C#教程

第一章 C#语言基础

本章介绍 C#语言的基础知识，希望具有 C 语言的读者能够基本掌握 C#语言，并以此为基础，能够进一步学习用 C#语言编写 window 应用程序和 Web 应用程序。当然仅靠一章的内容就完全掌握 C#语言是不可能的，如需进一步学习 C#语言，还需要认真阅读有关 C#语言的专著。

1.1 C#语言特点

Microsoft. NET(以下简称. NET)框架是微软提出的新一代 Web 软件开发模型，C#语言是.NET 框架中新一代的开发工具。C#语言是一种现代、面向对象的语言，它简化了 C++语言在类、命名空间、方法重载和异常处理等方面的操作，它摒弃了 C++的复杂性，更易使用，更少出错。它使用组件编程，和 VB 一样容易使用。C#语法和 C++和 JAVA 语法非常相似，如果读者用过 C++和 JAVA，学习 C#语言应是比较轻松的。

用 C#语言编写的源程序，必须用 C#语言编译器将 C#源程序编译为中间语言(MicroSoft Intermediate Language, MSIL)代码，形成扩展名为 exe 或 dll 文件。中间语言代码不是 CPU 可执行的机器码，在程序运行时，必须由通用语言运行环境(Common Language Runtime，CLR)中的既时编译器(JUST IN Time，JIT)将中间语言代码翻译为 CPU 可执行的机器码，由 CPU 执行。CLR 为 C#语言中间语言代码运行提供了一种运行时环境，C#语言的 CLR 和 JAVA 语言的虚拟机类似。这种执行方法使运行速度变慢，但带来其它一些好处，主要有：

- 通用语言规范(Common Language Specification，CLS)：.NET 系统包括如下语言：C#、C++、VB、J#，他们都遵守通用语言规范。任何遵守通用语言规范的语言源程序，都可编译为相同的中间语言代码，由 CLR 负责执行。只要为其它操作系统编制相应的 CLR，中间语言代码也可在其它系统中运行。
- 自动内存管理：CLR 内建垃圾收集器，当变量实例的生命周期结束时，垃圾收集器负责收回不被使用的实例占用的内存空间。不必象 C 和 C++语言，用语句在堆中建立的实例，必须用语句释放实例占用的内存空间。也就是说，CLR 具有自动内存管理功能。
- 交叉语言处理：由于任何遵守通用语言规范的语言源程序，都可编译为相同的中间语言代码，不同语言设计的组件，可以互相通用，可以从其它语言定义的类派生出本语言的新类。由于中间语言代码由 CLR 负责执行，因此异常处理方法是一致的，这在调试一种语言调用另一种语言的子程序时，显得特别方便。
- 增加安全：C#语言不支持指针，一切对内存的访问都必须通过对象的引用变量来实现，只允许访问内存中允许访问的部分，这就防止病毒程序使用非法指针访问私有成员。也避免指针的误操作产生的错误。CLR 执行中间语言代码前，要对中间语言代码的安全性，完整性进行验证，防止病毒对中间语言代码的修改。
- 版本支持：系统中的组件或动态联接库可能要升级，由于这些组件或动态联接库都要在注册表中注册，由此可能带来一系列问题，例如，安装新程序时自动安装新组件替换旧组件，有可能使某些必须使用旧组件才可以运行的程序，使用新组件运行不了。在. NET 中这些组件或动态联接库不必在注册表中注册，每个程序都可以使用自带的组件或动态联接库，只要把这些组件或动态联接库放到运行程序所在文件夹的子文件夹 bin 中，运行程序就自动使用在 bin 文件夹中的组件或动态联接库。由于不需要在注册表中注册，软件的安装也变得容易了，一般将运行程序及库文件拷贝到指定文件夹中就可以了。

- 完全面向对象：不象 C++语言，即支持面向过程程序设计，又支持面向对象程序设计，C#语言是完全面向对象的，在 C#中不再存在全局函数、全区变量，所有的函数、变量和常量都必须定义在类中，避免了命名冲突。C#语言不支持多重继承。

1.2 编写控制台应用程序

1.2.1使用 SDK 命令行工具编写控制台程序

第一个程序总是非常简单的，程序首先让用户通过键盘输入自己的名字，然后程序在屏幕上打印一条欢迎信息。程序的代码是这样的:

```
using System;//导入命名空间。//为C#语言新增解释方法，解释到本行结束
class Welcome//类定义，类的概念见下一节
{ /*解释开始，和C语言解释用法相同
  解释结束*/
  static void Main()//主程序，程序入口函数，必须在一个类中定义
  { Console.WriteLine("请键入你的姓名：");//控制台输出字符串
    Console.ReadLine();//从键盘读入数据，输入回车结束
    Console.WriteLine("欢迎！");
  }
}
```

可以用任意一种文本编辑软件完成上述代码的编写，然后把文件存盘，假设文件名叫做welcome.cs，C#源文件是以cs作为文件的扩展名。和C语言相同，C#语言是区分大小写的。高级语言总是依赖于许多在程序外部预定义的变量和函数。在C或C++中这些定义一般放到头文件中，用#include语句来导入这个头文件。而在C#语言中使用using语句导入名字空间，using System语句意义是导入System名字空间，C#中的using语句的用途与C++中#include语句的用途基本类似，用于导入预定义的变量和函数，这样在自己的程序中就可以自由地使用这些变量和函数。如果没有导入名字空间的话我们该怎么办呢？程序还能保持正确吗？答案是肯定的，那样的话我们就必须把代码改写成下面的样子：

```
class Welcome
{ static void Main()
  { System.Console.WriteLine("请键入你的姓名：");
    System.Console.ReadLine();
    System.Console.WriteLine("欢迎！");
  }
}
```

也就是在每个Console前加上一个前缀System.，这个小原点表示Console是作为System的成员而存在的。C#中抛弃了C和C++中繁杂且极易出错的操作符象::和->等，C#中的复合名字一律通过.来连接。System是.Net平台框架提供的最基本的名字空间之一，有关名字空间的详细使用方法将在以后详细介绍，这里只要学会怎样导入名字空间就足够了。

程序的第二行class Welcome声明了一个类，类的名字叫做Welcome。C#程序中每个变量或函数都必须属于一个类，包括主函数Main()，不能象C或C++那样建立全局变量。C#语言程序总是从Main()方法开始执行，一个程序中不允许出现两个或两个以上的Main()方法。请牢记C#中Main()方法必须被包含在一个类中,Main第一个字母必须大写,必须是一个静态方法，

也就是Main()方法必须使用static修饰。static void Main()是类Welcome中定义的主函数。静态方法意义见以后章节。

程序所完成的输入输出功能是通过Console类来完成的，Console是在名字空间System中已经定义好的一个类。Console类有两个最基本的方法WriteLine和ReadLine。ReadLine表示从输入设备输入数据，WriteLine则用于在输出设备上输出数据。

如果在电脑上安装了Visual Studio.Net，则可以在集成开发环境中直接选择快捷键或菜单命令编译并执行源文件。如果您不具备这个条件，那么至少需要安装Microsoft.Net Framework SDK，这样才能够运行C#语言程序。Microsoft.Net Framework SDK中内置了C#的编译器csc.exe，下面让我们使用这个微软提供的命令行编译器对程序welcome.cs进行编译。假设已经将welcome.cs文件保存在d:\Charp目录下，启动命令行提示符，在屏幕上输入一行命令：d:回车，cd Charp回车，键入命令：

C:\WINNT\Microsoft.NET\Framework\v1.0.3705\csc welcome.cs

如果一切正常welcome.cs文件将被编译，编译后生成可执行文件Welcome.exe。可以在命令提示符窗口运行可执行文件Welcome.exe，屏幕上出现一行字符提示您输入姓名：请键入你的姓名：输入任意字符并按下回车键，屏幕将打印出欢迎信息：欢迎！

