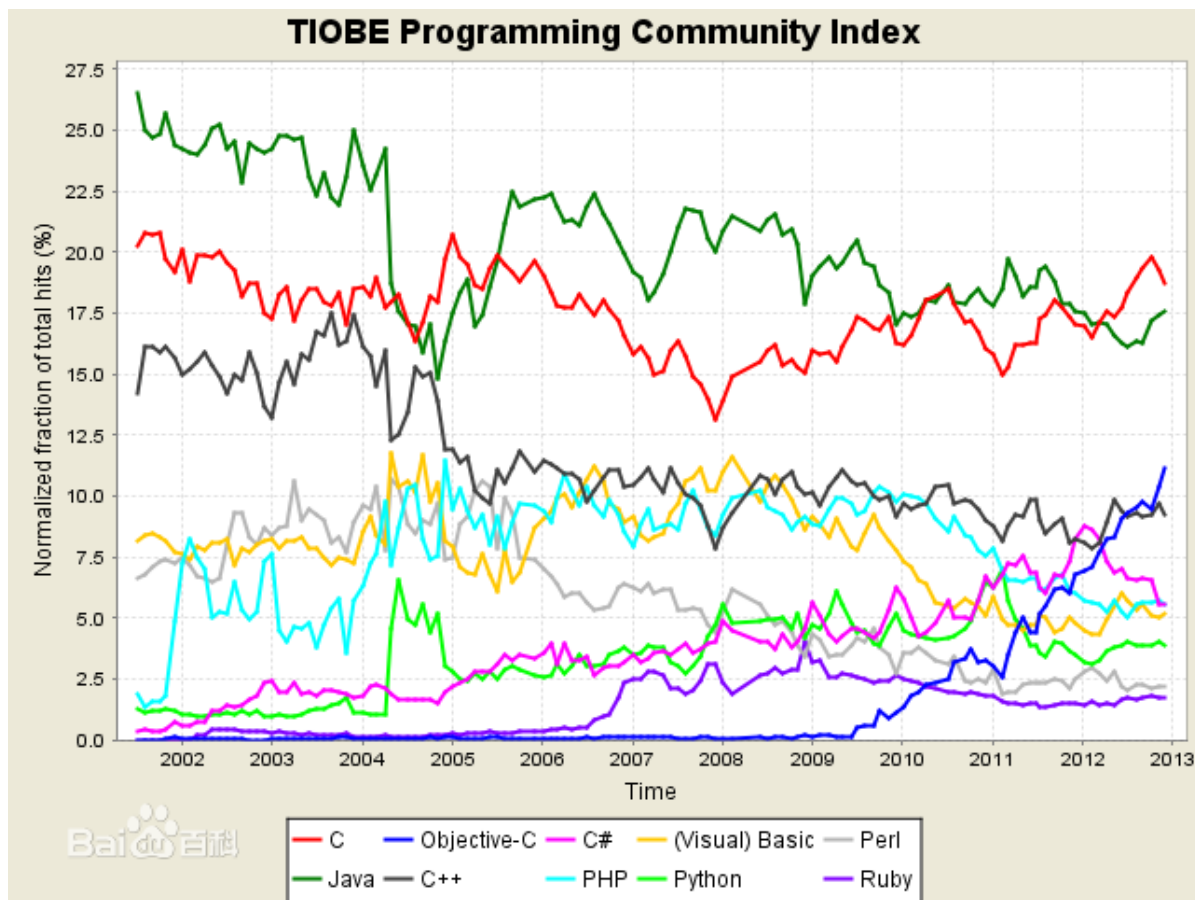


图1 2012年底的Tiobe编程语言排行榜

2013 年 8 月编程语言排行榜如图 2 所示。

Position Aug 2013	Position Aug 2012	Delta in Position	Programming Language	Ratings Aug 2013	Delta Aug 2012	Status
1	2	↑	Java	15.978%	-0.37%	A
2	1	↓	C	15.974%	-2.96%	A
3	4	↑	C++	9.371%	+0.04%	A
4	3	↓	Objective-C	8.082%	-1.46%	A
5	6	↑	PHP	6.694%	+1.17%	A
6	5	↓	C#	6.117%	-0.47%	A
7	7	=	(Visual) Basic	3.873%	-1.46%	A
8	8	=	Python	3.603%	-0.27%	A
9	11	↑↑	JavaScript	2.093%	+0.73%	A
10	10	=	Ruby	2.067%	+0.38%	A
11	9	↓↓	Perl	2.041%	-0.23%	A
12	15	↑↑↑	Transact-SQL	1.393%	+0.54%	A
13	14	↑	Visual Basic .NET	1.320%	+0.44%	A
14	12	↓↓	Delphi/Object Pascal	0.918%	-0.09%	A-
15	20	↑↑↑↑	MATLAB	0.841%	+0.31%	A-
16	13	↓↓↓	Lisp	0.752%	-0.22%	A
17	19	↑↑	PL/SQL	0.751%	+0.14%	A
18	16	↓↓	Pascal	0.620%	-0.17%	A-
19	23	↑↑↑↑	Assembly	0.616%	+0.11%	B
20	22	↑↑	SAS	0.580%	+0.06%	B

图2 2013年8月Tiobe编程语言排行榜

- Java，1995年由SUN公司正式推出，免费！具有卓越的通用性、高效性、平台移植性和安全性，广泛应用于个人PC、数据中心、游戏控制台、科学超级计算机、移动电话和互联网。(2010年Oracle公司收购了SUN)
- Objective-C，是扩充C的面向对象编程语言。它主要用于Mac OS X和GNUstep这两个使用OpenStep标准的系统。
- PHP（Hypertext Preprocessor的缩写，中文名：“超文本预处理器”）是一种通用开源脚本语言。语法吸收了C语言、Java和Perl的特点，入门门槛较低，易于学习，使用广泛，主要适用于Web开发领域。
- Python/'paɪθən/是一种解释型、面向对象、动态数据类型的高级程序设计语言。自从1991Python语言诞生至今，它逐渐被广泛应用于处理系统管理任务和Web编程。免费、开源。

数据表明：用C#的人比用VB的人多近一倍。

二. C#的历史：

C#读作C Sharp。符号#有2种解释：

#在五线谱中读作Sharp，是“升半个音”的意思。这里用#有对C提升之意。

C → C++ → C++++，为了方便写成了C#。



1996年原Broland公司的首席研发设计师安德斯 海尔斯伯格(Anders Hejlsberg)加入微软,开始开发Visual J++。Visual J++很快由1.1版本升级到6.0版。VJ++编译出来的虚拟机械码的执行效率不但比任何Java开发工具快,在某些方面甚至比原生的Windows开发工具,如: Delphi、VB、甚至是VC++效率还高。

SUN公司认为Visual J++ 违反了Java开发平台的中立性,对微软提出了诉讼。2000年6月26日微软在奥兰多举行的“职业开发人员技术大会(PDC 2000)”上,推出新的语言C#,它是在Visual J++基础上开发的软件,同时停止Visual J++的销售。所以,C#语言深受Java、C和C++的影响。

2000年6月微软公司发布一种新的编程语言C# 1.0。

2003年5月,微软推出了Visual Studio .NET 2003,同时也发布了C#的改进版本C# 1.1

微软在2004年的6月份发布了Visual Studio 2005的第一个Beta版,同时展示了C#2.0。

2005年9月份的PDC大会上微软推出C#3.0的技术预览版。

2010年4月推出的Visual Studio 2010及C#4.0,支持开发面向Windows7的应用程序。

2012年9月微软发布了Visual Studio 2012及C#5.0,可支持Windows8。

C#的巨大成功是安德斯 海尔斯伯格在编译器领域的领袖地位的又一次体现。可是说:安德斯 海尔斯伯格是C #之父。(Turbo Pascal、Delphi也是出自他手)

三. C#的优点:

C#是一种强大的、面向对象的程序开发语言,是专门用于.Net的编程语言,用C#编程的代码总是在.Net Framework中运行。

C#综合了VB简单的可视化操作和C++的高运行效率,以其强大的操作能力、优雅的语法风格、创新的语言特性和便捷的面向组件编程的支持成为.NET平台的首选语言。

C#在继承C和C++强大功能的同时去掉了一些它们的复杂特性(例如没有指针、宏以及不允许多重继承等)。

C#语言和Java很相似。学会C#后,学习Java很容易。

结论: 去熟悉VB2010,还不如直接学习、使用C#。



什么是.NET?

.NET平台是于2000年6月由Microsoft推出的全新的应用程序开发平台,用于构建和运行新一代的Microsoft Windows和Web应用程序。

.NET平台包括4种核心技术: .NET Framework, .NET企业服务器、构建模块服务、Visual Studio.NET。

.NET Framework是.NET平台核心中的核心。它为.NET平台下应用程序的运行提供基本框架。它有2个主要组件: CLR (公共语言运行时, Common Language Runtime)、.NET Framework类库。

本人感受:

- ◆ C#的功能强: 类库庞大、多线程、速度快;
- ◆ 界面设计简单、美观, 和VB类似;
- ◆ 稳定性比VB6.0强 (调试运动控制卡时没有自动退出现象);
- ◆ 智能化水平高 (写代码时格式自动对齐, 有智能提示; 调试程序时方便, 能给出有用的信息等);
- ◆ 学C#比学VC容易 (难点是要适应一些新概念, 如: 类)。

四. 安装C#

可以在网上下载免费的C#2010Express版 (也称为学习版) 软件包使用。

使用C#2010Express版可以正常运行运动控制卡。但缺少一些高级功能。

建议购买C#专业版软件使用。

五. 第一个C#程序

输入3个数字（字母也行），点击“开始”键，3个数按从小到大顺序重新排列，并输出结果。界面和运行结果如图3所示。

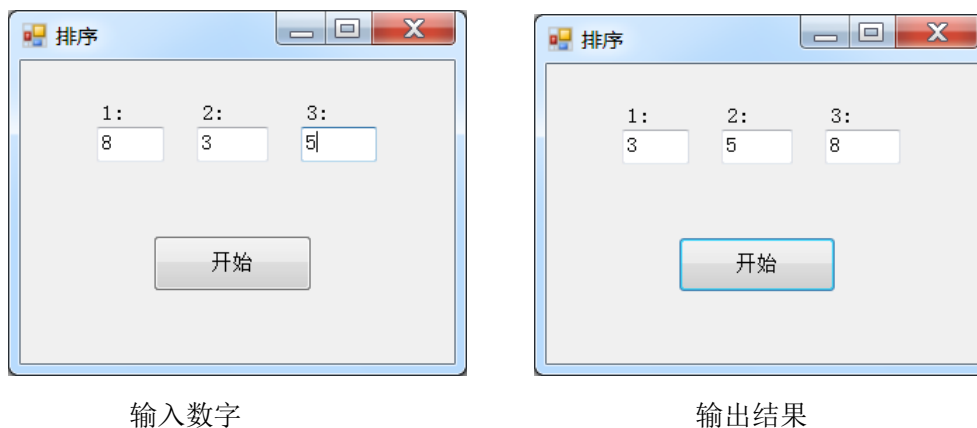


图3 第一个C#程序的界面和运行结果

C#的编辑界面和VB6.0相似，如图4、图5所示。和VB6.0不同的是：除了窗体及代码外，C#还有一个主程序Program.cs、一个系统自动生成的窗体设计程序Form1.Designer.cs。

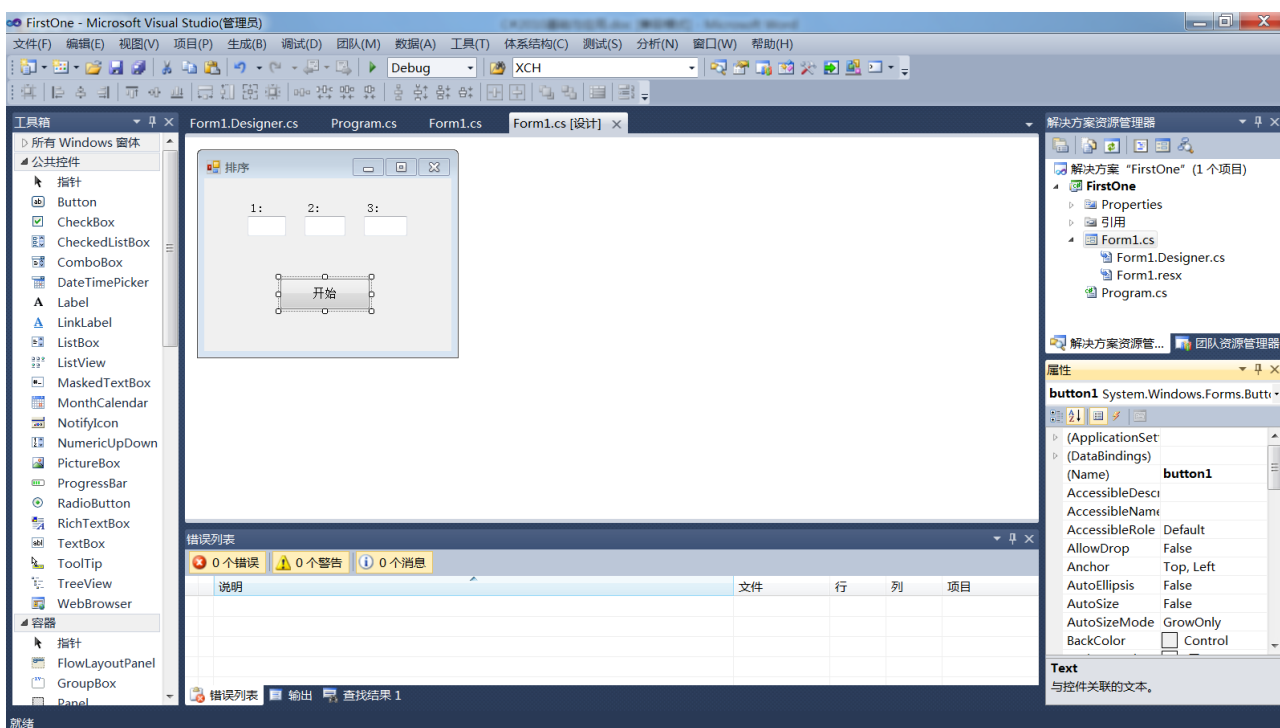


图4 C#的窗体编辑界面

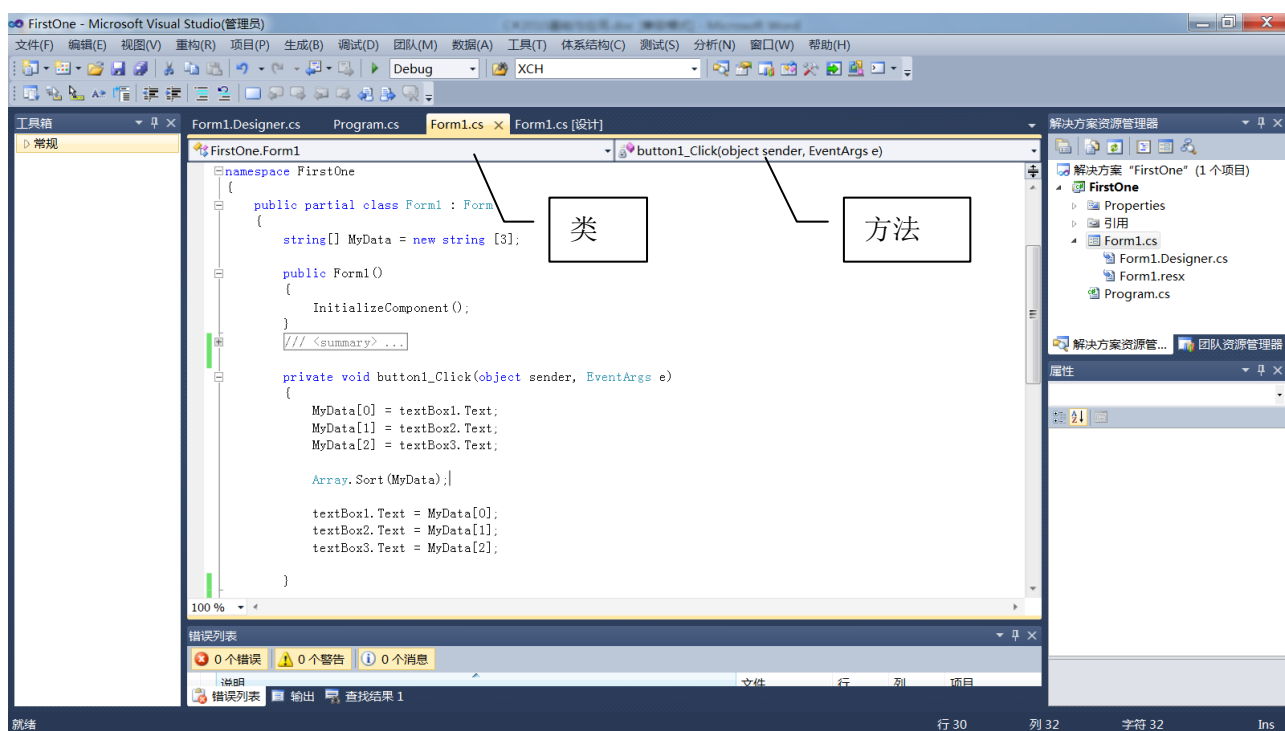


图5 C#的代码编辑界面

C#的程序结构如例程1代码所示。

例程1:

```

using System;                                // 引入System等命名空间
using System.Collections.Generic;             // 相当于VC的头文件
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace FirstOne                            // 创建名为FirstOne的命名空间
{                                              // 命名空间中有多个“类”文件

    // Program.cs 为主程序文件
    static class Program                      // 创建名为Program的静态类
    {
        /// <summary>
        /// 应用程序的主入口点。
        /// </summary>