注意，和我们使用过的绝大多数编译器不同，在 C#中编译器只执行编译这个过程，而在 C 和 C++中要经过编译和链接两个阶段。换而言之 C#源文件并不被编译为目标文件.obj，而是直接生成可执行文件.exe 或动态链接库.dll，C#编译器中不需要包含链接器。

1.2.1使用 Visual Studio.Net 建立控制台程序

(1) 运行 Visual Studio.Net 程序，出现如图 1.2.2A 界面。

(2) 单击新建项目按钮，出现如图 1.2.2B 对话框。在项目类型(P)编辑框中选择 Visual C#项目，在模板(T)编辑框中选择控制台应用程序，在名称(N)编辑框中键入 e1，在位置(L)编辑框中键入 D:\csarp，必须预先创建文件夹 D:\csarp。也可以单击浏览按钮，在打开文件对话框中选择文件夹。单击确定按钮，创建项目。出现如图 1.2.2C 界面。编写一个应用程序，可能包含多个文件，才能生成可执行文件，所有这些文件的集合叫做一个项目。

(3) 修改 class1.cs 文件如下，有阴影部分是新增加的语句，其余是集成环境自动生成的。

```
using System;
namespace e1
{
    /// <summary>
    /// Class1 的摘要说明。
    /// </summary>
    class Class1
    {
        /// <summary>
        /// 应用程序的主入口点。
        /// </summary>
        [STAThread]
        static void Main(string[] args)
```

```
        {
            //
            // TODO: 在此处添加代码以启动应用程序
            //
            Console.WriteLine("请键入你的姓名：");
            Console.ReadLine();
            Console.WriteLine("欢迎！");
        }
    }
}
```

(4) 按 CTRL+F5 键，运行程序，如右图，和 1.2.1 节运行效果相同。屏幕上出现一行字符，提示您输入姓名：请键入你的姓名：输入任意字符并按下回车键，屏幕将打印出欢迎信息：欢迎！输入回车退出程序。

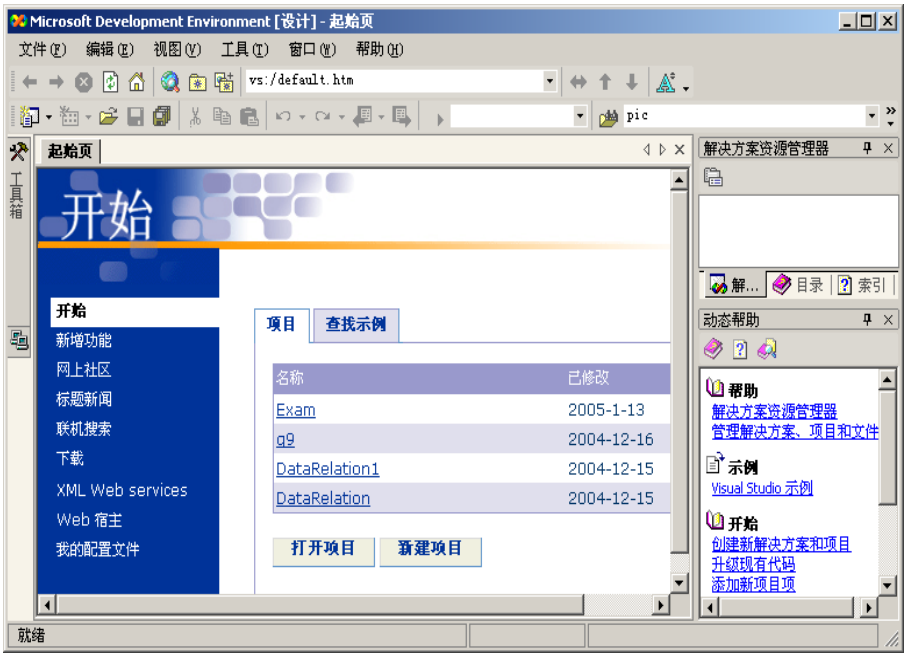


图 1.2.2A

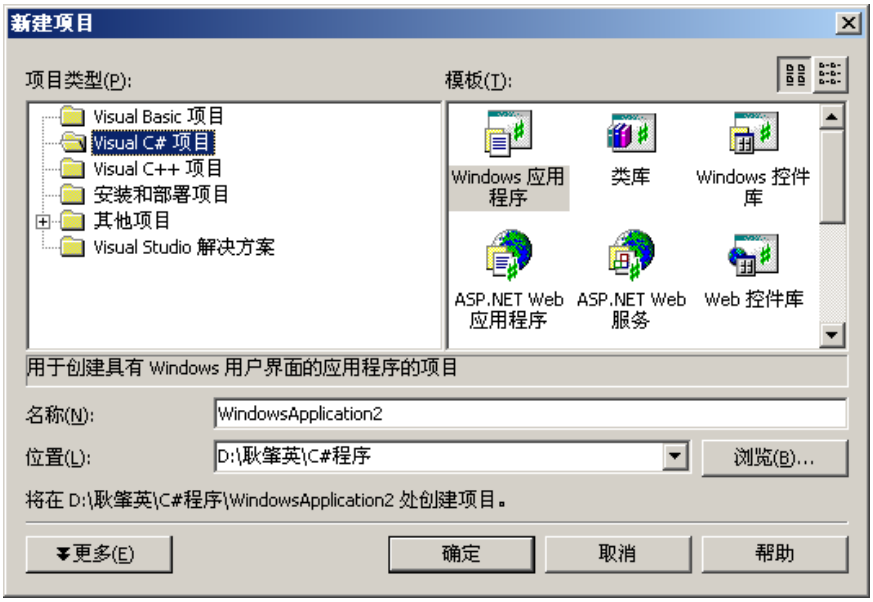

新建项目
项目类型(P):
模板(T):
Visual Basic 项目
Visual C# 项目
Visual C++ 项目
安装和部署项目
其他项目
Visual Studio 解决方案
Windows 应用程序
类库
Windows 控件库
ASP.NET Web 应用程序
ASP.NET Web 服务
Web 控件库
用于创建具有 Windows 用户界面的应用程序的项目
名称(N):
WindowsApplication2
位置(L):
D:\耿肇英\C#程序
浏览(B)...
将在 D:\耿肇英\C#程序\WindowsApplication2 处创建项目。
更多(E)
确定
取消
帮助

图 1.2.2B

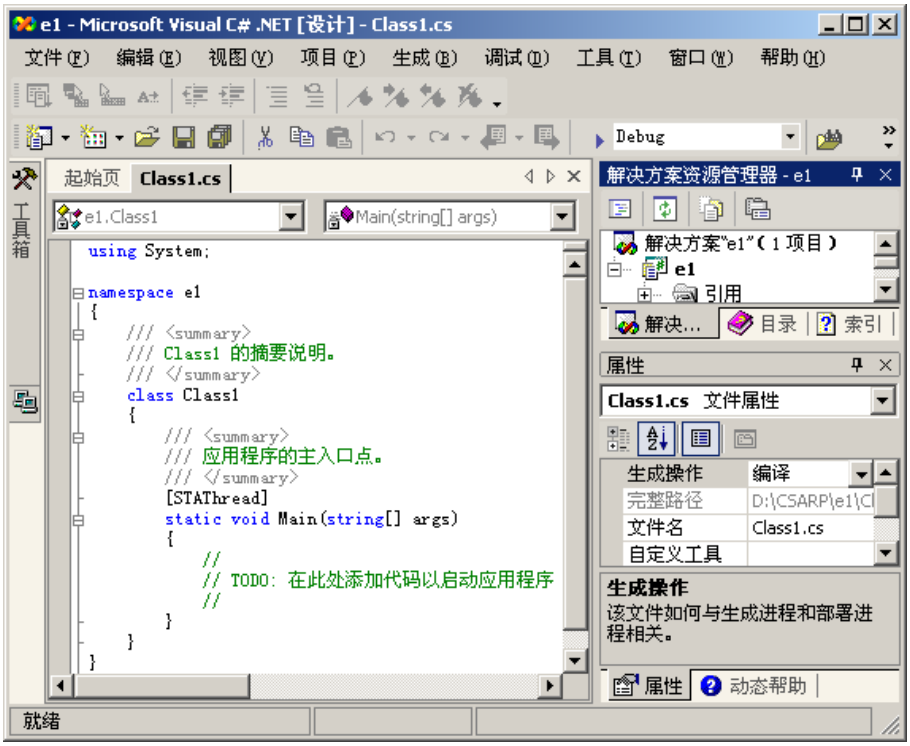

e1 - Microsoft Visual C# .NET [设计] - Class1.cs
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 工具(T) 窗口(W) 帮助(H)
Debug
起始页
Class1.cs
e1.Class1
Main(string[] args)
using System;
namespace e1
{
/// <summary>
/// Class1 的摘要说明。
/// </summary>
class Class1
{
/// <summary>
/// 应用程序的主入口点。
/// </summary>
[STAThread]
static void Main(string[] args)
{
//
// TODO: 在此处添加代码以启动应用程序
//
}
}
}
解决方案资源管理器 - e1
解决方案"e1"(1 项目)
e1
引用
解决...
目录
索引
属性
Class1.cs 文件属性
生成操作
编译
完整路径
D:\CSARP\e1\Cl
文件名
Class1.cs
自定义工具
生成操作
该文件如何与生成进程和部署进程相关。
属性
动态帮助
就绪

图 1.2.2C

1.3 类的基本概念

C#语言是一种现代、面向对象的语言。面向对象程序设计方法提出了一个全新的概念：类，它的主要思想是将数据（数据成员）及处理这些数据的相应方法（函数成员）封装到类中，类的实例则称为对象。这就是我们常说的封装性。

1.3.1 类的基本概念

类可以认为是对结构的扩充，它和 C 中的结构最大的不同是：类中不但可以包括数据，还包括处理这些数据的函数。类是对数据和处理数据的方法（函数）的封装。类是对某一类具有相同特性和行为的事物的描述。例如，定义一个描述个人情况的类 Person 如下：

```
using System;
class Person//类的定义，class是保留字，表示定义一个类，Person是类名
{   private string name="张三";//类的数据成员声明
    private int age=12;//private表示私有数据成员
    public void Display()//类的方法(函数)声明，显示姓名和年龄
    {   Console.WriteLine("姓名:{0},年龄：{1}",name,age);
    }
    public void SetName(string PersonName)//修改姓名的方法(函数)
    {   name=PersonName;
    }
    public void SetAge(int PersonAge)
    {   age=PersonAge;
    }
}
```

Console.WriteLine("姓名:{0},年龄：{1}",name,age)的意义是将第二个参数变量 name 变为字符串填到{0}位置，将第三个参数变量 age 变为字符串填到{1}位置，将第一个参数表示的字符串在显示器上输出。

大家注意，这里我们实际定义了一个新的数据类型，为用户自己定义的数据类型，是对个人的特性和行为的描述，他的类型名为 Person，和 int，char 等一样为一种数据类型。用定义新数据类型 Person 类的方法把数据和处理数据的函数封装起来。类的声明格式如下：

属性 类修饰符 class 类名{类体}

其中，关键字 class、类名和类体是必须的，其它项是可选项。类修饰符包括 new、public、protected、internal、private、abstract 和 sealed，这些类修饰符以后介绍。类体用于定义类的成员。

1.3.2 类成员的存取控制

一般希望类中一些数据不被随意修改，只能按指定方法修改，既隐蔽一些数据。同样一些函数也不希望被其它类程序调用，只能在类内部使用。如何解决这个问题呢？可用访问权限控制字，常用的访问权限控制字如下：private(私有)，public(公有)。在数据成员或函数成员前增加访问权限控制字，可以指定该数据成员或函数成员的访问权限。

私有数据成员只能被类内部的函数使用和修改，私有函数成员只能被类内部的其它函数调用。类的公有函数成员可以被类的外部程序调用，类的公有数据成员可以被类的外部程序直接使用修改。公有函数实际是一个类和外部通讯的接口，外部函数通过调用公有函数，按照预先设定好的方法修改类的私有成员。对于上述例子，name 和 age 是私有数据成员，只能通过公有函数 SetName()和 SetAge()修改，既它们只能按指定方法修改。

这里再一次解释一下封装，它有两个意义，第一是把数据和处理数据的方法同时定义在类中。第二是用访问权限控制字使数据隐蔽。

1.3.3 类的对象

Person 类仅是一个用户新定义的数据类型，由它可以生成 Person 类的实例，C#语言叫对象。用如下方法声明类的对象：Person OnePerson=new Person()；此语句的意义是建立 Person 类对象，返回对象地址赋值给 Person 类变量 OnePerson。也可以分两步创建 Person 类的对象：Person OnePerson；OnePerson=new Person()；OnePerson 虽然存储的是 Person 类对象地址，但不是 C 中的指针，不能象指针那样可以进行加减运算，也不能转换为其它类型地址，它是引用型变量，只能引用(代表)Person 对象，具体意义参见以后章节。和 C、C++不同，C#只能用此种方法生成类对象。

在程序中，可以用 OnePerson.方法名或 OnePerson.数据成员名访问对象的成员。例如：OnePerson.Display(),公用数据成员也可以这样访问。注意，C#语言中不包括 C++语言中的->符号。

1.3.4 类的构造函数和析构函数

在建立类的对象时，需做一些初始化工作，例如对数据成员初始化。这些可以用构造函数来完成。每当用 new 生成类的对象时，自动调用类的构造函数。因此，可以把初始化的工作放到构造函数中完成。构造函数和类名相同，没有返回值。例如可以定义 Person 类的构造函数如下：

```
public Person(string Name,int Age)//类的构造函数，函数名和类同名，无返回值。
{    name=Name;
     age=Age;
}
```

当用 Person OnePerson=new Person(“张五”，20)语句生成 Person 类对象时，将自动调用以上构造函数。请注意如何把参数传递给构造函数。

变量和类的对象都有生命周期，生命周期结束，这些变量和对象就要被撤销。类的对象被撤销时，将自动调用析构函数。一些善后工作可放在析构函数中完成。析构函数的名字为~类名，无返回类型，也无参数。Person 类的析构函数为~ Person()。C#中类析构函数不能显示地被调用，它是被垃圾收集器撤销不被使用的对象时自动调用的。

1.3.5 类的构造函数的重载

在 C#语言中，同一个类中的函数，如果函数名相同，而参数类型或个数不同，认为是不同的函数，这叫函数重载。仅返回值不同，不能看作不同的函数。这样，可以在类定义中，

定义多个构造函数，名字相同，参数类型或个数不同。根据生成类的对象方法不同，调用不同的构造函数。例如可以定义 Person 类没有参数的构造函数如下：

```
public Person()//类的构造函数，函数名和类同名，无返回值。
{   name="张三";
    age=12;
}
```

用语句 Person OnePerson=new Person("李四",30)生成对象时，将调用有参数的构造函数，而用语句 Person OnePerson=new Person()生成对象时，调用无参数的构造函数。由于析构函数无参数，因此，析构函数不能重载。

1.3.6 使用 Person 类的完整的例子

下边用一个完整的例子说明 Person 类的使用：(VisualStudio.Net 编译通过)

```
using System;
namespace e1//定义以下代码所属命名空间，意义见以后章节
{   class Person
    {   private String name="张三";//类的数据成员声明
        private int age=12;
        public void Display()//类的方法(函数)声明，显示姓名和年龄
        {   Console.WriteLine("姓名:{0},年龄：{1}",name,age);
        }
        public void SetName(string PersonName)//指定修改姓名的方法(函数)
        {   name=PersonName;
        }
        public void SetAge(int PersonAge)//指定修改年龄的方法(函数)
        {   age=PersonAge;
        }
        public Person(string Name,int Age)//构造函数,函数名和类同名,无返回值
        {   name=Name;
            age=Age;
        }
        public Person()//类的构造函数重载
        {   name="田七";
            age=12;
        }
    }
    class Class1
    {   static void Main(string[] args)
        {   Person OnePerson=new Person("李四",30);//生成类的对象
            OnePerson.Display();
//下句错误，在其它类(Class1类)中，不能直接修改Person类中的私有成员。
            //OnePerson.name="王五";
//只能通过Person类中公有方法SetName修改Person类中的私有成员name。
            OnePerson.SetName("王五");
```

```
            OnePerson.SetAge(40);
            OnePerson.Display();
            OnePerson=new Person();
            OnePerson.Display();
        }
    }
}
```