```




```
[STAThread]           // 指示应用程序的COM线程模式是单线程单元（STA）。
static void Main()      // 创建名为Main（）的静态方法
{
    Application.EnableVisualStyles();           // 启用可视化样式
    Application.SetCompatibleTextRenderingDefault(false); // 设置呈现格式
    Application.Run(new Form1());              // 运行Form1窗体
}
}
```

// Form1.cs 为窗体事件的代码

```
public partial class Form1 : Form           // 创建名为Form1的公共类
{
    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e) // 开始按键被点击
    {
        string[] MyData = new string [3];    // 红色所示代码为编程者写的，其他均为自动生成
        MyData[0] = textBox1.Text;           // 输入数据
        MyData[1] = textBox2.Text;
        MyData[2] = textBox3.Text;
        Array.Sort(MyData);                  // 排序
        textBox1.Text = MyData[0];           // 输出结果
        textBox2.Text = MyData[1];
        textBox3.Text = MyData[2];
    }
}
```

// Form1.Designer.cs 为窗体设计代码，由Visual Studio自动生成

```
/// <summary>
/// 必需的设计器变量。
/// </summary>
private System.ComponentModel.IContainer components = null;
/// <summary>
/// 清理所有正在使用的资源。
/// </summary>
/// <param name="disposing">如果应释放托管资源，为 true；否则为 false。</param>
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {

```




```
        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Windows 窗体设计器生成的代码
/// <summary>
/// 设计器支持所需的方法 -
/// 不要使用代码编辑器修改此方法的内容。
/// </summary>
private void InitializeComponent()
{
    this.textBox1 = new System.Windows.Forms.TextBox();
    this.textBox2 = new System.Windows.Forms.TextBox();
    this.textBox3 = new System.Windows.Forms.TextBox();
    this.button1 = new System.Windows.Forms.Button();
    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // textBox1
    //
    this.textBox1.Location = new System.Drawing.Point(54, 47);
    this.textBox1.Name = "textBox1";
    this.textBox1.Size = new System.Drawing.Size(48, 25);
    this.textBox1.TabIndex = 0;
    //
    // textBox2
    //
    this.textBox2.Location = new System.Drawing.Point(125, 47);
    this.textBox2.Name = "textBox2";
    this.textBox2.Size = new System.Drawing.Size(51, 25);
    this.textBox2.TabIndex = 1;
    //
    // textBox3
    //
    this.textBox3.Location = new System.Drawing.Point(199, 47);
    this.textBox3.Name = "textBox3";
    this.textBox3.Size = new System.Drawing.Size(54, 25);
    this.textBox3.TabIndex = 2;
    //
    // button1
```




```
//
this.button1.Location = new System.Drawing.Point(94, 124);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(113, 40);
this.button1.TabIndex = 4;
this.button1.Text = "开始";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(55, 29);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(23, 15);
this.label1.TabIndex = 5;
this.label1.Text = "1.";
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(126, 29);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(23, 15);
this.label2.TabIndex = 6;
this.label2.Text = "2.";
//
// label3
//
this.label3.AutoSize = true;
this.label3.Location = new System.Drawing.Point(200, 29);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(23, 15);
this.label3.TabIndex = 7;
this.label3.Text = "3.";
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 15F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(308, 215);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
```



```
this.Controls.Add(this.label1);  
this.Controls.Add(this.button1);  
this.Controls.Add(this.textBox3);  
this.Controls.Add(this.textBox2);  
this.Controls.Add(this.textBox1);  
this.Name = "Form1";  
this.Text = "排序";  
this.ResumeLayout(false);  
this.PerformLayout();  
}  
#endregion  
  
private System.Windows.Forms.TextBox textBox1;  
private System.Windows.Forms.TextBox textBox2;  
private System.Windows.Forms.TextBox textBox3;  
private System.Windows.Forms.Button button1;  
private System.Windows.Forms.Label label1;  
private System.Windows.Forms.Label label2;  
private System.Windows.Forms.Label label3;  
}  
}
```

注意：在Express版C#软件中，新建项目时没有设置“保存项目”路径的选项。

可以通过点击“文件”→“全部保存”，或点击工具栏中的图标“”，出现如图6所示的对话框。可设置项目保存位置。

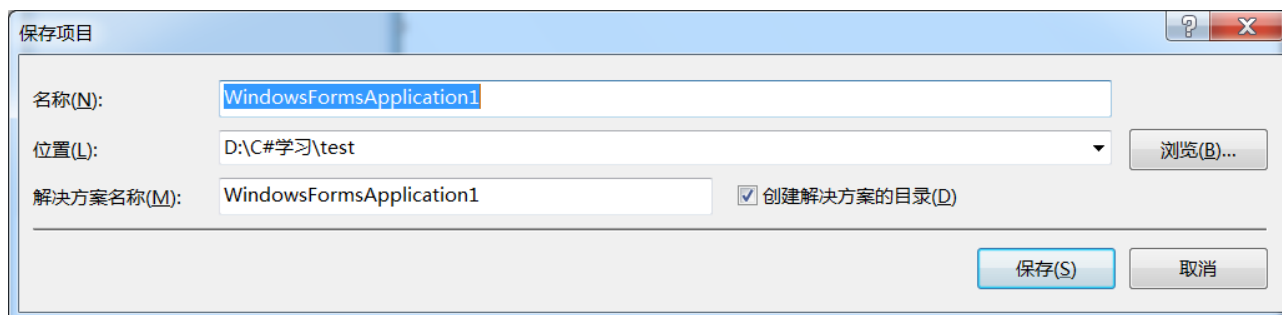


图6 Express版C#保存项目位置的设置



六. C#代码的语法、格式:

1. 使用using关键字把.Net Framework类库相对应的命名空间引入应用程序项目中。

2. 必须定义类

C#程序的源代码必须放在类中，一个程序至少包括一个自定义类，用关键字class声明。

3. 类的代码主要由方法组成

C#程序中必须包含Main方法。程序从Main方法的第1条语句开始运行，直到执行到其最后一条语句为止。

类中可以包含多个方法。方法名后一定有一对园括号。

4. C#语句以分号结束

一行代码可以写几条语句；一条语句也可以长达多行。

5. C#代码中区分大小写

6. C#的程序注释

// 单行注释

/*

多行注释

*/

/// <summary>

/// XML文档标记 (xml, eXtensible Markup Language, 即“可扩展标识语言”)

/// </summary>

7. 命名规则:

定义空间、类、方法、控件、变量等都需要命名。

名字可以由英文、数字或下划线组成，要使其意义、目的明确。不可用拼音。（下划线已不建议用）



微软在C#中主要使用的是帕斯卡命名法和骆驼命名法。

帕斯卡命名法 (Pascal)：每个单词的首字母大写，例如ProductType；

骆驼命名法 (Camel)：首个单词的首字母小写，其余单词的首字母大写，例如productType

还有匈牙利命名法，现在已不流行（微软已逐步减少了匈牙利命名法的使用）。其命名方法为：

变量类型前缀+变量名

例：intQuantity, lngPopulation, fltPayRate, dblWeight, strPhoneNumber

命名空间、类、窗体、方法、事件、属性的命名：

一般用大写字母开头，后面的单词的第一个字母也要大写。（帕斯卡命名法）

例：FirstOne、ScreenClass、MainForm、SetCopyNumber、ColorChanged、BackColor

控件的命名：第一个字母小写，其余单词首字母大写。（骆驼命名法）

例：textBox2、exitButton、streetNumberLabel

变量、方法的参数命名：

用有意义的单词命名即可。第一个字母小写，其余单词首字母大写。（骆驼命名法）

例：balance, index, nextMonthExpenditure, customerName

不要使用简称和无意义的名称，如：a, b1等。但i、j, x、y等约定俗成的变量可用。

常数的命名：

所有单词大写，多个单词之间用 “_” 隔开。 如：const double TAX_RATE = 0.23;

8. C#代码的其他规范：

- ◆ 接口的名称加前缀 **I**

```
interface ICompare
{
    int Compare();
}
```

- ◆ 自定义的属性以 **Attribute** 结尾

```
public class AuthorAttribute : Attribute
{
}
```

- ◆ 自定义的异常以 **Exception** 结尾

```
public class AppException : Exception
```

```
{  
}
```

- ◆ 代码的缩进。要用 **Tab**，而不要用 **space**。
- ◆ 所有的成员变量声明在类的顶端，用一个换行把它和方法分开。
- ◆ 生成和构建一个长的字符串时，一定要使用 **StringBuilder**，而不用 **string**。
- ◆ 始终使用“{ }”包含 **if** 下的语句，即使只有一条语句。
- ◆ 把相似的内容放在一起，比如数据成员、属性、方法、事件等，并适当的使用 **#region...#endregion**。

七. C#的常数、变量及其定义方法:

1. 常用的数据类型及取值范围，（如表1所示）

表1 C#的数据类型及取值范围

数据类型	.NET CTS 类型名	说明	字节数	取值范围
bool	System.Boolean	逻辑值		true, false
sbyte	System.Sbyte	8 位有符号整数	1	-128~127
byte	System.Byte	8 位无符号整数	1	0~255
short	System.Int16	16 位有符号整数	2	-32768~32767
ushort	System.UInt16	16 位无符号整数	2	0~65535
int	System.Int32	32 位有符号整数	4	-2147483648~2147483647
uint	System.UInt32	32 位无符号整数	4	0~4292967295
long	System.Int64	64 位有符号整数	8	-9223372036854775808~ 9223372036854775808
ulong	System.UInt64	64 位无符号整数	8	0~18446744073709551615
float	System.Single	32 位单精度浮点数	4	$\pm 3.4 \times 10^{-38} \sim \pm 3.4 \times 10^{+38}$ (约 7 位十进制数)
double	System.Double	64 位单精度浮点数	8	$\pm 1.7 \times 10^{-308} \sim \pm 1.7 \times 10^{+308}$ (约 16 位十进制数)
decimal	System.Decimal	128 位高精度十进制数	16	$\pm 1.0 \times 10^{-28} \sim \pm 7.9 \times 10^{+28}$ (约 29 位十进制数)
char	System.Char	16 位字符	2	所有 Unicode 编码字符
string	string 类包含在 System.Text 命名空间内			0~20 亿个字符

因为现在的计算机速度极快。建议：整数用long类型、浮点数用double类型。

bool类型是条件语句必须使用的类型。字符用char类型，字符串用string类型。

还有时间类型、日期类型、枚举类型、结构类型、对象类型等数据类型。不常用，不介绍。

和VB不同：int (Integer)、float (Single)、bool (Boolean)

2. 常数

1) 布尔常量：true和false

2) 整数常量：既可以是10进制也可以是16进制。

16进制数前加符号：0x。如：0x20, 0x1F

常整数后加符号L表示long型，常整数后加符号U表示无符号型。

例：32 // 默认为int型

32L // long型

32U // uint型

32UL // ulong型

3) 浮点常量

浮点数后加符号F表示float型， 如：3.14F

浮点数后加符号D表示double型， 如：3.14D

浮点数后加符号M表示decimal型。 如：3.14M

4) 字符串常量

A. 常规字符串：由双引号括起的一串字符，可以包括转义字符。

表2 转义字符表

转义字符	意义
\'	单引号
\"	双引号
\\	反斜线字符
\0	空字符(NULL)
\a	响铃符
\b	退格符，将当前位置移到前一行
\f	换页符，将当前位置移到下页开头
\n	换行符，将当前位置移到下一行开头
\r	回车符，将当前位置移到本行开头
\t	水平制表符，跳到下一个 TAB 位置
\v	垂直制表符

例：“Hello,World!\n”

// 字符Hello,World!后接换行符



“C:\\windows\\Microsoft” // 字符串为: C:\\windows\\Microsoft

B. 逐字字符串: 常规字符串前加符号@, 字符串中的字符均表示本意。

例: @“He said “Hello” to me.” // 字符串为: He said “Hello” to me.

5) 符号常量

定义格式: [常数修饰符] const 类型 常量名 = 初值

常用修饰符有: public、private

例: private const double PI = 3.1415926

3. 变量的定义 (和VB差别大)

声明变量的格式: [变量修饰符] 类型 变量名 [= 初值] [, ...];

常用修饰符有: public、private、static

(静态变量在函数调用结束后仍保留变量值, 只能在类中定义, 不能在方法中定义)

例: long distance;

double speed = 21.5, acc = 90.0 ;

string productName;

4. 数组

1) 一维数组的定义 (和VB差别很大)

方法1: 类型 [] 变量名; // 数组声明

变量名 = new 类型 [数组长度] { val1, val2, ... }; // 创建数组实例

方法2: 数组声明和创建数组实例同时进行

方法3: 数组声明和创建数组实例同时进行, 并完成初始化, 可省略new和类型

例: int [] number;

number = new int [10]; // 数组number有10个元素

float [] score = new float[100]; // 数组score有100个元素

long [] queue = {0,1,2,3,4,5}; // 数组queue有6个元素

2) 处理数组的常用方法:

a. Enumerable类的方法:

数组名.Count(); // 给出数组含元素的个数

数组名.Max(); // 给出数组中最大的元素

数组名.Min(); // 给出数组中最小的元素

数组名.Sum(); // 计算数组所有元素的和

数组名.Average(); // 计算数组所有元素的平均数

b. Array类的方法:

`Array.Clear`(数组名, 开始位, 长度); // 清零
`Array.Sort`(数组名); // 排序
`Array.Reverse`(数组名); // 反转数组中元素的顺序

八. 常用的运算符和表达式

1. 算术运算符:

加: + , 减: - , 乘: * , 除: / , 取模: % , 递增: ++ , 递减: --

2. 关系运算符:

大于: > , 大于等于: >= , 小于: < , 小于等于: <= , 等于: == , 不等于: !=

3. 逻辑运算符: 与: & , 或: | , 非: ! , 异或: ^

4. 位运算符:

与: & , 或: | , 非: ~ , 异或: ^ , 右移: >> , 左移: <<

和VB的不同: % (Mod), 无乘方符号 (^), ++、-- (无)

== (=), != (<>), & (And), | (Or), ! (Not)

5. 运算符及操作符从高到低的优先级顺序, 见表3

表3 运算符及操作符从高到低的优先级顺序表

类别	操作符	结合性
初级操作符	. () [] new typeof checked unchecked	从左到右
一元操作符	++ -- + - ! ~ (T)	从右到左
乘除操作符	* / %	从左到右
加减操作符	+ -	从左到右
移位操作符	<< >>	从左到右
关系操作符	< > <= >= is as	从左到右
等式操作符	== !=	从左到右
逻辑与操作符	&	从左到右
逻辑异或操作符	^	从左到右
逻辑或操作符		从左到右
条件与操作符	&&	从左到右
条件或操作符		从左到右
条件操作符	?:	从右到左
赋值操作符	= *= /= %= += -= <<= >>= &= ^= =	从右到左



6. 常用数学函数的调用：（System命名空间中的Math类）

Math.Abs(x); // x 的绝对值
Math.Pow(x,y); // x 的 y 次幂
Math.Sqrt(x); // x 的平方根
Log10(x); // x 的10为底的对数
Math.Sin(x); // x 的正弦
Math. Cos(x); // x 的余弦
Math.Tan(x); // x 的正切
Math.Asin(x); // x 的反正弦
Math. Acos(x); // x 的反余弦
Math.Atan(x); // x 的反正切
Math.Floor(x); // 返回小于或等于x的最大整数
Math.Round(x); // 四舍五入取整
Math.Sign(x); // x<0,返回-1; x=0,返回0; x>0,返回1
Math.Max(x,y); // 返回x,y中的最大值
Math.Min(x,y); // 返回x,y中的最小值

7. 数据类型的转换的方法：

a. 隐式转换：

C#可以自己进行一些数据类型的转换。字节占用小的类型可以自动转为字节占用多的类型。如：

int 可自动转为 long、float、double

float可自动转为double

b. 显式转换：

格式：（类型说明符）（待转换的数据）

例：b = (float) a;

z = (int) (x+y);

c. 使用Convert类提供的方法（System命名空间中的Convert类）



常用的有: Convert.ToString(x); // x 可以是int, long, float, double

Convert.ToDouble(x); // x 可以是string, int, long, float

Convert.ToInt32(x); // 即转换为 int

Convert.ToInt64(x); //即转换为 long

8. 常用字符串处理函数: (String 类)