键入CTRL+F5运行后，显示的效果是：

姓名：李四,年龄：30
姓名：王五,年龄：40
姓名：田七,年龄：12

1.4 C#的数据类型

从大的方面来分，C#语言的数据类型可以分为三种：值类型，引用类型，指针类型，指针类型仅用于非安全代码中。本节重点讨论值类型和引用类型。

1.4.1 值类型和引用类型区别

在 C#语言中，值类型变量存储的是数据类型所代表的实际数据，值类型变量的值(或实例)存储在栈(Stack)中，赋值语句是传递变量的值。引用类型(例如类就是引用类型)的实例，也叫对象，不存在栈中，而存储在可管理堆(Managed Heap)中，堆实际上是计算机系统中的空闲内存。引用类型变量的值存储在栈(Stack)中，但存储的不是引用类型对象，而是存储引用类型对象的引用，即地址，和指针所代表的地址不同，引用所代表的地址不能被修改，也不能转换为其它类型地址，它是引用型变量，只能引用指定类对象，引用类型变量赋值语句是传递对象的地址。见下例：

```
using System;
class MyClass//类为引用类型
{   public int a=0;
}
class Test
{   static void Main()
    {   f1();
    }
    static public void f1()
    {   int v1=1;//值类型变量 v1，其值 1 存储在栈(Stack)中
        int v2=v1;//将 v1 的值(为 1)传递给 v2，v2=1,v1 值不变。
        v2=2;//v2=2,v1 值不变。
        MyClass r1=new MyClass();//引用变量 r1 存储 MyClass 类对象的地址
        MyClass r2=r1;//r1 和 r2 都代表是同一个 MyClass 类对象
        r2.a=2;//和语句 r1.a=2 等价
    }
}
```

存储在栈中的变量，当其生命周期结束，自动被撤销，例如，v1 存储在栈中，v1 和函数 f1 同生命周期，退出函数 f1，v1 不存在了。但在堆中的对象不能自动被撤销。因此 C 和 C++语言，在堆中建立的对象，不使用时必须用语句释放对象占用的存储空间。.NET 系统 CLR 内建垃圾收集器，当对象的引用变量被撤销，表示对象的生命周期结束，垃圾收集器负责收回不被使用的对象占用的存储空间。例如，上例中引用变量 r1 及 r2 是 MyClass 类对象的引用，存储在栈中，退出函数 f1，r1 和 r2 都不存在了，在堆中的 MyClass 类对象也就被垃圾收集器撤销。也就是说，CLR 具有自动内存管理功能。

1.4.2 值类型变量分类

C#语言值类型可以分为以下几种：

- 简单类型(Simple types)
  简单类型中包括：数值类型和布尔类型(bool)。数值类型又细分为：整数类型、字符类型(char)、浮点数类型和十进制类型(decimal)。
- 结构类型(Struct types)
- 枚举类型(Enumeration types)

C#语言值类型变量无论如何定义，总是值类型变量，不会变为引用类型变量。

1.4.3 结构类型

结构类型和类一样，可以声明构造函数、数据成员、方法、属性等。结构和类的最根本的区别是结构是值类型，类是引用类型。和类不同，结构不能从另外一个结构或者类派生，本身也不能被继承，因此不能定义抽象结构，结构成员也不能被访问权限控制字 protected 修饰，也不能用 virtual 和 abstract 修饰结构方法。在结构中不能定义析构函数。虽然结构不能从类和结构派生，可是结构能够继承接口，结构继承接口的方法和类继承接口的方法基本一致。下面例子定义一个点结构 point：

```
using System;
struct point//结构定义
{   public int x,y;//结构中也可以声明构造函数和方法，变量不能赋初值
}
class Test
{   static void Main()
    {   point P1;
        P1.x=166;
        P1.y=111;
        point P2;
        P2=P1;//值传递，使 P2.x=166,P2.y=111
        point P3=new point();//用 new 生成结构变量 P3，P3 仍为值类型变量
    }//用 new 生成结构变量 P3 仅表示调用默认构造函数，使 x=y==0。
}
```

1.4.4 简单类型

简单类型也是结构类型，因此有构造函数、数据成员、方法、属性等，因此下列语句int i=int.MaxValue;string s=i.ToString()是正确的。即使一个常量，C#也会生成结构类型的实例，因此也可以使用结构类型的方法，例如：string s=13.ToString()是正确的。简单类型包括：整数类型、字符类型、布尔类型、浮点数类型、十进制类型。见下表：

保留字	System 命名空间中的名字	字节数	取值范围
sbyte	System.Sbyte	1	-128~127
byte	System.Byte	1	0~255
short	System.Int16	2	-32768~32767
ushort	System.UInt16	2	0~65535
int	System.Int32	4	-2147483648~2147483647
uint	System.UInt32	4	0~4292967295
long	System.Int64	8	-9223372036854775808~9223372036854775808
ulong	System.UInt64	8	0~18446744073709551615
char	System.Char	2	0~65535
float	System.Single	4	3.4E-38~3.4E+38
double	System.Double	8	1.7E-308~1.7E+308
bool	System.Boolean		(true,false)
decimal	System.Decimal	16	正负 $1.0\cdot10^{-28}$ 到 $7.9\cdot10^{28}$之间

C#简单类型使用方法和C、C++中相应的数据类型基本一致。需要注意的是：

- 和C语言不同，无论在何种系统中，C#每种数据类型所占字节数是一定的。
- 字符类型采用Unicode字符集，一个Unicode标准字符长度为16位。
- 整数类型不能隐式被转换为字符类型(char)，例如char c1=10是错误的，必须写成：char c1=(char)10，char c='A'，char c='\x0032';char c='\u0032'。
- 布尔类型有两个值：false,true。不能认为整数0是false，其它值是true。bool x=1是错误的，不存在这种写法，只能写成x=true 或x=false。
- 十进制类型(decimal)也是浮点数类型，只是精度比较高，一般用于财政金融计算。

1.4.5 枚举类型

C#枚举类型使用方法和C、C++中的枚举类型基本一致。见下例：

```
using System;
class Class1
{    enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
     //使用Visual Studio.Net,enum语句添加在[STAThread]前边
     static void Main(string[] args)
     {    Days day=Days.Tue;
          int x=(int)Days.Tue;//x=2
          Console.WriteLine("day={0},x={1}",day,x);//显示结果为:day=Tue,x=4
     }
}
```

在此枚举类型 Days 中，每个元素的默认类型为 int，其中 Sun=0，Mon=1，Tue=2，依此类推。也可以直接给枚举元素赋值。例如：

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

在此枚举中，Sun=1，Mon=2，Tue=3，Wed=4，等等。和 C、C++中不同，C#枚举元素类型可以是 byte、sbyte、short、ushort、int、uint、long 和 ulong 类型，但不能是 char 类型。见下例：

```
enum Days:byte{Sun,Mon,Tue,Wed,Thu,Fri,Sat};//元素为字节类型
```

1.4.6 值类型的初值和默认构造函数

所有变量都要求必须有初值，如没有赋值，采用默认值。对于简单类型，sbyte、byte、short、ushort、int、uint、long 和 ulong 默认值为 0，char 类型默认值是(char)0，float 为 0.0f，double 为 0.0d，decimal 为 0.0m，bool 为 false，枚举类型为 0，在结构类型和类中，数据成员的数值类型变量设置为默认值，引用类型变量设置为 null。

可以显示的赋值，例如 int i=0。而对于复杂结构类型，其中的每个数据成员都按此种方法赋值，显得过于麻烦。由于数值类型都是结构类型，可用 new 语句调用其构造函数初始化数值类型变量，例如：int j=new int()。请注意，用 new 语句并不是把 int 变量变为引用变量，j 仍是值类型变量，这里 new 仅仅是调用其构造函数。所有的数值类型都有默认的无参数的构造函数，其功能就是为该数值类型赋初值为默认值。对于自定义结构类型，由于已有默认的无参数的构造函数，不能再定义无参数的构造函数，但可以定义有参数的构造函数。

1.4.7 引用类型分类

C#语言中引用类型可以分为以下几种：

- 类：C#语言中预定义了一些类：对象类(object 类)、数组类、字符串类等。当然，程序员可以定义其它类。
- 接口。
- 代表。

C#语言引用类型变量无论如何定义，总是引用类型变量，不会变为值类型变量。C#语言引用类型对象一般用运算符 new 建立，用引用类型变量引用该对象。本节仅介绍对象类型(object 类型)、字符串类型、数组。其它类型在其它节中介绍。

1.4.8 对象类(object 类)

C#中的所有类型(包括数值类型)都直接或间接地以object类为基类。对象类(object类)是所有其它类的基类。任何一个类定义，如果不指定基类，默认object为基类。继承和基类的概念见以后章节。C#语言规定，基类的引用变量可以引用派生类的对象(注意，派生类的引用变量不可以引用基类的对象)，因此，对一个object的变量可以赋予任何类型的值：

```
int x =25;
object obj1;
obj1=x;
object obj2= 'A';
```

object 关键字是在命名空间 System 中定义的，是类 System.Object 的别名。

1.4.9 数组类

在进行批量处理数据的时候，要用到数组。数组是一组类型相同的有序数据。数组按照数组名、数据元素的类型和维数来进行描述。C#语言中数组是类System.Array类对象，比如声明一个整型数数组：int[] arr=new int[5];实际上生成了一个数组类对象，arr是这个对象的引用(地址)。

在C#中数组可以是一维的也可以是多维的，同样也支持数组的数组，即数组的元素还是数组。一维数组最为普遍，用的也最多。我们先看一个一维数组的例子：

```
using System;
class Test
{ static void Main()
  { int[] arr=new int[3];//用new运算符建立一个3个元素的一维数组
    for(int i=0;i<arr.Length;i++)//arr.Length是数组类变量，表示数组元素个数
        arr[i]=i*i;//数组元素赋初值，arr[i]表示第i个元素的值
    for (int i=0;i<arr.Length;i++)//数组第一个元素的下标为0
        Console.WriteLine("arr[{0}]={1}",i,arr[i]);
  }
}
```

这个程序创建了一个int类型3个元素的一维数组，初始化后逐项输出。其中arr.Length表示数组元素的个数。注意数组定义不能写为C语言格式：int arr[]。程序的输出为：

```
arr[0] = 0
arr[1] = 1
arr[2] = 4
```

上面的例子中使用的是一维数组，下面介绍多维数组：

```
string[] a1;//一维string数组类引用变量a1
string[,] a2;//二维string数组类引用变量a2
a2=new string[2,3];
a2[1,2]="abc";
string[,,] a3;//三维string数组类引用变量a3
string[][] j2;//数组的数组，即数组的元素还是数组
string[][][][] j3;
```

在数组声明的时候，可以对数组元素进行赋值。看下面的例子：

```
int[] a1=new int[]{1,2,3};//一维数组，有3个元素。
int[] a2=new int[3]{1,2,3};//此格式也正确
int[] a3={1,2,3};//相当于int[] a3=new int[]{1,2,3};
int[,] a4=new int[,]{{1,2,3},{4,5,6}};//二维数组，a4[1,1]=5
int[][] j2=new int[3][];//定义数组j2,有三个元素，每个元素都是一个数组
j2[0]=new int[]{1,2,3};//定义第一个元素，是一个数组
j2[1]=new int[]{1, 2, 3, 4, 5, 6};//每个元素的数组可以不等长
j2[2]=new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9};
```

1.4.10 字符串类(string 类)

C#还定义了一个基本的类 string，专门用于对字符串的操作。这个类也是在名字空间 System 中定义的，是类 System.String 的别名。字符串应用非常广泛，在 string 类的定义中封装了许多方法，下面的一些语句展示了 string 类的一些典型用法：

- 字符串定义

```
string s;//定义一个字符串引用类型变量 s
s="Zhang";//字符串引用类型变量 s 指向字符串"Zhang"
string  FirstName="Ming";
string  LastName="Zhang";
string  Name=FirstName+"  "+LastName;//运算符+已被重载
string  SameName=Name;
char[] s2={'计','算','机','科','学'};
string s3=new String(s2);
```

- 字符串搜索

```
string s="ABC科学";
int i=s.IndexOf("科");
```

搜索"科"在字符串中的位置，因第一个字符索引为0，所以"A"索引为0，"科"索引为3，因此这里i=3，如没有此字符串i=-1。注意C#中，ASCII和汉字都用2字节表示。

- 字符串比较函数

```
string s1="abc";
string s2="abc";
int n=string.Compare(s1,s2);//n=0
```

n=0表示两个字符串相同，n小于零，s1<s2，n大于零，s1>s2。此方法区分大小写。也可用如下办法比较字符串：

```
string s1="abc";
string s="abc";
string s2="不相同";
if(s==s1)//还可用!=。虽然String是引用类型，但这里比较两个字符串的值
    s2="相同";
```

- 判断是否为空字符串

```
string s="";
string s1="不空";
if(s.Length==0)
    s1="空";
```

- 得到子字符串或字符

```
string s="取子字符串";
string sb=s.Substring(2,2);//从索引为2开始取2个字符，Sb="字符",s内容不变
char sb1=s[0];//sb1='取'
Console.WriteLine(sb1);//显示：取
```

- 字符串删除函数

```
string s="取子字符串";
string sb=s.Remove(0,2);//从索引为0开始删除2个字符，Sb="字符串",s内容不变
```

- 插入字符串

```
string s="计算机科学";
string s1=s.Insert(3,"软件");//s1="计算机软件科学", s内容不变
```

- 字符串替换函数

```
string s="计算机科学";
string s1=s.Replace("计算机","软件");//s1="软件科学", s内容不变
```

- 把 String 转换为字符数组

```
string S="计算机科学";
char[] s2=S.ToCharArray(0,S.Length);//属性Length为字符类对象的长度
```

- 其它数据类型转换为字符串

```
int i=9;
string s8=i.ToString();//s8="9"
float n=1.9f;
string s9=n.ToString();//s8="1.9"
```

其它数据类型都可用此方法转换为字符类对象

- 大小写转换

```
string s="AaBbCc";
string s1=s.ToLower();//把字符转换为小写, s 内容不变
string s2=s.ToUpper();//把字符转换为大写, s 内容不变
```

- 删除所有的空格

```
string s="A   bc ";
s.Trim();//删除所有的空格
```

string 类其它方法的使用请用帮助系统查看,方法是打开 Visual Studio.Net 的代码编辑器, 键入 string, 将光标移到键入的字符串 string 上, 然后按 F1 键。

1.4.11 类型转换

在编写 C#语言程序中，经常会碰到类型转换问题。例如整型数和浮点数相加，C#会进行隐式转换。详细记住那些类型数据可以转换为其它类型数据，是不可能的，也是不必要的。程序员应记住类型转换的一些基本原则，编译器在转换发生问题时，会给出提示。C#语言中类型转换分为：隐式转换、显示转换、加框(boxing)和消框(unboxing)等三种。

一. 隐式转换

隐式转换就是系统默认的、不需要加以声明就可以进行的转换。例如从int类型转换到long类型就是一种隐式转换。在隐式转换过程中，转换一般不会失败，转换过程中也不会导致信息丢失。例如：

```
int i=10;
long l=i;
```