字符串1.IndexOf(字符串2) // 在字符串1中查找第一个字符串2的位置

字符串.Substring(a, b) // 获取从字符串的第a个字符开始, 共b个字符

字符串.Substring(a) // 获取从字符串第a个字符开始的全部字符

字符串.Remove(a, b) // 从字符串的第a个字符开始, 删除b个字符

字符串1=字符串2+字符串3; // 字符串的加法

第二讲：C#2010 编程语言

九. C#的基本语句：

1. if else 语句（流程见图7）

语法格式：

```
if （条件表达式）
{
    语句块1;
}
else
{
    语句块2;
}
```

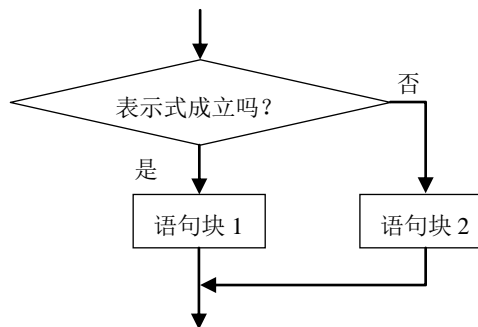


图 7 条件语句流程图

2. for 语句（流程见图8）

语法格式：

```
for（初始化表达式； 条件表达式； 迭代表达式）
{
    循环体;
}
```

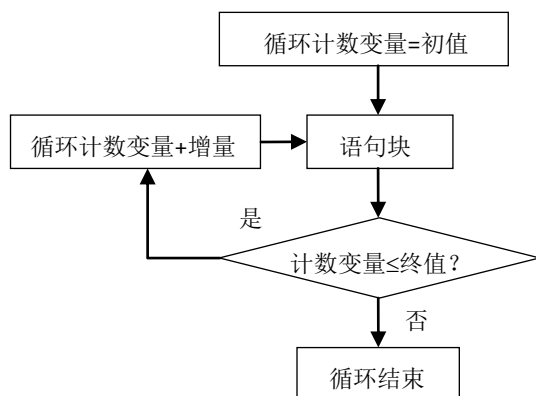


图 8 For 循环语句流程图

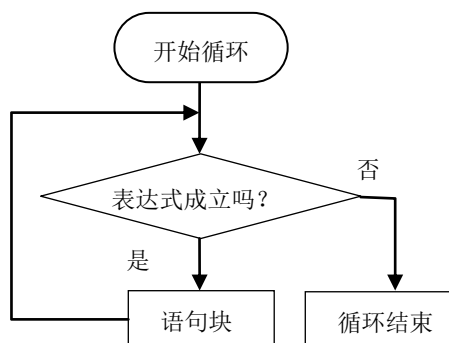


图 9 While 循环语句流程图

3. while 语句（流程见图9）

语法格式：

```
while（条件表达式）
{
    循环体;
}
```

4. 跳转语句

break语句：用于退出for、while、switch等循环语句

continue语句：在for、while等循环语句中，开始新的一次循环

return语句：**return**出现在方法中。执行**return**后，程序回到方法的调用处。

return可带参数。

例程2：百钱百鸡问题

鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

即：假定公鸡每只 5 元，母鸡每只 3 元，小鸡 3 只 1 元。现有 100 元，要买 100 只鸡，问公鸡、母鸡和小鸡各买几只？

分析：设买公鸡的数量为 **cock**，买母鸡的数量为 **hen**，买小鸡的数量为 **chick**。

显然， $1 \leq \text{cock} \leq 19$,

$1 \leq \text{hen} \leq 31$,

$\text{chick} = 100 - \text{cock} - \text{hen}$;

三种鸡的总价格为

$\text{sum} = \text{cock} \times 5 + \text{hen} \times 3 + \text{chick} / 3$

用搜索法可很快找到答案。即用两重循环语句控制买公鸡和母鸡的数量在 1~19 和 1~31 中依次变化；

再用条件语句判断哪种组合使买鸡的总价格正好等于 100 元。

计算流程见图 10。C#程序如下，结果如图 11 所示。

```
namespace Money100Cock100
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

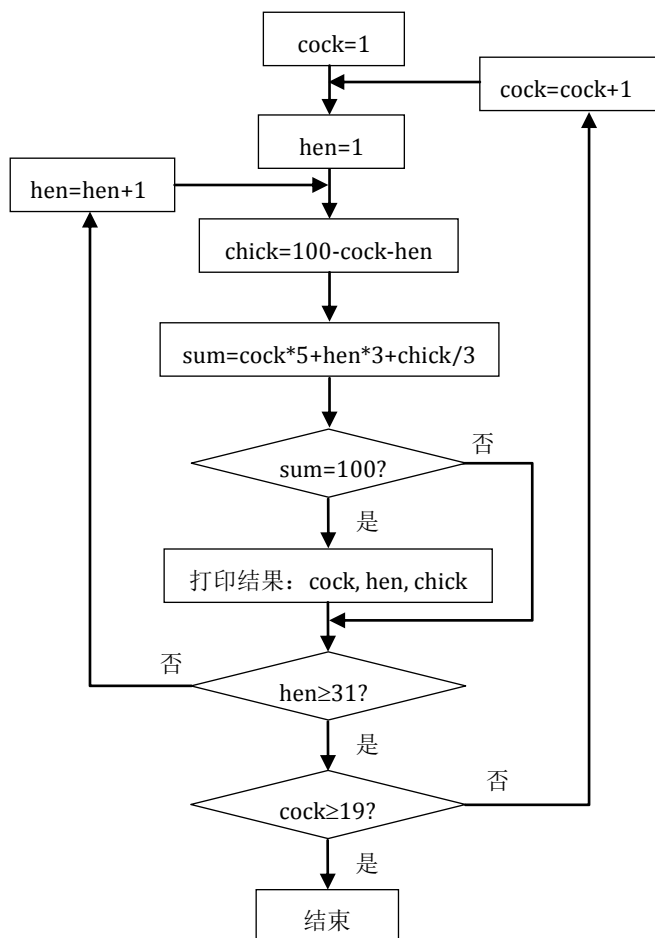


图 19 求解百钱百鸡问题的流程图



```
InitializeComponent();
}

private void start_Click(object sender, EventArgs e)
{
    double sum,cock,hen,chick;           // 定义局部变量
    string result;
    for ( cock = 1; cock < 20; cock ++ )   // 公鸡数量在1~19之间变化
    {
        for( hen = 1; hen< 32; hen ++ )    // 母鸡数量在1~31之间变化
        {
            chick = 100-cock-hen ;          // 小鸡数量为100-cock-hen
            sum = cock*5+hen*3+chick/3 ;    // 计算购鸡的总价格
            if (sum == 100)                 // 如果总价格sum = 100
            {
                result = "cock = " + Convert.ToString(cock);
                result = result + "   hen = " + Convert.ToString(hen);
                result = result + "   chick = " + Convert.ToString(chick)+"\n";
                richTextBox1.AppendText(result); // 输出符合“百钱百鸡”条件的结果
            }
        }
    }
}
}
```



图11 百钱百鸡问题的解



十. C#的类、方法：

1. 类：

类构成了面向对象编程的核心。

类实际上就是数据和处理数据代码的封装体，即：类封装了数据成员和函数成员。

其中数据成员有：常数、字段（即：变量）、域等；函数成员有：方法、属性、事件等。

类的实例则称为对象。

类的声明：

```
[类修饰符] class 类名 [ : 类基 ]  
{  
    类成员  
}
```

主要的类修饰符有：

public: 该类为公开的，访问不受限制

protected: 该类只能是本身或其派生的类访问

internal: 该类只能是在当前应用程序中访问

private: 该类只能是本身访问

类基：该类的直接基类和由该类实现的接口。

2. 类通过实例化才能得到具体的对象。

实例化类的格式：

类名 对象名 = new 类名 ([参数]);

3. 方法：

方法：就是按照一定格式组织的一段代码。即：VB中的过程、函数。

a. 方法的声明：

```
[方法修饰符] 返回类型 方法名 ([ 形参表 ])  
{  
    方法体  
}
```

主要的方法修饰符有：

- public:** 该方法为公开的，可以在任何地方访问
- protected:** 该方法可以在类中或其派生的类中被访问
- internal:** 该方法可以被同处一处的工程中的文件访问
- private:** 该方法只能在类中被访问
- static:** 静态方法从类一创建就开始存在

b. 方法中的参数传递

传值参数无需额外的修饰符。

传址参数需要修饰符ref（参数定义和参数调用时都要加上）

传值参数（值参数）在方法调用过程中如果改变了参数的值，那么传入方法的参数在方法调用完成以后并不因此而改变，而是保留原来传入时的值。

传址参数（引用参数）恰恰相反，如果方法调用过程改变了参数的值，那么传入方法的参数在调用完成以后也随之改变。

例程3：对3个数排序、计算3次方。

窗体如图12所示。

（类、方法、参数的传递）

namespace ExampleClass

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void start_Click(object sender, EventArgs e) // 开始排序

{

double a, b, c;

a = Convert.ToDouble(textBox1.Text); // 输入3个数

b = Convert.ToDouble(textBox2.Text);

c = Convert.ToDouble(textBox3.Text);

MyClass m = new MyClass(); // 实例化类MyClass为m

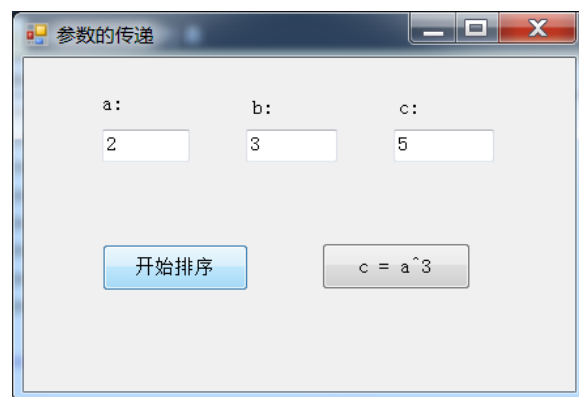


图 12 3 个数排序、计算 3 次方



```
m.Sort(ref a, ref b, ref c);           // 调用类m中的Sort方法，代入参数a,b,c
textBox1.Text = Convert.ToString(a);
textBox2.Text = Convert.ToString(b);
textBox3.Text = Convert.ToString(c);
}

private void power3_Click(object sender, EventArgs e)    // 计算3次方
{
    double a, c;
    a = Convert.ToDouble(textBox1.Text);
    c = power3(a);           // 调用方法，代入参数a
    textBox3.Text = Convert.ToString(c);
}

private double power3(double x)    // 计算x的3次方
{
    double z;
    z = x * x * x;           // x是传入的参数
    return z;               // z被回传
}

public class PowerThree
{
    public void Sort(ref double x, ref double y, ref double z)
    {
        // 该方法其他类中可以调用
        // a,b,c 三参数从小到大排序
        if (x > y) Swap(ref x, ref y);
        if (x > z) Swap(ref x, ref z);
        if (y > z) Swap(ref y, ref z);
    }

    private void Swap(ref double m, ref double n)
    {
        // 该方法其他类中不可调用
        // m,n两参数对换
        double temp;
        temp = m;
        m = n;
        n = temp;
    }
}
}
```

第三讲：C#2010 在运动控制卡上的应用

十一. C#中使用运动控制卡（以DMC2410C为例）

1. 将光盘中的DMC2410.cs文件拷贝至“项目名”文件夹内。
2. 在解决方案资源管理器下，鼠标右键点击“项目名称”→左键点击“添加”→点击“现存项”→选择DMC2410.cs文件→点击“添加”按钮

在解决方案资源管理器中可看到新增DMC2410.cs

该过程和VB6.0中添加运动控制卡的模块相似。

3. 将动态链接库DMC2410.dll拷贝至“项目名”文件夹→bin文件夹→Debug文件夹内。
4. 在代码文件开头处添加控制卡的命名空间：

```
using csDMC2410;
```

5. 调用运动控制卡的函数：函数名前加Dmc2410.，如：

```
Dmc2410.d2410_ex_s_pmove(Xch, Distance, 0); // 以S形速度曲线作相对运动  
(可理解为调用了Dmc2410类中的d2410_ex_s_pmove方法)
```

例程4：运动控制卡基本应用

可输入最大速度、运动距离；

点击按键实现正转、反转；

显示运动位置。

窗体如图13所示，代码如下。

和VB6.0一样，在C#程序中也是将运动控制卡的初始化函数放在FormLoad中，将控制卡的关闭函数放在FormClosed中。

（VB6.0为FormUnload）

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;
```

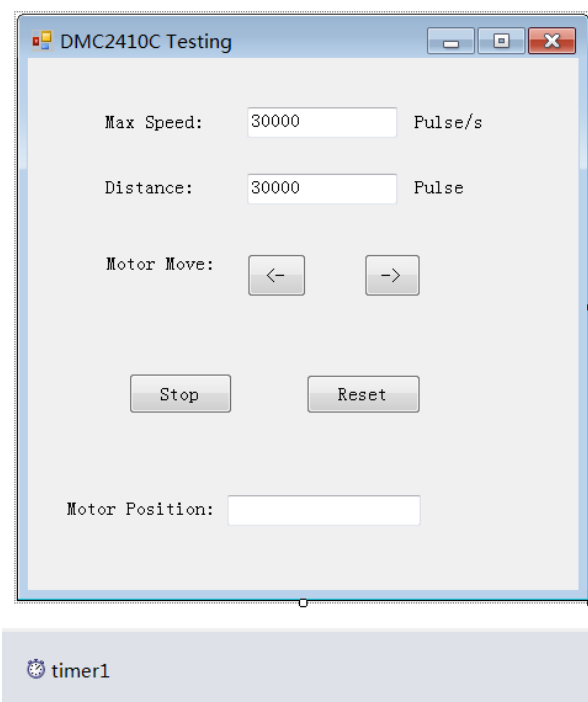


图 13 电机运动控制界面设计



```
using System.Text;
using System.Windows.Forms;
using csDmc2410;    // 引入DMC2410的类库

namespace Testing2410
{
    public partial class Form1 : Form
    {
        const Int32 Xch = 0;    // X轴为0号轴
        double InitSpeed = 1000, MaxSpeed, Tacc = 0.2, Tdec = 0.2;
        Int32 Distance;

        public Form1()
        { InitializeComponent(); }

        private void Form1_Load(object sender, EventArgs e) // 窗体载入事件
        {
            ushort nCard = Dmc2410.d2410_board_init(); // 初始化DMC2410
            if ((nCard <= 0) || (nCard >= 8)) // 正常情况卡数在1- 8之间
                MessageBox.Show("初始化DMC2410C卡失败！ ", "出错");
            timer1.Interval = 200;
            timer1.Stop(); // VB6.0用Enable=false
            Dmc2410.d2410_set_pulse_outmode(Xch, 0); // 设定脉冲输出模式
            Dmc2410.d2410_set_position(Xch, 0); // 设置X轴的脉冲位置为0
            textBoxPosition.Text = "0"; // 显示初始位置信息
        }

        private void Form1_FormClosed(object sender, FormClosedEventArgs e) // 窗体关闭事件
        {
            Dmc2410.d2410_board_close(); // 关闭DMC2410
        }

        private void RightMove_Click(object sender, EventArgs e) // 电机正转键被点击
        {
            if (Dmc2410.d2410_check_done(0) == 0)
                return; // 电机未停止运动
            MaxSpeed = Convert.ToDouble(textBoxMaxSpeed.Text);
            Distance = Convert.ToInt32(textBoxDistance.Text);
            Dmc2410.d2410_set_profile(Xch, InitSpeed, MaxSpeed, Tacc, Tdec);
            // 设置速度、加减速时间
            Dmc2410.d2410_t_pmove(Xch, Distance, 0); // 以梯形速度曲线作相对运动
            timer1.Start();
        }
    }
}
```



```
private void LeftMove_Click(object sender, EventArgs e) // 电机反转键被点击
{
    if (Dmc2410.d2410_check_done(0) == 0)
        return; // 电机未停止运动
    MaxSpeed = Convert.ToDouble(textBoxMaxSpeed.Text);
    Distance = Convert.ToInt32(textBoxDistance.Text);
    Dmc2410.d2410_set_profile(Xch, InitSpeed, MaxSpeed, Tacc, Tdec); // 设置速度、加减速时间
    Dmc2410.d2410_t_pmove(Xch, -1 * Distance, 0); // 以梯形速度曲线作相对运动
    timer1.Start();
}

private void timer1_Tick(object sender, EventArgs e) // 定时器触发事件
{
    long Position = Dmc2410.d2410_get_position(Xch); // 读取X轴当前指令位置
    textBoxPosition.Text = Convert.ToString(Position); // 显示位置信息
    if (Dmc2410.d2410_check_done(Xch) == 1)
        timer1.Stop(); // 电机停止，关闭定时器
}

private void ResetPosition_Click(object sender, EventArgs e) // Reset键被点击
{
    Dmc2410.d2410_set_position(Xch, 0); // 设置X轴的脉冲位置为0
    textBoxPosition.Text = "0"; // 显示初始位置信息
}

private void StopMove_Click(object sender, EventArgs e) // Stop键被点击
{
    Dmc2410.d2410_decel_stop(Xch, 0.1); // 减速停止
}
}
```

注意：等待电机运动停止时，要用DoEvents()

```
while (Dmc2410.d2410_check_done(0) == 0) // 等待运动停止
{
    Application.DoEvents(); // 处理当前消息队列中的消息。
}
```

在执行某事件的代码时，其他事件在队列中等待，应用程序不会响应；如果执行DoEvents()，则应用程序可以处理其他事件。

十二. C#的窗体及常用控件

前面已经用过了Button、Label、textBox、RichTextBox、Timer控件。

1. 菜单和多文档界面 MDI（多窗体）的应用方法：

1) 创建MDI父窗体

将窗体的属性IsMDIContainer改为true，则该窗体为父窗体；

最好将父窗体的属性WindowState改为Maximized。便于显示子窗体，因为子窗体不能超出父窗体。

2) 在父窗体上添加菜单

将MenuStrip控件拖入窗体，填写菜单各层子项名称。如图14所示。

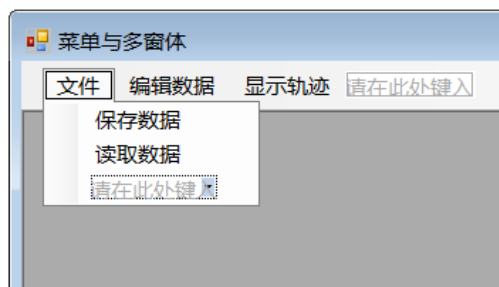


图 14 菜单的设置

3) 在项目中添加子窗体

右键点击解决方案资源管理器中的“方案名”，→“添加”→“Windows窗体”（默认名称为Form2）→点击“添加”键，完成一个窗体的添加。

设计子窗体的控件和代码。

4) 打开子窗体

双击菜单中某项名称，创建Click事件，并写代码如下：

```
Form2 DataList = new Form2(); // 实例化Form2，名称为DataList
DataList.MdiParent = this;    // 设置子窗体的父窗体
DataList.Show();              // 显示子窗体
```

5) 窗体间数据的传递

方法不少，但最简单的方法是：通过一个公共类的公共静态成员来实现。



例：

```
public class class1          //自定义一个公共类class1
{
    public static string str;  //定义公共的静态字符串，名称为str
}
```

在Form1中赋值：class1.str = "传入的内容";

在Form2中使用：string s = class1.str;

用这样的方式可以很方便地在窗体之间传递变量、数组、字符串等数据。

例程5：多窗体

Form1是菜单，Form2是文本编辑，Form3是轨迹显示及加工显示。

程序运行结果如图15所示。

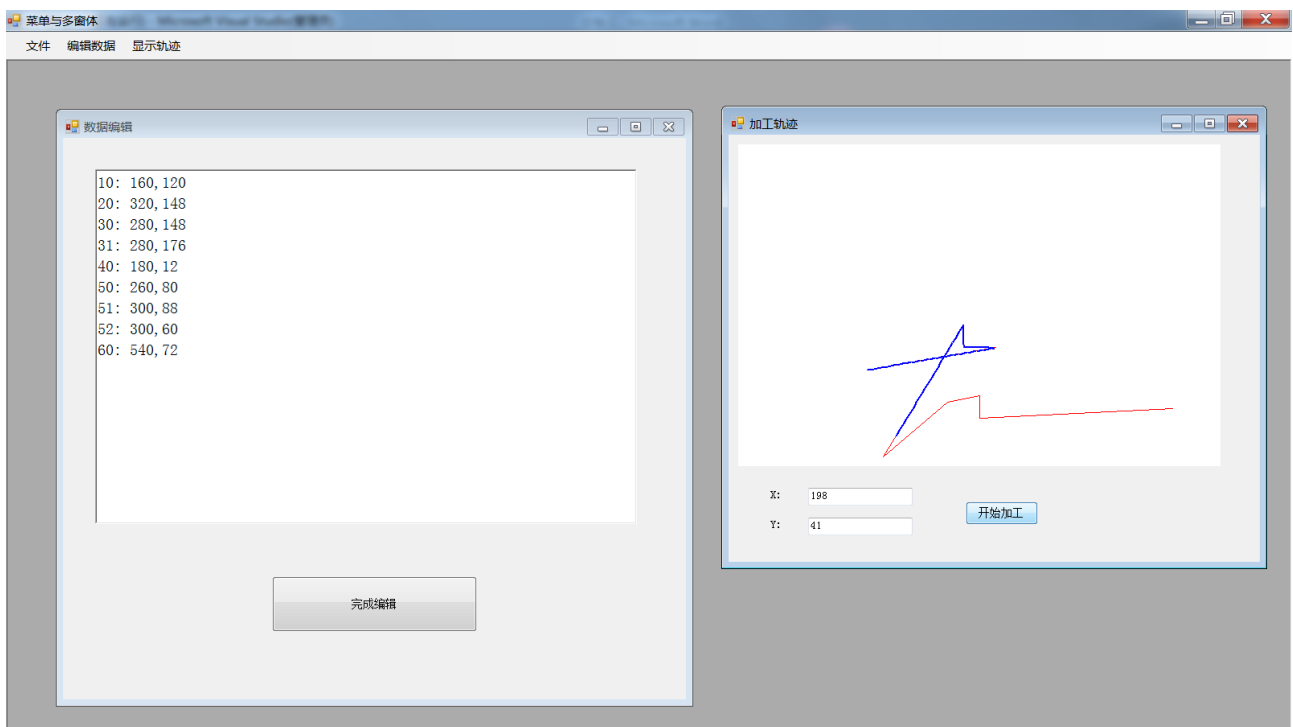


图 15 多窗体界面及菜单

Form1的代码见后面的例程5（续）。

Form2的代码如下：

```
public partial class Form2 : Form
{
    public Form2()
    { InitializeComponent(); }
```



```
private void Done_Click(object sender, EventArgs e)
{   shareData.shareString = richTextBox1.Text;   }

private void Form2_Load(object sender, EventArgs e)
{   richTextBox1.Text = shareData.shareString;   }
}
```

Form3的主要代码如下：

```
private void StartMove_Click(object sender, EventArgs e)
{
    Dmc2410.d2410_set_profile(0, 1000, 3000, 0.1, 0.1); // 设置速度、加减速时间
    Dmc2410.d2410_set_profile(1, 1000, 3000, 0.1, 0.1); // 设置速度、加减速时间
    Dmc2410.d2410_t_pmove(0, x[0], 1);                // 以梯形速度曲线作绝对运动
    Dmc2410.d2410_t_pmove(1, y[0], 1);                // 运动至轨迹起点
    while (Dmc2410.d2410_check_done(0) == 0 | Dmc2410.d2410_check_done(1) == 0)
    { Application.DoEvents(); }                       // 等待运动停止
    x0 = x[0];                                         // x[ ]、y[ ]为轨迹的各线段端点坐标
    y0 = y[0];
    timer1.Interval = 60;
    timer1.Start();
    Dmc2410.d2410_set_vector_profile(50, 60, 0.1, 0.1); // 设置矢量速度、加减速时间
    for (int j = 1; j < x.Count(); j++)
    {
        if (x[j] == 0 & y[j] == 0) break;
        Dmc2410.d2410_t_line2(0, x[j], 1, y[j], 1);    // 以绝对坐标做直线插补运动
        while (Dmc2410.d2410_check_done(0) == 0 | Dmc2410.d2410_check_done(1) == 0)
        { Application.DoEvents(); }                   // 等待运动停止
    }
    timer1.Stop();
}

private void timer1_Tick(object sender, EventArgs e)
{
    Graphics g = pictureBox1.CreateGraphics();        // 创建画板
    g.TranslateTransform(0, 400);
    Pen myPen = new Pen(Color.Blue, 2);              // 画笔宽度为2
    int x = Dmc2410.d2410_get_position(0);            // 读取当前指令位置
    int y = Dmc2410.d2410_get_position(1);
    textBox1.Text = Convert.ToString(x);
    textBox2.Text = Convert.ToString(y);
    g.DrawLine(myPen, x0, -1 * y0, x, -1 * y);        // 画出已完成的轨迹
}
```

```
x0 = x;  
y0 = y;  
}
```

2. 常用对话框

1) 消息框：（见图16）

```
MessageBox.Show("数据读取成功!");
```

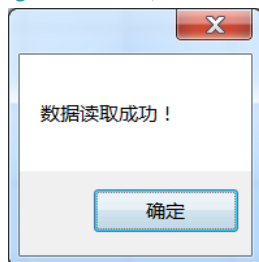


图 16 消息框界面

2) 保存文件对话框及数据存储：

存储数据有多种格式：顺序文件、随机文件、二进制文件；还有数据库。

为方便：我们使用顺序文件，存储字符串。相当于TXT文件，可用任何文本编辑软件打开、编辑。

保存文件对话框如图17所示，对应的代码见例程5（续）。

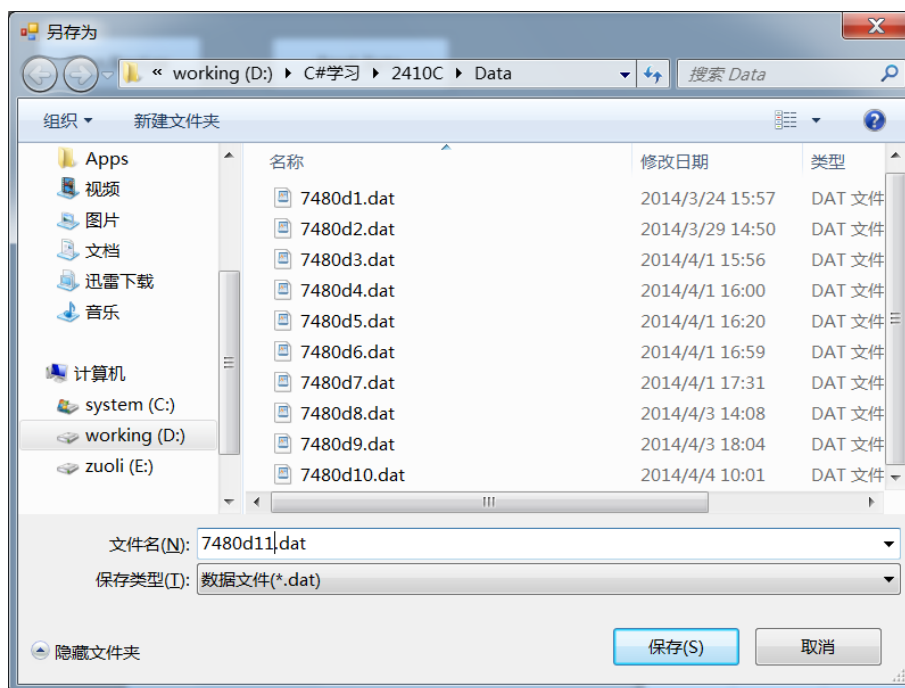


图 17 存文件对话框

3) 打开文件对话框及数据读取:

打开文件对话框如图18所示, 对应的代码见例程5 (续)。

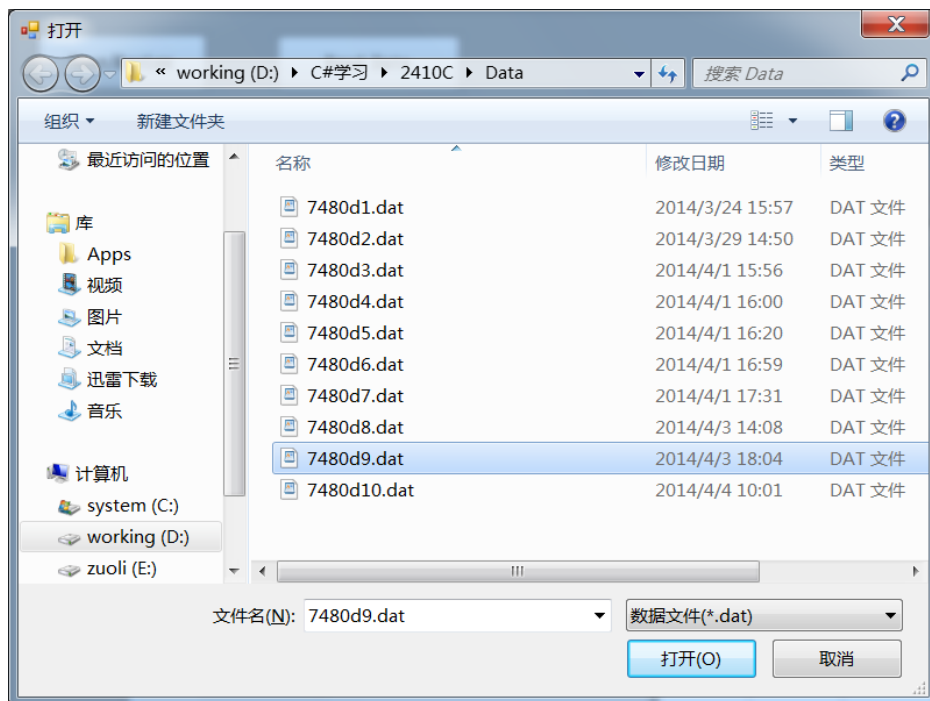


图 18 打开文件对话框

例程5 (续): Form1的代码

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;           // 文件读写的类
```

```
namespace MultiForm
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```



```
private void SaveToolStripMenuItem_Click(object sender, EventArgs e) // 存文件
{
    SaveFileDialog sfd = new SaveFileDialog(); //创建保存文件对话框的实例
    sfd.Filter = "数据文件(*.dat)|*.dat|All files(*.*)|*.*"; //设置为数据文件或所有文件
    sfd.DefaultExt = ".dat"; //设置默认扩展名为dat
    if (sfd.ShowDialog() == DialogResult.OK) //打开对话框
    {
        try
        {
            StreamWriter m_SW = new StreamWriter(sfd.FileName); // 实例化StreamWriter类
            m_SW.Write(shareData.shareString); // 写入数据
            m_SW.Close(); // 关闭文件
        }
        catch (IOException ex)
        {
            MessageBox.Show("写入出错! \n" + ex.Message);
        }
    }
}

private void ReadToolStripMenuItem_Click(object sender, EventArgs e) // 读文件
{
    OpenFileDialog ofd = new OpenFileDialog(); //创建打开文件对话框的实例
    ofd.Filter = "数据文件(*.dat)|*.dat|All files(*.*)|*.*"; //设置为数据文件或所有文件
    ofd.DefaultExt = ".dat"; //设置默认扩展名为dat
    if (ofd.ShowDialog() == DialogResult.OK) //打开对话框
    {
        try
        {
            StreamReader m_SW = new StreamReader(ofd.FileName); // 实例化StreamReader类
            shareData.shareString = m_SW.ReadToEnd(); // 读数据
            m_SW.Close(); // 关闭文件
        }
        catch (IOException ex)
        {
            MessageBox.Show("读取出错! \n" + ex.Message);
            return;
        }
    }
}

private void EditToolStripMenuItem_Click_1(object sender, EventArgs e)
{
    label1.Visible = false;
    label2.Visible = false;
}
```



```
Form2 DataList = new Form2(); // 实例化Form2, 名称为DataList
DataList.MdiParent = this;    // 设置子窗体的父窗体
DataList.Show();              // 显示子窗体
}

private void DrawToolStripMenuItem_Click(object sender, EventArgs e)
{
    label1.Visible = false;
    label2.Visible = false;
    Form3 DrawCurve = new Form3(); // 实例化Form2, 名称为DrawCurve
    DrawCurve.MdiParent = this;    // 设置子窗体的父窗体
    DrawCurve.Show();              // 显示子窗体
}

}

public class shareData          // 自定义的一个公共类
{
    public static string shareString; // 各窗口的公用字符串
}

}
```

第四讲：C#2010 绘图、多线程编程方法

十三. C#的绘图方法

.NET Framework 提供了图形设备接口GDI (Graphic Device Interface)。

C#提供的Graphics类，提供了将对象绘制到显示设备的方法。

绘图通常在pictureBox中进行。主要步骤如下：

1. 在窗体上新建一个pictureBox控件，背景色设为白色；
2. 创建Graphics对象：

```
Graphics g = pictureBox1.CreateGraphics(); // 相当于C++中获取句柄
```

3. 定义坐标：

默认坐标原点是绘图区的左上角。Y轴坐标方向朝下。如图19所示。

坐标的最小单位即像素点。

坐标可以平移，函数如下：

```
g.TranslateTransform(200,200); // X轴右移200；y轴下移200，方向仍朝下
```

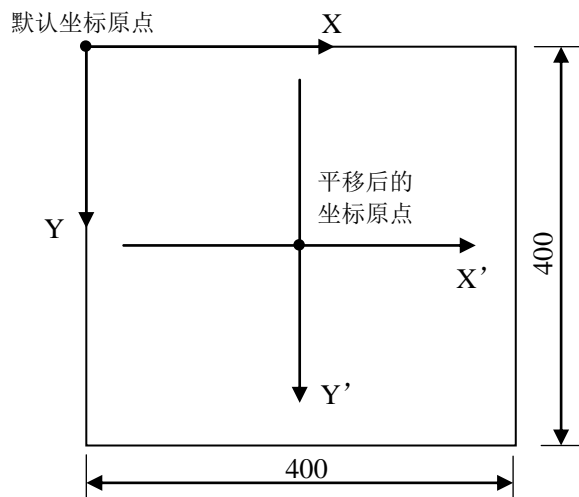


图 19 默认坐标原点及坐标平移

画图时，y坐标的数据乘-1，则得到正常图形。



4. 定义画笔

方法1:

```
Pen myPen = new Pen( Color.Red ); // 实例化Pen为myPen, 红色, 宽度为1
```

方法2:

```
Pen myPen = new Pen( Color.Red, 2.0f ); // 宽度为2
```

5. 画直线:

方法 1: 绘制一条由坐标对指定的两端点的直线。

```
DrawLine(Pen, Int32, Int32, Int32, Int32);
```

方法 2: 绘制一条由两个 PointF 结构定义的直线。

```
PointF point1= new PointF(100.0F, 100.0F);
```

```
PointF point2= new PointF(500.0F, 300.0F);
```

```
DrawLine( Pen, point1, Point2 );
```