二. 显示转换

显式类型转换，又叫强制类型转换。与隐式转换正好相反，显式转换需要明确地指定转换类型，显示转换可能导致信息丢失。下面的例子把长整形变量显式转换为整型：

```
long l=5000;
int i=(int)l;//如果超过int取值范围, 将产生异常
```

三. 加框(boxing)和消框(unboxing)

加框(boxing)和消框(unboxing)是C#语言类型系统提出的核心概念,加框是值类型转换

为object(对象)类型，消框是object(对象)类型转换为值类型。有了加框和消框的概念，对任何类型的变量来说最终我们都可以看作是object类型。

1 加框操作

把一个值类型变量加框也就是创建一个object对象，并将这个值类型变量的值复制给这个object对象。例如：

```
int i=10;
object obj=i;//隐式加框操作，obj为创建的object对象的引用。
```

我们也可以用显式的方法来进行加框操作，例如：

```
int i =10;
object obj=object(i);//显式加框操作
```

值类型的值加框后，值类型变量的值不变，仅将这个值类型变量的值复制给这个object对象。我们看一下下面的程序：

```
using System
class Test
{   public static void Main()
    {   int n=200;
        object o=n;
        o=201;//不能改变n
        Console.WriteLine("{0},{1}",n,o);
    }
}
```

输出结果为：200，201。这就证明了值类型变量n和object类对象o都独立存在着。

2. 消框操作

和加框操作正好相反，消框操作是指将一个对象类型显式地转换成一个值类型。消框的过程分为两步：首先检查这个object对象，看它是否为给定的值类型的加框值，如是，把这个对象的值拷贝给值类型的变量。我们举个例子来看看一个对象消框的过程：

```
int i=10;
object obj=i;
int j=(int)obj;//消框操作
```

可以看出消框过程正好是加框过程的逆过程，必须注意加框操作和消框操作必须遵循类型兼容的原则。

3. 加框和消框的使用

定义如下函数：

```
void Display(Object o)//注意，o为Object类型
{   int x=(int)o;//消框
    System.Console.WriteLine("{0},{1}",x,o);
}
```

调用此函数：int y=20;Display(y);在此利用了加框概念，虚参被实参替换：Object o=y，也就是说，函数的参数是Object类型，可以将任意类型实参传递给函数。

1.5 运算符

C#语言和C语言的运算符用法基本一致。以下重点讲解二者之间不一致部分。

1.5.1 运算符分类

与C语言一样，如果按照运算符所作用的操作数个数来分，C#语言的运算符可以分为以下几种类型：

- 一元运算符：一元运算符作用于一个操作数，例如：-X、++X、X--等。
- 二元运算符：二元运算符对两个操作数进行运算，例如：x+y。
- 三元运算符：三元运算符只有一个：x? y:z。

C#语言运算符的详细分类及操作符从高到低的优先级顺序见下表。

类别	操作符
初级操作符	(x) x.y f(x) a[x] x++ x-- new type of sizeof checked unchecked
一元操作符	+ - ! ~ ++x --x (T)x
乘除操作符	* / %
加减操作符	+ -
移位操作符	<< >>
关系操作符	< > <= >= is as
等式操作符	== !=
逻辑与操作符	&
逻辑异或操作符	^
逻辑或操作符	|
条件与操作符	&&
条件或操作符	||
条件操作符	?:
赋值操作符	= *= /= %= += -= <<= >>= &= ^= |=

1.5.2 测试运算符 is

is操作符用于动态地检查表达式是否为指定类型。使用格式为：e is T，其中e是一个表达式，T是一个类型，该式判断e是否为T类型，返回值是一个布尔值。例子：

```
using System;
class Test
{ public static void Main()
    {   Console.WriteLine(1 is int);
        Console.WriteLine(1 is float);
        Console.WriteLine(1.0f is float);
        Console.WriteLine(1.0d is double);
    }
}
```

输出为：

```
True
False
True
True
```

1.5.3 typeof 运算符

typeof操作符用于获得指定类型在system名字空间中定义的类型名字，例如：

```
using System;
class Test
{   static void Main()
    {   Console.WriteLine(typeof(int));
        Console.WriteLine(typeof(System.Int32));
        Console.WriteLine(typeof(string));
        Console.WriteLine(typeof(double[]));
    }
}
```

产生如下输出，由输出可知int和System.int32是同一类型。

```
System.Int32
System.Int32
System.String
System.Double[]
```

1.5.4 溢出检查操作符 checked 和 unchecked

在进行整型算术运算(如+、-、*、/等)或从一种整型显式转换到另一种整型时，有可能出现运算结果超出这个结果所属类型值域的情况，这种情况称之为溢出。整型算术运算表达式可以用checked或unchecked溢出检查操作符，决定在编译和运行时是否对表达式溢出进行检查。如果表达式不使用溢出检查操作符或使用了checked操作符，常量表达式溢出，在编译时将产生错误，表达式中包含变量，程序运行时执行该表达式产生溢出，将产生异常提示信息。而使用了unchecked操作符的表达式语句，即使表达式产生溢出，编译和运行时都不会产生错误提示。但这往往会出现一些不可预期的结果，所以使用unchecked操作符要小心。下面的例子说明了checked和unchecked操作符的用法：

```
using System;
class Class1
{   static void Main(string[] args)
    {   const int x=int.MaxValue;
        unchecked//不检查溢出
        {   int z=x*2;//编译时不产生编译错误，z=-2
            Console.WriteLine("z={0}",z);//显示-2
        }
        checked//检查溢出
        {   int z1=(x*2);//编译时会产生编译错误
            Console.WriteLine("z={0}",z1);
        }
    }
}
```

1.5.5 new 运算符

new操作符可以创建值类型变量、引用类型对象，同时自动调用构造函数。例如：

```
int x=new int();//用new创建整型变量x，调用默认构造函数
Person C1=new Person ();//用new建立的Person类对象。Person 变量C1对象的引用
int[] arr=new int[2];//数组也是类，创建数组类对象，arr是数组对象的引用
```

需注意的是，int x=new int()语句将自动调用int结构不带参数的构造函数，给x赋初值0，x仍是值类型变量，不会变为引用类型变量。

1.5.6 运算符的优先级

当一个表达式包含多种操作符时，操作符的优先级控制着操作符求值的顺序。例如，表达式x+y*z按照x+(y*z)顺序求值，因为*操作符比+操作符有更高的优先级。这和数学运算中的先乘除后加减是一致的。1.5.1节中的表总结了所有操作符从高到低的优先级顺序。

当两个有相同优先级的操作符对操作数进行运算时，例如x+y-z，操作符按照出现的顺序由左至右执行，x+y-z按(x+y)-z进行求值。赋值操作符按照右接合的原则，即操作按照从右向左的顺序执行。如x=y=z按照x=(y=z)进行求值。建议在写表达式的时候，如果无法确定操作符的实际顺序，则尽量采用括号来保证运算的顺序，这样也使得程序一目了然，而且自己在编程时能够思路清晰。

1.6 程序控制语句

C#语言控制语句和C基本相同，使用方法基本一致。C#语言控制语句包括：if语句、swith语句、while语句、do…while语句、for语句、foreach语句、break语句、continue语句、goto语句、return语句、异常处理语句等，其中foreach语句和异常语句是C#语言新增加控制语句。本节首先介绍一下这些语句和C语言的不同点，然后介绍C#语言新增的控制语句。