结构: 由一系列相关参数组织成的一个单一实体。使用方法和“类”相似。也可以自己定义。

常用的结构成员有: 类型、常数、变量等。

6. 画椭圆:

```
DrawEllipse(Pen, x, y, h, w);
```

// 椭圆的大小由一矩形定义, 该矩形的左上角坐标(x, y)、高度h、宽度w。

7. 画曲线: 使用DrawLines或DrawCurve函数

例程6: 画曲线 (结果见图20)

```
int[] x = { 30, 60, 90, 120, 150, 180, 210, 240, 270, 300 };  
int[] y = { 60, 120, 30, 240, 300, 240, 30, 120, 60, 30 };  
Graphics g = pictureBox1.CreateGraphics(); //创建Graphics对象  
g.TranslateTransform(0, 400);  
Pen RedPen = new Pen(Color.Red); //创建红色的画笔  
Pen GreenPen = new Pen(Color.Green); //创建绿色的画笔  
Point[] points = new Point[10]; //创建坐标点的结构  
for (int i = 0; i < points.Length; i++)  
{
```



```
points[i] = new Point(x[i], -1 * y[i]);    //设置各个点的坐标值
}
g.DrawCurve(RedPen, points);              // 绘制经过一组指定的坐标点的样条曲线。
g.DrawLines(GreenPen, points);            // 绘制折线
for (int i = 0; i < 10; i++)
{
    g.DrawEllipse(new Pen(Color.Blue,3),x[i],-1*y[i],1,1);    //画各个点
}
```

用DrawLines、DrawCurve画曲线比用DrawLine画曲线快！

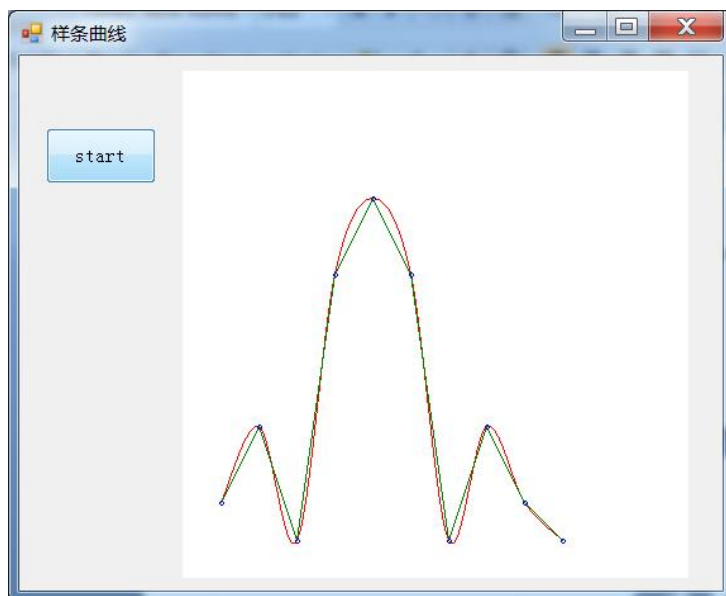


图 20 两种画曲线方法的对比

8. 画矩形:

```
Rectangle rect = new Rectangle(x1, y1, width, height); // 定义矩形: 左上角坐标、宽、高
g.DrawRectangle(Pen, rect);                          // 画矩形
```

十四. 鼠标事件:

例程7: 用鼠标在pictureBox中画框

在窗体设计界面选中pictureBox1，点击属性框上的事件图标（如图19所示），双击MouseDown，自动生成“鼠标键按下事件”的方法名：

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
```

然后编写相关代码即可。



建立“鼠标键抬起、鼠标移动事件的方法”过程也一样。

用鼠标在pictureBox中画框的结果见图21，代码如下所示。

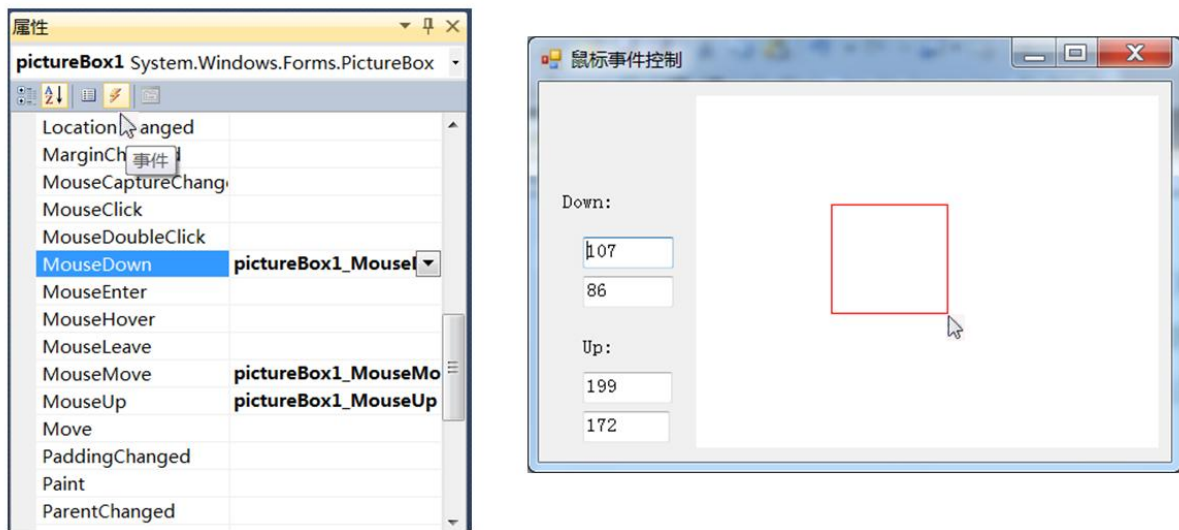


图 21 鼠标事件及例程界面

```
int x1, y1, x2, y2, mouseDown = 0;
```

```
int x0 = 0, y0 = 0, height = 0, width = 0;
```

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
```

```
{
    if (e.Button == MouseButtons.Left)           // 鼠标左键按下
    {
        x1 = e.X;                                // 获取鼠标左键按下时的坐标
        y1 = e.Y;
        textBox1.Text = Convert.ToString(x1);    // 显示坐标
        textBox2.Text = Convert.ToString(y1);
        mouseDown = 1;                           // 鼠标左键按下标识为1
    }
}
```

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
```

```
{
    if (e.Button == MouseButtons.Left)           // 鼠标左键抬起
    {
        x2 = e.X;                                // 获取鼠标左键抬起时的坐标
        y2 = e.Y;
        textBox3.Text = Convert.ToString(x2);    // 显示坐标
        textBox4.Text = Convert.ToString(y2);
        mouseDown = 0;                           // 鼠标左键按下标识为0
    }
}
```



```
}  
  
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)  
{  
    if (mouseDown == 1) // 鼠标左键按下后的移动  
    {  
        Graphics g = pictureBox1.CreateGraphics();  
        Pen WhitePen = new Pen(Color.White);  
        Pen RedPen = new Pen(Color.Red);  
        Rectangle rect0 = new Rectangle(x0, y0, width, height);  
        g.DrawRectangle(WhitePen, rect0); // 擦除上次画的框  
        width = e.X - x1; // 计算鼠标左键按下后移动的距离  
        height = e.Y - y1;  
        Rectangle rect = new Rectangle(x1, y1, width, height); //创建矩形数据结构并赋值  
        g.DrawRectangle(RedPen, rect); // 画框  
        x0 = x1; y0 = y1;  
    }  
}
```

十五. C#的多线程:

“实时多任务”是自动化设备应用软件的基本要求。只有这样，设备的生产效率才高。使用SMC6480运动控制器后，大家对多任务程序在自动化设备上应用的优势都深有体会。多任务就是多线程。

多线程程序允许在单个程序中创建多个任务，而且能并行完成这些任务。

在单CPU系统的一个单位时间（time slice）内，CPU只能运行单个线程，运行顺序取决于线程的优先级。如果在单位时间内线程未能完成执行，系统就会把线程的状态信息保存到线程的本地存储器（TLS）中，以便下次执行时恢复执行。而多线程只是系统带来的一个假像，它在多个单位时间内进行多个线程的切换。因为切换频密而且单位时间非常短暂，所以多线程可被视作同时运行。

在C#中，与多线程相关的类都放在System.Threading命名空间中。编程方法如下：

1. 线程的创建、启动和终止

默认情况下，C#程序只有一个线程（方法Main（）），即主线程，参见例程1。

创建多线程，即创建辅助线程（子线程）。过程如下所示：

```
private Thread Job1; // 定义线程名 Job1  
Job1 = new Thread(MyJob); // 创建线程Job1，调用名为“MyJob”的方法  
Job1.Start(); // 启动线程Job1
```

```
.....  
Job1.Abort();                // 终止线程Job1  
  
.....  
private void MyJob()        // 定义方法 “MyJob”  
{  
    .....  
}
```

2. 线程的挂起

代码为：`Thread.Sleep(10);` // 延时10ms

调用`Thread.Sleep (m)`后，当前线程会阻塞m毫秒，并将其时间片段的剩余部分提供给其他线程。相当于该线程延时m毫秒。

`Sleep`计时误差大。（Express版、专业版C#中的`Sleep(1)`的最长时间达15ms。建议最小延时用`Sleep(10)`，这样误差相对较小。）

若要精确计时，可采用`Stopwatch`，详见后续的第五讲：“C#2010高速采样的方法”。

3. 线程的优先级（见表4）

表4 线程的优先级

优先级	说明
Lowest	可以将 Thread 安排在具有任何其他优先级的线程之后。
BelowNormal	可以将 Thread 安排在具有 Normal 优先级的线程之后，在具有 Lowest 优先级的线程之前。
Normal	可以将 Thread 安排在具有 AboveNormal 优先级的线程之后，在具有 BelowNormal 优先级的线程之前。
AboveNormal	可以将 Thread 安排在具有 Highest 优先级的线程之后，在具有 Normal 优先级的线程之前。
Highest	可以将 Thread 安排在具有任何其他优先级的线程之前。

优先级设置的方法如下：

`Job1.Priority = ThreadPriority.Highest;` // 线程Job1优先级最高

在创建线程时，如果不指定其优先级，系统将默认为`Normal`。

4. 子线程访问主线程控件的方法

一般情况下子线程不能直接访问主线程的控件。

如果要进行跨线程操作，需要使用“`MethodInvoker`委托”。



例：`this.Invoke((MethodInvoker)delegate {this.textBox1.Text = count1.ToString();});`

例程8：多线程及跨线程使用控件

2个子线程做累加。因其优先级不同，
做累加的速度也就不同。（参见图22）
主线程做加法，不受2个子线程的影响。

代码如下：

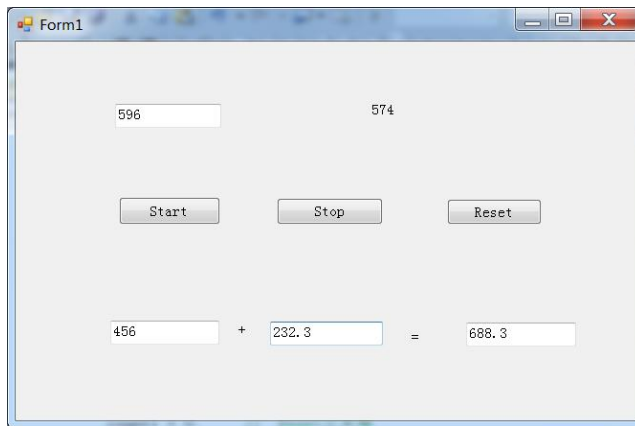


图 22 主线程界面

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
```

// 引入多线程命名空间

```
namespace ThreadingTest2
```

```
{
```

```
    public partial class Form1 : Form
```

```
    {
```

```
        private Thread Job1;           // 定义线程名 Job1
```

```
        private Thread Job2;           // 定义线程名 Job2
```

```
        private long count1, count2 = 0;
```

```
        public Form1()
```

```
        { InitializeComponent(); }
```

```
        private void button1_Click(object sender, EventArgs e) // Start键按下
```

```
        {
```

```
            Job1 = new Thread(MyJob1);           // 创建线程
```

```
            Job2 = new Thread(MyJob2);
```

```
            Job1.Priority = ThreadPriority.Highest; // 线程Job1优先级最高
```

```
            Job2.Priority = ThreadPriority.Lowest; // 线程Job2优先级最低
```

```
            Job1.Start();                         // 启动线程
```

```
            Job2.Start();
```

```
            button1.Visible = false;              // 防止多次启动线程
```

```
        }
```

```
        private void MyJob1()                     // 线程1的方法
```



```
{
    for (int i = 0; i < 100000; i++)
    {
        count1 = count1 + 1;
        this.Invoke((MethodInvoker)delegate { this.textBox1.Text = count1.ToString(); });
        // 跨线程操作，在Form1线程上执行指定的委托
        Thread.Sleep(1);           // 延时1ms
    }
}

private void MyJob2()           // 线程2的方法
{
    for (int j = 0; j < 100000; j++)
    {
        count2 = count2 + 1;
        this.Invoke((MethodInvoker)delegate { this.label3.Text = count2.ToString(); });
        Thread.Sleep(1);
    }
}

private void button2_Click(object sender, EventArgs e) // Stop键按下
{
    Job1.Abort();           // 终止线程
    Job2.Abort();
    button1.Visible = true;
}

private void button3_Click(object sender, EventArgs e) // Reset键按下
{
    count1 = 0;
    count2 = 0;
    textBox1.Text = "Job1 Result";           // Reset文本框
    label3.Text = "Job2 Result";           // Reset标签
}

private void label4_Click(object sender, EventArgs e) // “等号”被点击
// 主线程的工作
{
    double a = Convert.ToDouble(textBoxA.Text);
    double b = Convert.ToDouble(textBoxB.Text);
    textBoxC.Text = Convert.ToString(a + b);
}
}
```

例程9：多线程在运动控制卡上的应用

点击Start后，电机正转、反转3次后停止。在电机运动中，检测输入口Input2；若有输入信号，则Output1立即输出。界面如图23所示。

先比较一下使用单线程的程序运行情况：Stop按键很难响应；定时器55ms扫描一次电机位置及IO状态。

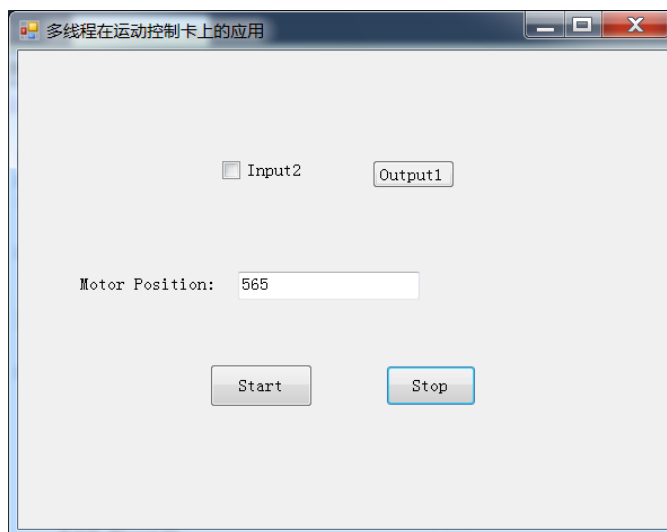


图23 多线程程序界面

单线程控制电机正反转的代码：

```
private void button1_Click(object sender, EventArgs e)
{
    timer1.Start();
    Dmc2410.d2410_set_position(0, 0); // 设置X轴的脉冲位置为0
    textBox1.Text = "0";             // 显示初始位置信息
    Dmc2410.d2410_set_st_profile(0, 100, 200, 0.3, 0.3, 0.1, 0.1);
    // 设置速度、加减速时间、S段时间
    for (int i = 0; i < 3; i++)
    {
        Dmc2410.d2410_ex_s_pmove(0, 1000, 0); // 电机正转
        while (Dmc2410.d2410_check_done(0) == 0) // 等待运动停止
            Application.DoEvents(); // 处理当前消息队列中的消息。
        Dmc2410.d2410_ex_s_pmove(0, -1000, 0); // 电机反转
        while (Dmc2410.d2410_check_done(0) == 0) // 等待运动停止
            Application.DoEvents(); // 处理当前消息队列中的消息。
    } // 查询电机停止，占用了大量CPU时间！
}
```

单线程检测控制IO的代码：

```
private void timer1_Tick(object sender, EventArgs e) // IO口每55ms检测一次
{
    int Position = Dmc2410.d2410_get_position(0); // 读取X轴当前指令位置
    textBox1.Text = Convert.ToString(Position); // 显示位置信息
    if (Dmc2410.d2410_read_inbit(0, 2) == 0) // 读取输入口状态
    {
        Dmc2410.d2410_write_outbit(0, 1, 1); // 写输出口
        checkBox1.Checked = true;
    }
}
```




```
        checkBox2.Checked = true;
    }
    else
    {
        Dmc2410.d2410_write_outbit(0, 1, 0);
        checkBox1.Checked = false;
        checkBox2.Checked = false;
    }
}
```