1.6.1 和 C 语言的不同点

- 与C不同，if语句、while语句、do…while语句、for语句中的判断语句，一定要用布尔表达式，不能认为0为false，其它数为true。
- switch语句不再支持遍历,C和C++语言允许switch语句中case标签后不出现break语句,但C#不允许这样,它要求每个case标签项后使用break语句或goto跳转语句,即不允许从一个case自动遍历到其它case，否则编译时将报错。switch语句的控制类型,即其中控制表达式的数据类型可以是sbyte、byte、short、ushort、uint、long、ulong、char、string或枚举类型。每个case标签中的常量表达式必须属于或能隐式转换成控制类型。如果有两个或两个以上case标签中的常量表达式值相同，编译时将会报错。执行switch语句，首先计算switch表达式，然后与case后的常量表达式的值进行比较，执行第一个与之匹配的case分支下的语句。如果没有case常量表达式的值与之匹配,则执行dafault分支下的语句，如果没有dafault语句，则退出switch语句。switch语句中可以没有dafault语句，但最多只能有一个dafault语句。见下例：

```
using System;
class class1
{   static void Main()
    {   System.Console.WriteLine("请输入要计算天数的月份");
        string s=System.Console.ReadLine();
        string s1="";
        switch(s)
        {   case "1": case "3": case "5":
            case "7": case "8": case "10":
            case "12"://共用一条语句
                s1="31";break;
            case "2":
                s1="28";break;
            case "4": case "6": case "9":
                goto case "11";//goto语句仅为说明问题，无此必要
            case "11":
                s1="30";break;
            default:
                s1="输入错误";break;
        }
        System.Console.WriteLine(s1);
    }
}
```

1.6.2 foreach 语句

foreach语句是C#语言新引入的语句，C和C++中没有这个语句，它借用Visual Basic中的foreach语句。语句的格式为：

foreach(类型 变量名 in 表达式) 循环语句

其中表达式必须是一个数组或其它集合类型，每一次循环从数组或其它集合中逐一取出数据，赋值给指定类型的变量，该变量可以在循环语句中使用、处理，但不允许修改变量，该变量的指定类型必须和表达式所代表的数组或其它集合中的数据类型一致。例子：

```
using System;
class Test
{   public static void Main()
    {   int[] list={10,20,30,40};//数组
        foreach(int m in list)
            Console.WriteLine("{0}",m);
    }
}
```

对于一维数组，foreach语句循环顺序是从下标为0的元素开始一直到数组的最后一个元素。对于多维数组，元素下标的递增是从最右边那一维开始的。同样break和continue可以出现在foreach语句中，功能不变。

1.6.3 异常语句

在编写程序时，不仅要关心程序的正常操作，还应该考虑到程序运行时可能发生的各类不可预期的事件，比如用户输入错误、内存不够、磁盘出错、网络资源不可用、数据库无法使用等，所有这些错误被称作异常，不能因为这些异常使程序运行产生问题。各种程序设计语言经常采用异常处理语句来解决这类异常问题。

C#提供了一种处理系统级错误和应用程序级错误的结构化的、统一的、类型安全的方法。C#异常语句包含try子句、catch子句和finally子句。try子句中包含可能产生异常的语句，该子句自动捕捉执行这些语句过程中发生的异常。catch子句中包含了对不同异常的处理代码，可以包含多个catch子句，每个catch子句中包含了一个异常类型，这个异常类型必须是System.Exception类或它的派生类引用变量，该语句只扑捉该类型的异常。可以有一个通用异常类型的catch子句，该catch子句一般在事先不能确定会发生什么样的异常的情况下使用，也就是可以扑捉任意类型的异常。一个异常语句中只能有一个通用异常类型的catch子句，而且如果有的话，该catch子句必须排在其它catch子句的后面。无论是否产生异常，子句finally一定被执行，在finally子句中可以增加一些必须执行的语句。

异常语句捕捉和处理异常的机理是：当try子句中的代码产生异常时，按照catch子句的顺序查找异常类型。如果找到，执行该catch子句中的异常处理语句。如果没有找到，执行通用异常类型的catch子句中的异常处理语句。由于异常的处理是按照catch子句出现的顺序逐一检查catch子句，因此catch子句出现的顺序是很重要的。无论是否产生异常，一定执行finally子句中的语句。异常语句中不必一定包含所有三个子句，因此异常语句可以有以下三种可能的形式：

- try －catch语句，可以有多个catch语句
- try -finally语句
- try -catch-finally语句，可以有多个catch语句

请看下边的例子：

1. try－catch-finally语句

```
using System
using System.IO//使用文件必须引用的名字空间
public class Example
{   public static void Main()
    {   StreamReader sr=null;//必须赋初值null,否则编译不能通过
        try
        {  sr=File.OpenText("d:\\csarp\\test.txt");//可能产生异常
           string s;
           while(sr.Peek()!=-1)
           {  s=sr.ReadLine();//可能产生异常
              Console.WriteLine(s);
           }
        }
        catch(DirectoryNotFoundException e)//无指定目录异常
        {   Console.WriteLine(e.Message);
        }
        catch(FileNotFoundException e)//无指定文件异常
```

```
        {   Console.WriteLine("文件"+e.FileName+"未被发现");
        }
        catch(Exception e)//其它所有异常
        {  Console.WriteLine("处理失败：{0}",e.Message);
        }
        finally
        {   if(sr!=null)
               sr.Close();
        }
    }
}
```

2. try -finally语句

上例中，其实可以不用catch语句，在finally子句中把文件关闭，提示用户是否正确打开了文件，请读者自己完成。

3. try -catch语句

请读者把上例修改为使用try-catch结构，注意在每个catch语句中都要关闭文件。

1.7 类的继承

在1.3节，定义了一个描述个人情况的类Person，如果我们需要定义一个雇员类，当然可以从头开始定义雇员类Employee。但这样不能利用Person类中已定义的函数和数据。比较好的方法是，以Person类为基类，派生出一个雇员类Employee，雇员类Employee继承了Person类的数据成员和函数成员，既Person类的数据成员和函数成员成为Employee类的成员。这个Employee类叫以Person类为基类的派生类，这是C#给我们提出的方法。C#用继承的方法，实现代码的重用。

1.7.1 派生类的声明格式

派生类的声明格式如下：

```
属性 类修饰符 class 派生类名:基类名 {类体}
```

雇员类Employee定义如下：

```
class Employee:Person//Person类是基类
{   private string department;//部门，新增数据成员
    private decimal salary;//薪金，新增数据成员
    public Employee(string Name,int Age,string D,decimal S):base(Name,Age)
    {//注意base的第一种用法，根据参数调用指定基类构造函数，注意参数的传递
        department=D;
        salary=S;
    }
    public new void Display()//覆盖基类Display()方法,注意new,不可用override
    {   base.Display();//访问基类被覆盖的方法，base的第二种用法
        Console.WriteLine("部门：{0}  薪金：{1}",department,salary);
    }
```

```
}
```

修改主函数如下：

```
class Class1
{   static void Main(string[] args)
    {   Employee OneEmployee=new Employee("李四",30,"计算机系",2000);
        OneEmployee.Display();
    }
}
```

Employee类继承了基类Person的方法SetName()、SetAge()，数据成员name和age，即认为基类Person的这些成员也是Employee类的成员，但不能继承构造函数和析构函数。添加了新的数据成员department和salary。覆盖了方法Display()。请注意，虽然Employee类继承了基类Person的name和age，但由于它们是基类的私有成员，Employee类中新增或覆盖的方法不能直接修改name和age，只能通过基类原有的公有方法SetName()和SetAge()修改。如果希望在Employee类中能直接修改name和age，必须在基类中修改它们的属性为protected。

1.7.2 base 关键字

base关键字用于从派生类中访问基类成员，它有两种基本用法：

- 在定义派生类的构造函数中，指明要调用的基类构造函数，由于基类可能有多个构造函数，根据base后的参数类型和个数，指明要调用哪一个基类构造函数。参见上节雇员类Employee构造函数定义中的base的第一种用法。
- 在派生类的方法中调用基类中被派生类覆盖的方法。参见上节雇员类Employee的Display()方法定义中的base的第二种用法。