采样多线程控制方式实现该功能：

主线程负责人机界面、子线程1负责电机控制、子线程2负责IO控制。完整的代码如下：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;           // 引入多线程命名空间
using csDmc2410;                 // 引入Dmc2410命名空间

namespace Testing2410Input
{
    public partial class Form1 : Form
    {
        private Thread Job1;       // 定义线程名 Job1
        private Thread Job2;       // 定义线程名 Job2
        int flagJob = 0;           // 线程工作的标志

        public Form1()
        { InitializeComponent(); }

        private void button1_Click(object sender, EventArgs e)
        {
            Job1 = new Thread(MyJob1);           // 创建线程
            Job2 = new Thread(MyJob2);
            Job1.Priority = ThreadPriority.BelowNormal; // 线程Job1优先级较低
            Job2.Priority = ThreadPriority.AboveNormal; // 线程Job2优先级较高
            Job1.Start();                         // 启动线程
            Job2.Start();
        }
    }
}
```



```
flagJob = 1;
button1.Visible = false;           // 防止多次启动线程
timer1.Start();
Dmc2410.d2410_set_position(0, 0);  // 设置X轴的脉冲位置为0
textBox1.Text = "0";               // 显示初始位置信息
}

private void MyJob1()               // 子线程1负责电机控制
{
    Dmc2410.d2410_set_st_profile(0, 100, 200, 0.3, 0.3, 0.1, 0.1);
                                     // 设置速度、加减速时间、S段时间
    for (int i = 0; i < 3; i++)
    {
        Dmc2410.d2410_ex_s_pmove(0, 1000, 0);    // 电机正转
        while (Dmc2410.d2410_check_done(0) == 0) // 等待运动停止
            Thread.Sleep(10);                     // 延时10ms
        Dmc2410.d2410_ex_s_pmove(0, -1000, 0);    // 电机反转
        while (Dmc2410.d2410_check_done(0) == 0) // 等待运动停止
            Thread.Sleep(10);                     // 延时10ms
    }
    this.Invoke((MethodInvoker)delegate { this.button1.Visible = true; }); // 跨线程操作窗体
    flagJob = 0;
    Job1.Abort();                               // 终止线程
    Job2.Abort();
}

private void MyJob2()               // 子线程2负责IO控制
{
    while (flagJob == 1)
    {
        if (Dmc2410.d2410_read_inbit(0, 2) == 0) // 读取输入口状态
        {
            this.Invoke((MethodInvoker)delegate { this.checkBox1.Checked = true; });
            Dmc2410.d2410_write_outbit(0, 1, 1); // 写输出口
            this.Invoke((MethodInvoker)delegate { this.checkBox3.Checked = true; });
        }
        else
        {
            this.Invoke((MethodInvoker)delegate { this.checkBox1.Checked = false; });
            Dmc2410.d2410_write_outbit(0, 1, 0);
            this.Invoke((MethodInvoker)delegate { this.checkBox3.Checked = false; });
        }
        Thread.Sleep(10); // 延时10ms,提高输入信号的响应速度5倍
    }
}
```



```
}  
}  
  
private void Form1_Load(object sender, EventArgs e)  
{  
    ushort nCard = Dmc2410.d2410_board_init(); // 初始化DMC2410C  
    if ((nCard <= 0) || (nCard >= 8)) // 正常情况卡数在1- 8之间  
        MessageBox.Show("初始化DMC2410C卡失败！ ", "出错");  
    Dmc2410.d2410_set_pulse_outmode(0, 0); // 设定脉冲输出模式  
    Dmc2410.d2410_set_position(0, 0); // 设置X轴的脉冲位置为0  
    textBox1.Text = "0"; // 显示初始位置信息  
    Dmc2410.d2410_counter_config(0, 0); // 设置编码器信号模式  
    timer1.Interval = 55;  
    timer1.Stop();  
}  
  
private void timer1_Tick(object sender, EventArgs e)  
{  
    int Position = Dmc2410.d2410_get_position(0); // 读取X轴当前指令位置  
    textBox1.Text = Convert.ToString(Position); // 显示位置信息  
}  
  
private void button2_Click(object sender, EventArgs e)  
{  
    Dmc2410.d2410_decel_stop(0, 0.1); // 减速停止  
    Job1.Abort(); // 终止线程  
    Job2.Abort();  
    flagJob = 0;  
    button1.Visible = true;  
}  
  
private void Form1_FormClosed(object sender, FormClosedEventArgs e)  
{  
    if (flagJob == 1)  
    {  
        Job1.Abort(); // 终止线程  
        Job2.Abort(); // 终止线程  
    }  
    Dmc2410.d2410_board_close(); // 关闭DMC2410C  
}  
}
```

单线程和多线程2种方法编写的程序结果对比：

1. 多线程程序的响应速度快；（Stop键能快速响应、IO扫描周期为10ms）
2. 多线程程序的CPU占用时间小（少1/3，参见图24、25）；
3. 多线程程序结构较复杂。

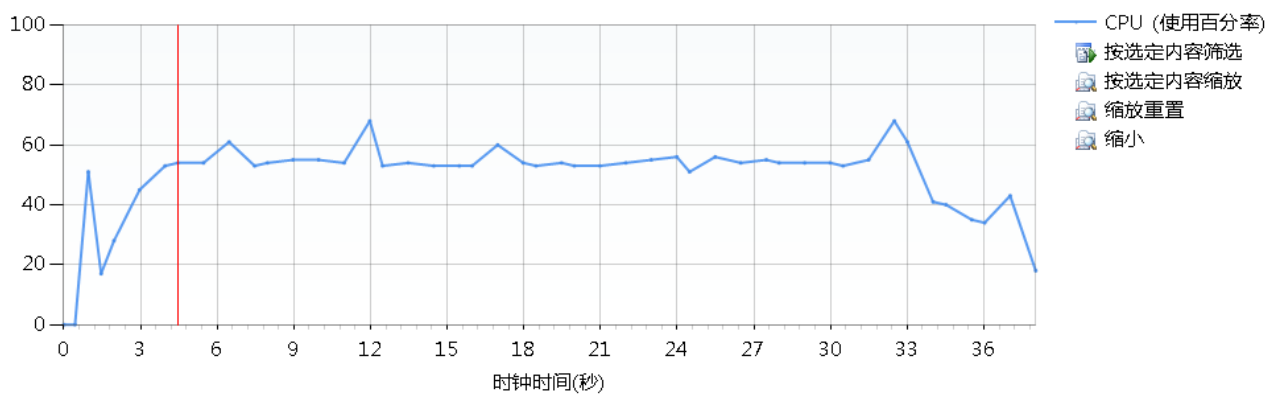


图24 单线程程序占用CPU的比例

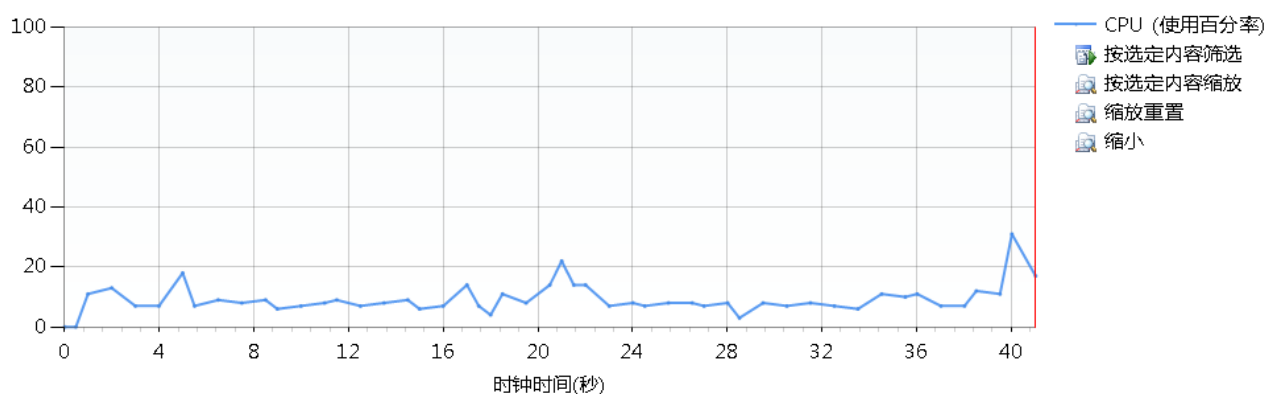


图25 多线程程序占用CPU的比例



第五讲：C#2010 高速采样的方法

十六. C#的高速采样:

在 C# 中定时器有 3 种:

- 1、基于 Windows 的定时器 (System.Windows.Forms.Timer)
- 2、基于服务器的定时器 (System.Timers.Timer)
- 3、线程定时器 (System.Threading.Timer)

第 1 种: 是通过 Windows 消息机制实现的。使用方便, 但精度不高。

第 2、3 种: 这 2 种定时器类似, 他们通过 .NET Thread Pool 实现, 对应用程序、消息没有特别要求。定时精度较高。System.Timers.Timer 还能够应用于 WinForm, 完全取代第一种定时器。他们的缺点是不支持控件直接拖放, 需要手工编写代码。

经过验证, 第 1 种定时器最短计时时间约 55ms; 精度差。

第 2、3 种定时器: 最短计时时间约 15ms。精度还是不够高。

在一篇文章中看到: “在 C++ 中使用 QueryPerformanceFrequency() 函数可获得高精度计数器。” 使用 “QueryPerformanceFrequency” 一词在 C# 帮助中搜索, 找到 Stopwatch 类。

实验表明: 使用 Stopwatch 类可以获得高精度时间! (VC++ 的类库就是强大!)

在 Pentium 2.9GHz 双核 CPU 的计算机上, Stopwatch 计时最小单位为 $1/2825673$ 秒, 即 0.354 微秒。

Stopwatch 的使用方法:

1. 用 Stopwatch.IsHighResolution 检测高分辨率计时器是否可用;
2. 调用 Start 方法 (或 Restart 方法) 启动时钟, 然后调用 Stop 方法停止时钟;
3. 最后使用属性 Stopwatch.ElapsedTicks 和 Stopwatch.Frequency 计算时间。

所用时间 (秒) = Stopwatch.ElapsedTicks / Stopwatch.Frequency

例程10: Stopwatch 实验代码

```
using System;  
using System.Collections.Generic;
```



```
using System.Linq;
using System.Text;
using System.Diagnostics;           // Stopwatch类的命名空间
using System.Threading;

namespace Stopwatch
{
    class Program
    {
        public static void DisplayTimerProperties()
        {
            // 显示Stopwatch的频率和分辨率
            if (Stopwatch.IsHighResolution) // IsHighResolution指示高分辨率计时器是否可用
                Console.WriteLine("Operations timed using the system's high-resolution performance counter.");
            else
                Console.WriteLine("Operations timed using the DateTime class.");
            long frequency = Stopwatch.Frequency;
            Console.WriteLine(" Timer frequency in ticks per second = {0}", frequency);
            long nanosecPerTick = (1000L * 1000L * 1000L) / frequency;
            Console.WriteLine(" Timer is accurate within {0} nanoseconds" + "\n", nanosecPerTick);
            Console.ReadKey();
        }

        static void Main(string[] args)
        {
            double PassTime;
            double sUnit = Stopwatch.Frequency; // 获取Stopwatch的频率，即每秒有多少个计时单位
            DisplayTimerProperties();
            Stopwatch myWatch = new Stopwatch(); // 实例化Stopwatch
            for (int i = 0; i < 20; i++)
            {
                myWatch.Restart(); // 将运行时间重置为零，然后开始测量时间
                //while (myWatch.ElapsedTicks < 28256730) //延时10秒
                //{
                //    Thread.Sleep(1); // 测量Stopwatch的精度
                //}
                Thread.Sleep(1); // 测量Sleep(1)的精度
                myWatch.Stop(); // 停止时间测量
                PassTime = myWatch.ElapsedTicks / sUnit; // 计算所用时间
                Console.WriteLine(PassTime.ToString()); // 显示结果
                Console.ReadKey(); // 等待按键按下
            }
        }
    }
}
```



```
}  
}
```

例程11：用Stopwatch高速采样，用Forms.Timer显示动态曲线。

子线程用Stopwatch控制时间，每1ms采样1次；

主线程的Forms.Timer每110毫秒触发一次，用DrawLines画100个点。结果见图25。

因为人的视觉暂留时间约0.1~0.4秒，故0.11秒画一段曲线，仍然觉得画曲线的速度很连续。

因为Forms.Timer时间不准确，曲线显示过程的时间和实际时间有差距（慢10%）；但是Stopwatch精确，所以，得到的数据是准确的！

高速采样的代码如下：

```
.....  
  
double sUnit = Stopwatch.Frequency;           // 获取Stopwatch的计时单位  
.....  
  
SamplingPos = new Thread(MyJob);               // 创建线程SamplingPos  
SamplingPos.Priority = ThreadPriority.Highest; // 设置线程优先级为最高  
Dmc2410.d2410_set_st_profile(Xch, InitSpeed, MaxSpeed, Tacc, Tdec, 0.3 * Tacc, 0.3 * Tdec);  
                                                // 设置速度、加减速时间段时间  
Dmc2410.d2410_ex_s_pmove(Xch, Distance, 0);    // 以S形速度曲线作相对运动  
SamplingPos.Start();                           // 运行线程SamplingPos  
timer1.Start();                                // 开启定时器定时画曲线  
.....  
  
private void MyJob()                           // 采样线程  
{  
    double waitTime = 0;  
    if (Stopwatch.IsHighResolution == false)  
    {  
        MessageBox.Show("不可使用Stopwatch! ");  
        return;  
    }  
    Stopwatch MyWatch = new Stopwatch();        // 实例化计时器MyWatch  
    MyWatch.Restart();                          // 计时器清零，开启计时器  
    for (int i = 0; i < 12000; i++)  
    {  
        DateTime[i] = MyWatch.ElapsedTicks;    // 记录时间值  
        DataPos[i] = ENC7480.Enc7480_Get_Encoder(Xch); // 读取X轴当前实际位置  
        do  
        {  
            waitTime = (MyWatch.ElapsedTicks - DateTime[i]) / sUnit;
```



```
        } while (waitTime < 0.001);           // 等待1毫秒。
    }
    MyWatch.Stop();                           // 采样数据已满，计时器停止
}
```

Forms.Timer画曲线的代码:

```
private void timer1_Tick(object sender, EventArgs e) // 110ms触发一次
{
    double temp0, temp1=0, temp2,temp3,temp4;
    PointF[] pointS = new PointF[100];
    PointF[] pointV = new PointF[100];
    Graphics g = pictureBox.CreateGraphics();    // 创建画板
    Pen RedPen = new Pen(Color.Red);
    Pen BluePen = new Pen(Color.Blue);
    g.TranslateTransform(1, 601); // 坐标平移，从左上角移至底边，但y轴方向还是朝下
    for (j = 0; j < 100; j++)
    {
        // 准备曲线的数据
        temp1 = DateTime[j0 + j] / sUnit;           // 时间t1
        temp0 = DateTime[j0 + j - 1] / sUnit;       // 时间t0
        xd = temp1 / Tscale;                       // 计算出时间的坐标
        temp3 = DataPos[j0 + j];                   // 位移x1
        yd = temp3 / Sscale;                       // 计算出位移的坐标
        temp2 = DataPos[j0 + j - 1];               // 位移x0
        temp4 = (temp3 - temp2) / (temp1 - temp0); // 计算速度
        vd = temp4 / Vscale;                       // 计算出速度的坐标
        pointS[j] = new PointF((float)xd, (float)(-1 * yd)); // 生成位移曲线数据
        pointV[j] = new PointF((float)xd, (float)(-1 * vd)); // 生成速度曲线数据
    }
    j0 = j0 + j - 1;
    g.DrawLines(RedPen, pointS); // 绘制位移曲线
    g.DrawLines(BluePen, pointV); // 绘制速度曲线
    if (temp1 > Tmax - 0.1)      // 曲线画完
    {
        timer1.Stop();           // 关闭定时器
        SamplingPos.Abort();      // 终止采样线程
        button1.Visible = true;
        button2.Visible = false;
    }
    textBoxPosition.Text = Convert.ToString(Dmc2410.d2410_get_position(Xch)); // 显示位置
}
```

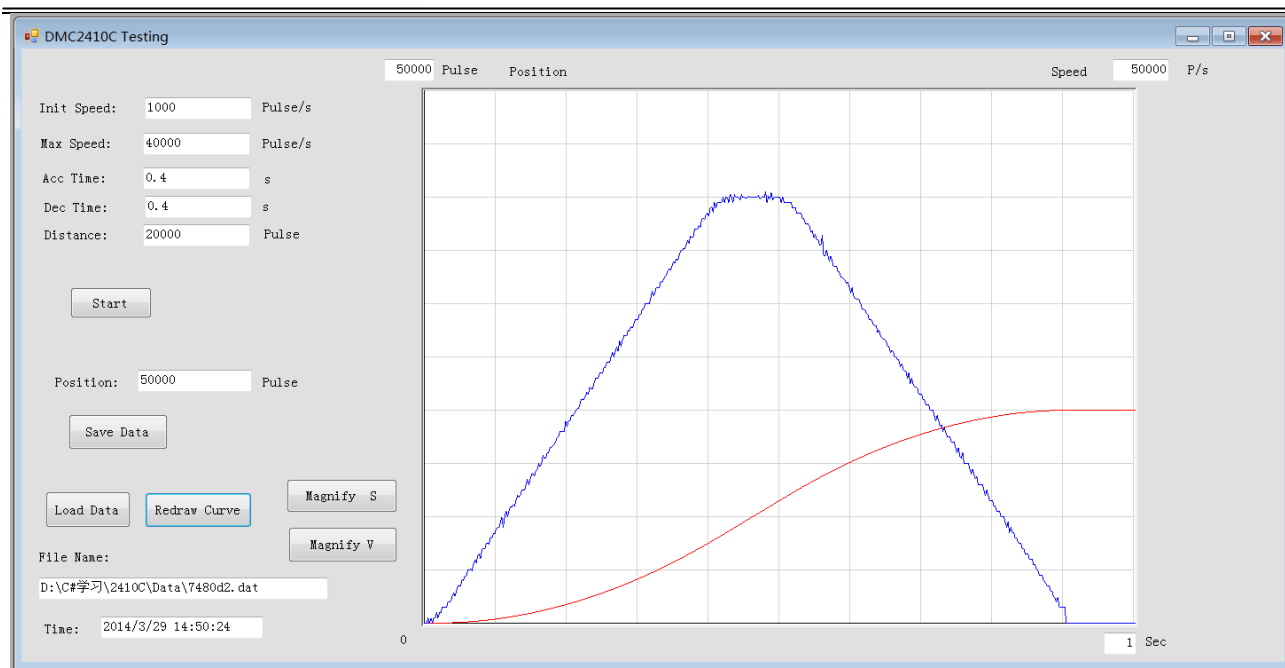



图26 用Stopwatch高速采样，用FormTimer实时显示曲线

十七. 常见问题：（C#的小bug）

1. 仅删除一个过程的代码，如：`private void Form1_Load(object sender, EventArgs e) { }`，会出错。
窗体设计器中的一条对应的代码，也要删除才行。

```
private void InitializeComponent()
{
    .....
    this.Load += new System.EventHandler(this.Form1_Load); // 这条代码也要删除
    .....
}
```

2. 将事件触发的代码直接拷贝在Form.cs中不行。

如：只是拷贝了这段代码，并不会在textBox1中显示Hello！也不报错。

```
private void Form1_Load(object sender, EventArgs e)
{
    textBox1.Text = "Hello!";
}
```

只有在Form1上双击Load事件，自动生成Form1_Load过程，再将textBox1.Text = "Hello!"; 拷入，才行！

3. 局部变量没有赋初值，会报错

例如：

```
private void button1_Click(object sender, EventArgs e)
{
```



```
int m, n ;  
if (m == 0) ← 报错
```

或者：

```
m = n + 1 ; ← 报错  
报错提示：使用了未赋值的局部变量。
```

```
int m, n=0 ;  
m = n + 1 ; ← 这样ok!
```

4. 如果程序有语法错误，不会执行代码格式自动对齐（Ctrl+E+D，Ctrl+E+F）

十八. 技巧

快捷键：

Ctrl+E+D：调整全部代码的格式（编辑→高级→设置文档的格式）

Ctrl+E+F：调整所选代码的格式（编辑→高级→设置选定内容的格式）

F1：查看帮助：将光标移至相关代码处，按键F1，可得相应的帮助内容。

F7：进入代码编辑窗口；

Shift+F7：进入窗体设计窗口；

F12：光标在方法名上，按F12跳方法定义代码处。

Ctrl+Home：光标跳至文件头；

Ctrl+End：光标跳至文件尾；

Ctrl+L：删除当前行；

Ctrl+C、Ctrl+V、Ctrl+X、Ctrl+Z：和Office一样。大家都会用的！

充分利用网络上的例程和帮助文件：

C#类库资源丰富，要充分利用。先有自己的想法，再到网上搜索例程；最后仔细阅读C#的帮助文件。

不断学习、实践，就能成为C#高手！

第六讲：C#2010 程序生成安装软件

十九. 生成安装软件的方法(Express版不支持该功能)

用 C#开发完成应用程序后，要打包项目文件，生成安装软件。让没有安装 C#的电脑也能使用 C#编写的程序。

1. 添加安装文件

首先打开一个要打包的项目，然后点击菜单上的“文件”→“添加”→“新建项目”→“已安装的模板”→“其他项目类型”→“安装和部署”→“Visual Studio Installer”→“安装项目”。

然后为安装项目起一个名称，再选择一个存储位置，如图 27 所示。最后点击确定。得到安装项目的 3 个文件夹，如图 28 所示。

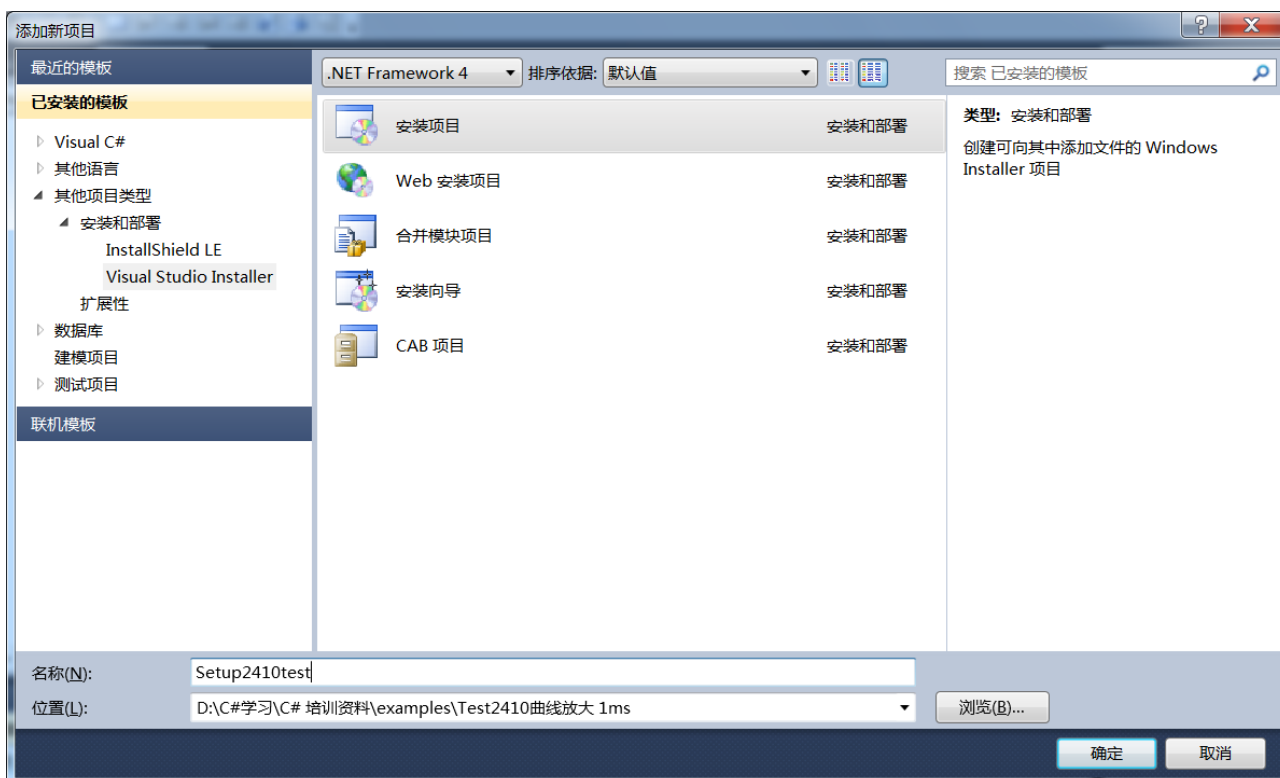


图27 生成安装项目的界面

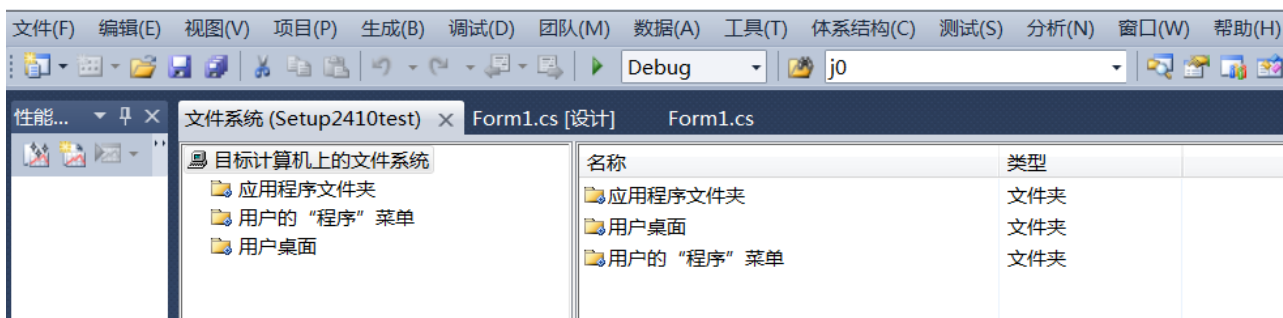


图28 安装项目的3个文件夹

2. 配置应用程序文件夹

选中图 27 左边的“应用程序文件夹”，在右边空白位置右键 “添加” → “项目输出” → “主输出”，如图 29 所示。

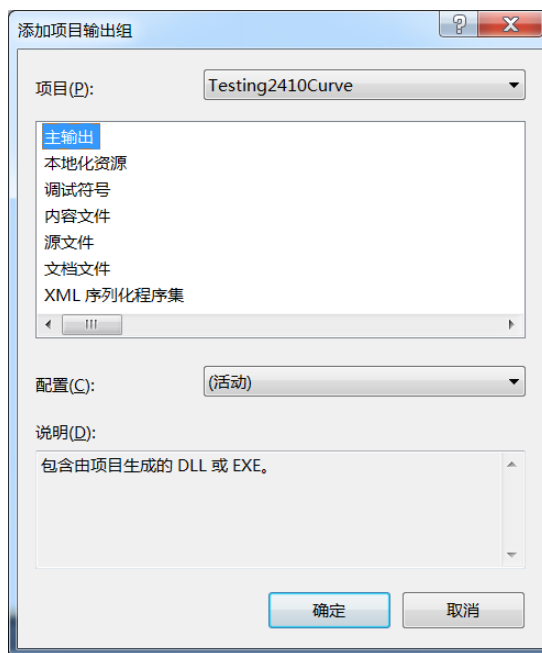


图29 添加项目输出组的对话框

最后点击“确定”，得到如图 30 所示结果。

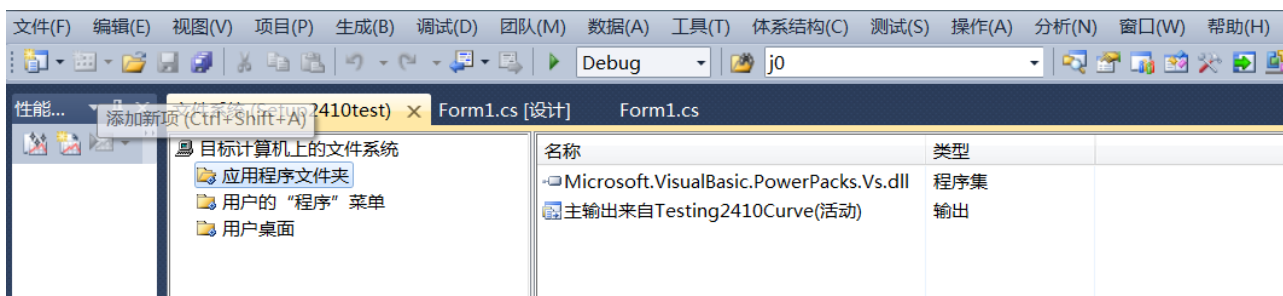


图30 应用程序文件夹中自动生成2个文件

然后添加执行文件和动态链接库。右键“添加”→“文件”，选择要添加的文件。执行文件和动态链接库在项目文件中的 bin\Debug 文件夹中，如图 31 所示。

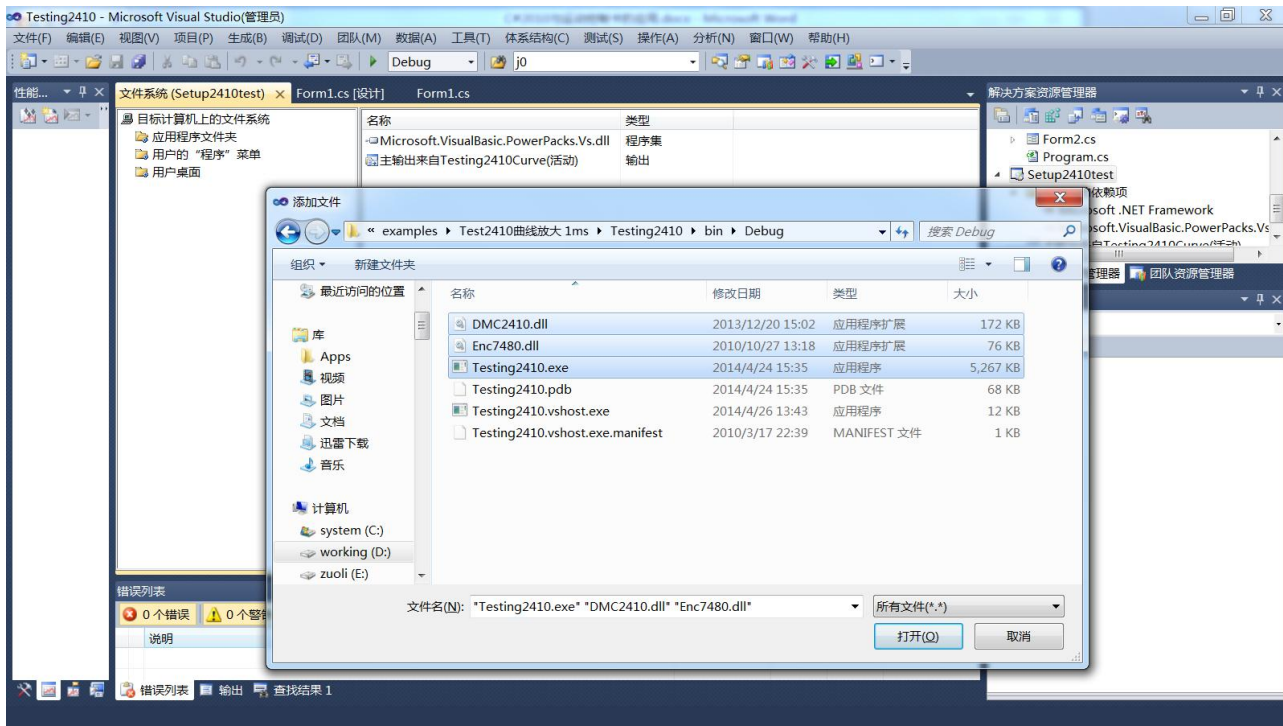


图31 添加执行文件和动态链接库

最好再添加一个图标文件*.ico。现在应用程序文件夹就准备好了。如图 32 所示。

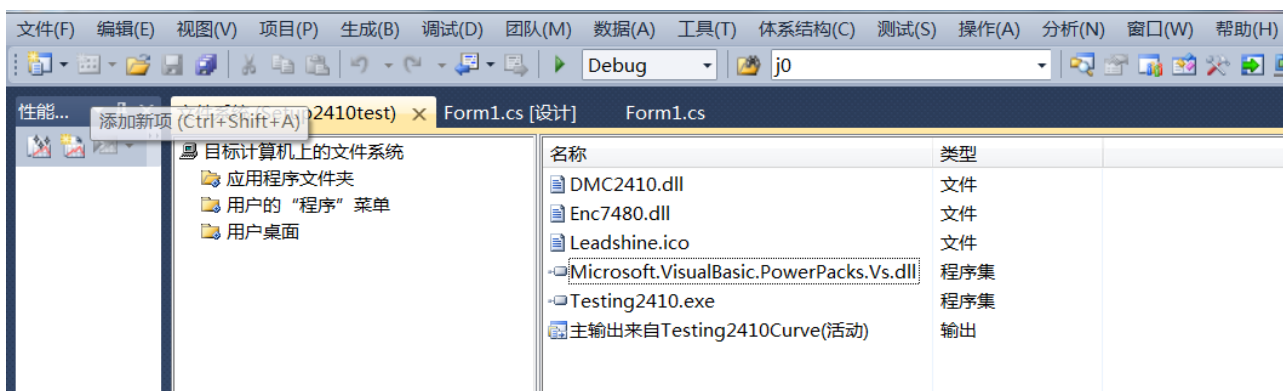


图32 应用程序文件夹准备就绪

3. 创建快捷方式

快捷方式可以创建两个：“开始”菜单中一个，桌面一个。

1) 创建“开始”菜单中的快捷方式：

点击左边中间的‘用户的“程序”菜单’，在右面空白处点击右键，“添加”→“创建新

的快捷方式”，然后选择“应用程序文件夹”下的程序的 exe 文件，如图 33 所示。

然后给快捷方式命名，如图 34 所示。

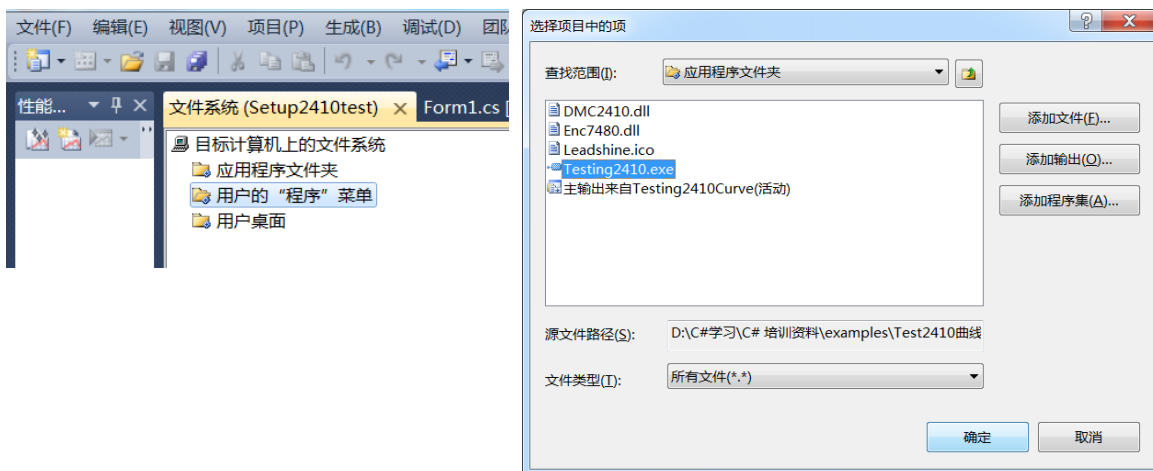


图33 添加快捷方式

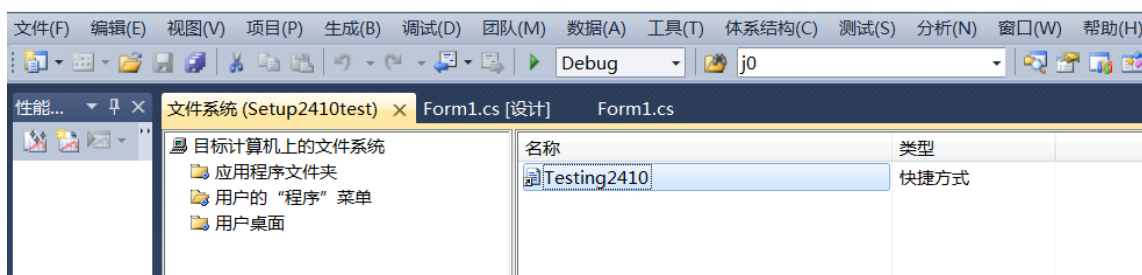


图34 给快捷方式命名

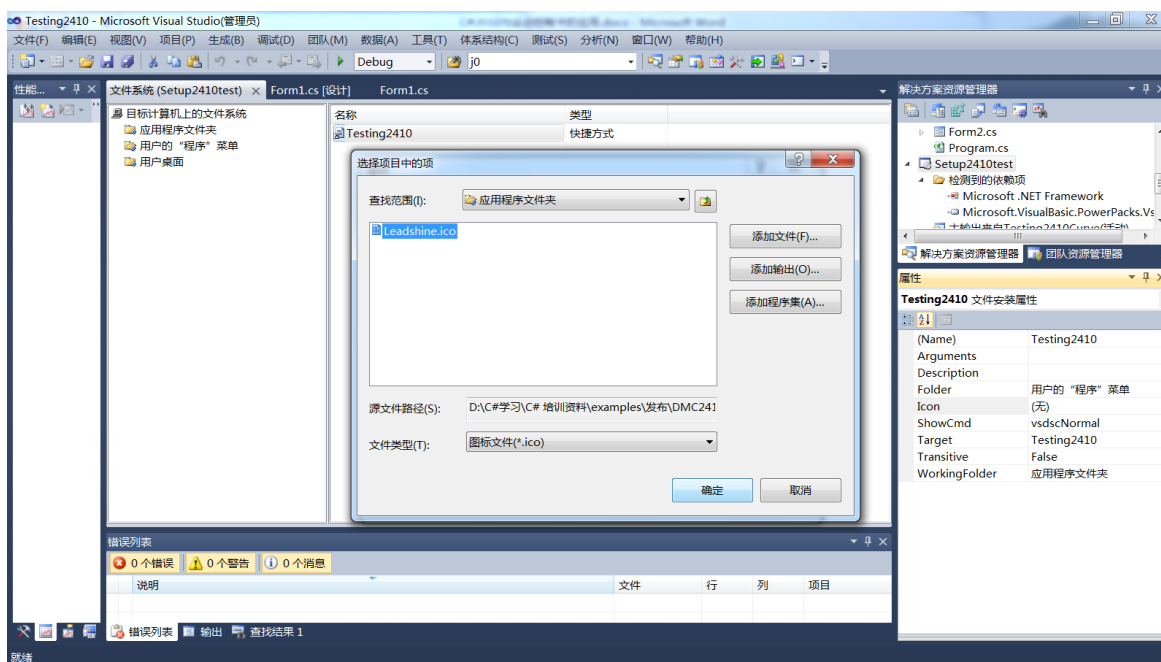


图35 给快捷方式添加图标

再设置快捷图标。右键点击新创建的快捷方式的名称，打开“属性窗口”，通过 Icon 属性设置快捷方式的图标，如图 35 所示。

2) 创建桌面的快捷方式图标:

过程同创建“开始”菜单中的快捷方式一样。

4. 更改默认安装目录:

右键点击“应用程序文件夹”→“属性”→属性窗口里的“DefaultLocation”属性，可以更改默认的安装目录。如图 36 所示。

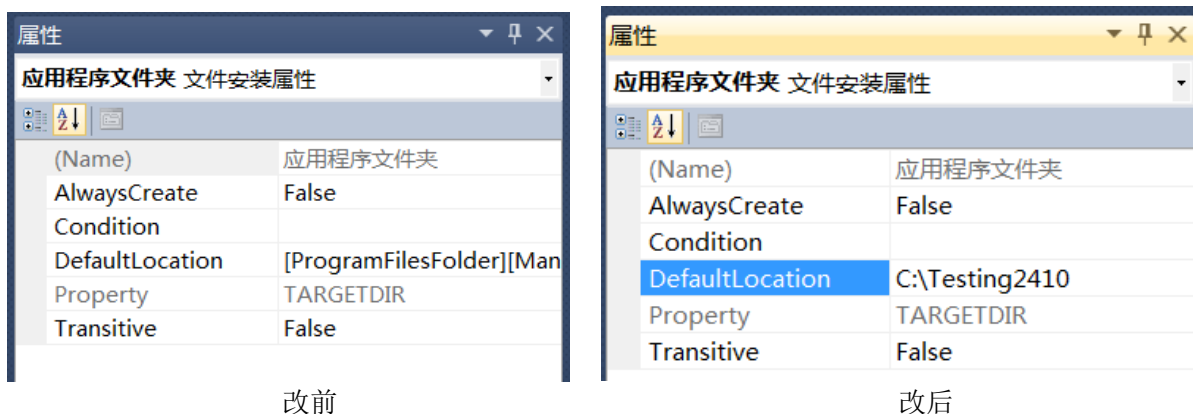


图36 修改默认安装目录

5. 打包时，如果需要把对应的.NET Framework 打到安装包中。这样即使是 Window XP 系统也可正常运行。操作如下:

右键点击安装项目名（在解决方案资源管理器中）→“属性”→“系统必备”，勾选对应的系统必备组件，然后指定系统必备组件的安装位置，一般选“从我的应用程序相同的位置下载系统必备组件”，那么打包时将会从程序中把组件打入包中；安装软件时，PC 机无需联网。如图 37 所示。

最后点击“生成”→ 生成 XXXX 打包程序，即可。如图 38 所示。生成的安装文件在“安装位置”文件夹中的 Debug 文件夹中。

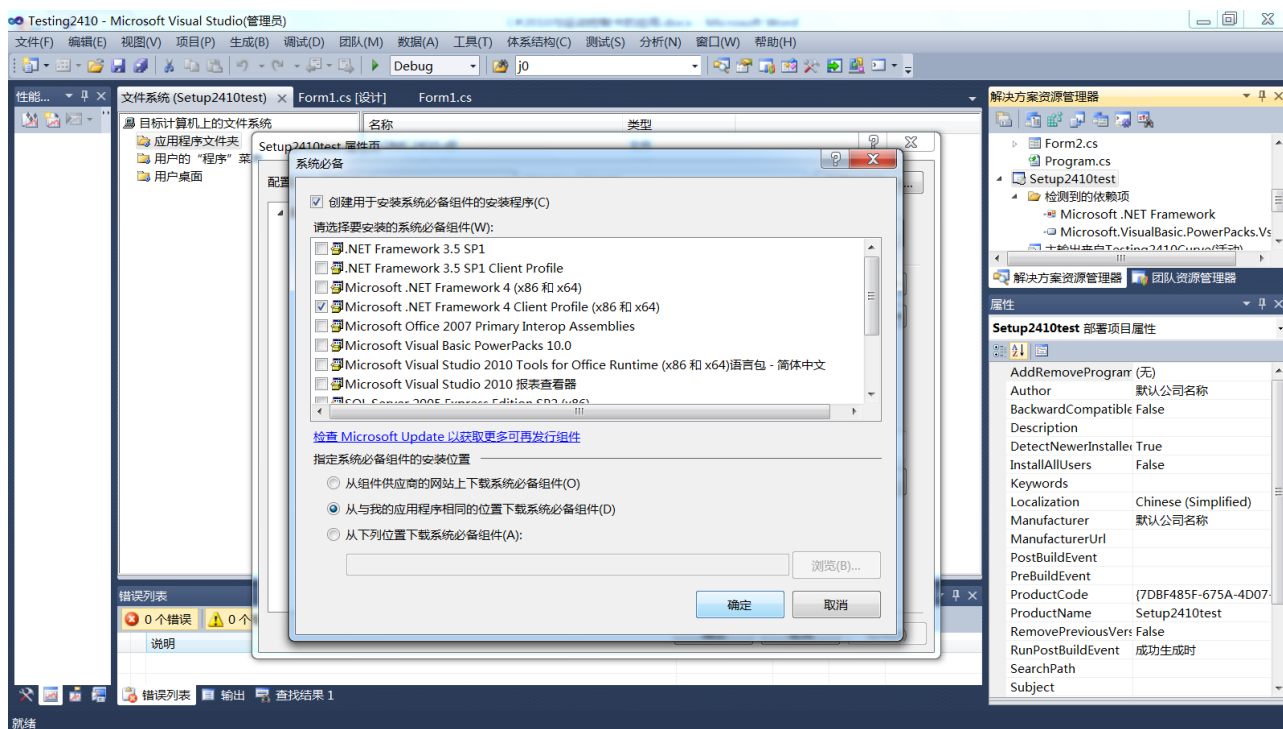


图37 将.NET Framework打入安装包中



图38 生成打包文件

6. 用 winrar 将生成的安装程序压缩成一个执行文件

如图 39 所示，用 Visual Studio 生成的安装文件有 1 个 setup.exe 文件、1 个*.msi 文件和 2 个文件夹；而 setup.exe 安装时，又依赖于.msi 文件，另外两个文件夹是对应的 .NET Framework 组件。这样给用户安装时拷贝的内容较多，容易出错。把这些安装文件压缩成一个执行文件，就比较方便。

可以用 winrar 的“自解压格式压缩文件”来实现。过程如下：

1) 将要压缩的文件及文件夹全部选中，点击右键 → “添加到压缩文件”，在打开的压缩面

板的“常规”选项卡中勾选“创建自解压格式压缩文件”；压缩方式最好选择“存储”。如图 39 所示。

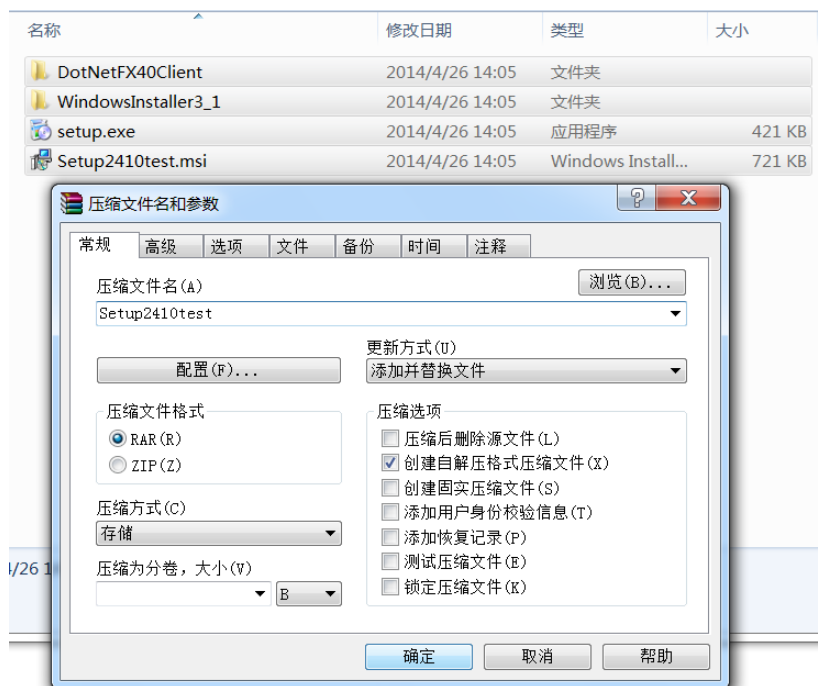


图39 压缩打包文件

2) 设置运行文件：再切换到“高级”选项卡，点击“自解压选项”；“设置”里设置程序解压后运行的文件，如图 40 所示。选“常规”，填写解压路径，如图 41 所示。



图40 填写压缩运行文件

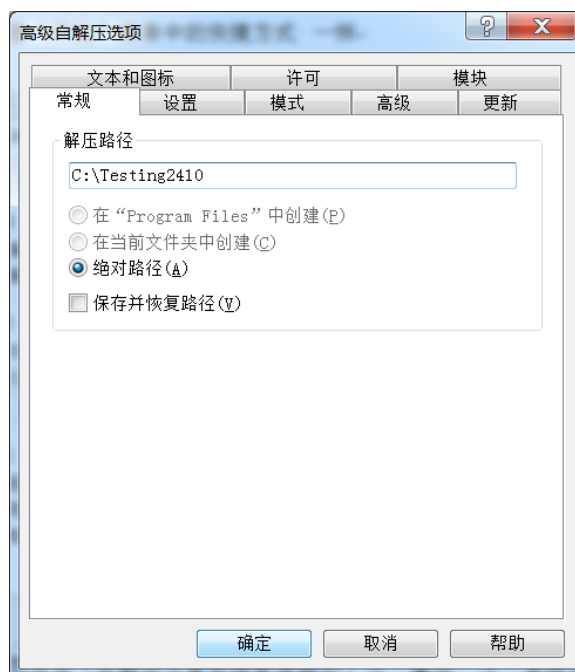


图41 填写压缩路径

3) 设置安装程序文件的图标:

切换到“文本和图标”，点击“从文件加载自解压文件图标”后的“浏览”按钮，选择安装程序文件的图标，如图 42 所示。

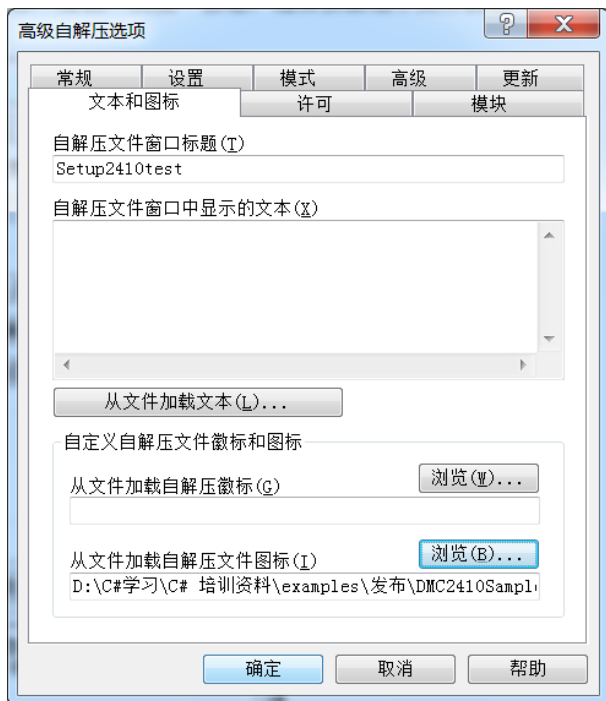


图42 填写图标文件



图43 选择解压选项

4) 最后切换到“模式”下，勾选“解包到临时文件夹”和“全部隐藏”，如图 43 所示。然后点击“确定”。

7. 大功告成

此时我们会看到在 Debug 目录下多出一个执行文件，如图 44 所示。这就是我们最后所要的安装程序。将该文件拷贝到其他计算机中，直接点击运行就 OK 啦！

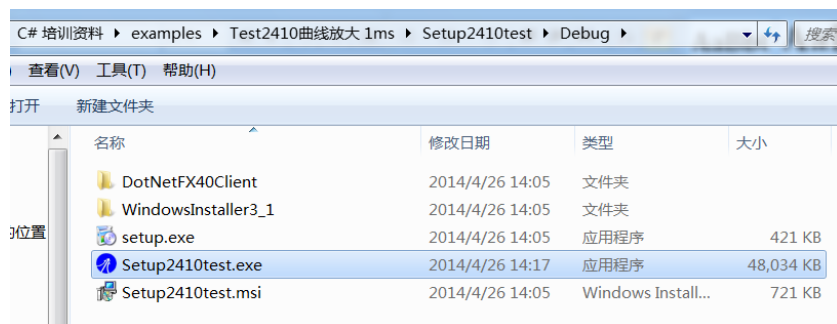


图44 最后生成的打包文件