1.7.3 覆盖基类成员

在派生类中，通过声明与基类完全相同新成员，可以覆盖基类的同名成员，完全相同是指函数类型、函数名、参数类型和个数都相同。如上例中的方法Display()。派生类覆盖基类成员不算错误，但会导致编译器发出警告。如果增加new修饰符，表示认可覆盖，编译器不再发出警告。请注意，覆盖基类的同名成员，并不是移走基类成员，只是必须用如下格式访问基类中被派生类覆盖的方法：base.Display()。

1.7.4 C#语言类继承特点

C#语言类继承有如下特点：

- C#语言只允许单继承，即派生类只能有一个基类。
- C#语言继承是可以传递的，如果C从B派生，B从A派生，那么C不但继承B的成员，还要继承A中的成员。
- 派生类可以添加新成员，但不能删除基类中的成员。
- 派生类不能继承基类的构造函数、析构函数和事件。但能继承基类的属性。
- 派生类可以覆盖基类的同名成员，如果在派生类中覆盖了基类同名成员，基类该成员在派生类中就不能被直接访问，只能通过base.基类方法名访问。

- 派生类对象也是其基类的对象，但基类对象却不是其派生类的对象。例如，前边定义的雇员类Employee是Person类的派生类，所有雇员都是人类，但很多人并不是雇员，可能是学生，自由职业者，儿童等。因此C#语言规定，基类的引用变量可以引用其派生类对象，但派生类的引用变量不可以引用其基类对象。

1.8 类的成员

由于C#程序中每个变量或函数都必须属于一个类或结构，不能象C或C++那样建立全局变量，因此所有的变量或函数都是类或结构的成员。类的成员可以分为两大类:类本身所声明的以及从基类中继承来的。

1.8.1 类的成员类型

类的成员包括以下类型：

- 局部变量：在for、switch等语句中和类方法中定义的变量，只在指定范围内有效。
- 字段：即类中的变量或常量，包括静态字段、实例字段、常量和只读字段。
- 方法成员：包括静态方法和实例方法。
- 属性：按属性指定的get方法和Set方法对字段进行读写。属性本质上是方法。
- 事件：代表事件本身，同时联系事件和事件处理函数。
- 索引指示器：允许象使用数组那样访问类中的数据成员。
- 操作符重载：采用重载操作符的方法定义类中特有的操作。
- 构造函数和析构函数。

包含有可执行代码的成员被认为是类中的函数成员，这些函数成员有方法、属性、索引指示器、操作符重载、构造函数和析构函数。

1.8.2 类成员访问修饰符

访问修饰符用于指定类成员的可访问性，C#访问修饰符有 private、protected、public 和 internal4 种。Private 声明私有成员，私有数据成员只能被类内部的函数使用和修改，私有函数成员只能被类内部的函数调用。派生类虽然继承了基类私有成员，但不能直接访问它们，只能通过基类的公有成员访问。protected 声明保护成员，保护数据成员只能被类内部和派生类的函数使用和修改，保护函数成员只能被类内部和派生类的函数调用。public 声明公有成员，类的公用函数成员可以被类的外部程序所调用，类的公用数据成员可以被类的外部程序直接使用。公有函数实际是一个类和外部通讯的接口，外部函数通过调用公有函数，按照预先设定好的方法修改类的私有成员和保护成员。internal 声明内部成员，内部成员只能在同一程序集中的文件中才是可以访问的，一般是同一个应用(Application)或库(Library)。

1.9 类的字段和属性

一般把类或结构中定义的变量和常量叫字段。属性不是字段，本质上是定义修改字段的方法，由于属性和字段的紧密关系，把它们放到一起叙述。

1.9.1 静态字段、实例字段、常量和只读字段

用修饰符 static 声明的字段为静态字段。不管包含该静态字段的类生成多少个对象或根本无对象，该字段都只有一个实例，静态字段不能被撤销。必须采用如下方法引用静态字段：类名.静态字段名。如果类中定义的字段不使用修饰符 static，该字段为实例字段，每创建该类的一个对象，在对象内创建一个该字段实例，创建它的对象被撤销，该字段对象也被撤销，实例字段采用如下方法引用：实例名.实例字段名。用 const 修饰符声明的字段为常量，常量只能在声明中初始化，以后不能再修改。用 readonly 修饰符声明的字段为只读字段，只读字段是特殊的实例字段，它只能在字段声明中或构造函数中重新赋值，在其它任何地方都不能改变只读字段的值。例子：

```
public class Test
{   public const int intMax=int.MaxValue;//常量，必须赋初值
    public int x=0;//实例字段
    public readonly int y=0;//只读字段
    public static int cnt=0;//静态字段
    public Test(int x1,int y1)//构造函数
    {   //intMax=0;//错误，不能修改常量
        x=x1;//在构造函数允许修改实例字段
        y=y1;//在构造函数允许修改只读字段
        cnt++;//每创建一个对象都调用构造函数，用此语句可以记录对象的个数
    }
    public void Modify(int x1,int y1)
    {   //intMax=0;//错误，不能修改常量
        x=x1;
        cnt=y1;
        //y=10;//不允许修改只读字段
    }
}
class Class1
{   static void Main(string[] args)
    {   Test T1=new Test(100,200);
        T1.x=40;//引用实例字段采用方法：实例名.实例字段名
        Test.cnt=0;//引用静态字段采用方法：类名.静态字段名
        int z=T1.y;//引用只读字段
        z=Test.intMax;//引用常量
    }
}
```

1.9.2 属性

C#语言支持组件编程，组件也是类，组件用属性、方法、事件描述。属性不是字段，但必然和类中的某个或某些字段相联系，属性定义了得到和修改相联系的字段的方法。C#中的属性更充分地体现了对象的封装性：不直接操作类的数据内容，而是通过访问器进行访问，

借助于get和set方法对属性的值进行读写。访问属性值的语法形式和访问一个变量基本一样，使访问属性就象访问变量一样方便，符合习惯。

在类的基本概念一节中，定义一个描述个人情况的类Person，其中字段name和age是私有字段，记录姓名和年龄，外部通过公有方法SetName和SetAge修改这两个私有字段。现在用属性来描述姓名和年龄。例子如下：

```
using System;
public class Person
{   private string P_name="张三";//P_name是私有字段
    private int P_age=12;//P_age是私有字段
    public void Display()//类的方法声明，显示姓名和年龄
    {   Console.WriteLine("姓名:{0},年龄：{1}",P_name,P_age);
    }
    public string Name//定义属性Name
    {   get
        {   return  P_name;}
        set
        {   P_name=value;}
    }
    public int Age//定义属性Age
    {   get
        {   return  P_age;}
        set
        {   P_age=value;}
    }
}
public class Test
{   public static void Main()
    {   Person OnePerson= new Person();
        OnePerson.Name="田七";//value="田七"，通过set方法修改变量P_Name
        string s=OnePerson.Name;//通过get方法得到变量P_Name值
        OnePerson.Age=20;//通过定义属性，既保证了姓名和年龄按指定方法修改
        int x=OnePerson.Age;//语法形式和修改、得到一个变量基本一致，符合习惯
        OnePerson.Display();
    }
}
```

在属性的访问声明中，只有 set 访问器表明属性的值只能进行设置而不能读出，只有 get 访问器表明属性的值是只读的不能改写，同时具有 set 访问器和 get 访问器表明属性的值的读写都是允许的。

虽然属性和字段的语法比较类似，但由于属性本质上是方法，因此不能把属性当做变量那样使用，也不能把属性作为引用型参数或输出参数来进行传递。

1.10 类的方法

方法是类中用于执行计算或其它行为的成员。所有方法都必须定义在类或结构中。

1.10.1 方法的声明

方法的声明格式如下:

属性 方法修饰符 返回类型 方法名(形参列表){方法体}

方法修饰符包括new、public、protected、internal、private、static、virtual、sealed、override、abstract和extern。这些修饰符有些已经介绍过，其它修饰符将逐一介绍。返回类型可以是任何合法的C#数据类型，也可以是void，即无返回值。形参列表的格式为：(形参类型 形参1,形参类型 形参2,...),可以有多个形参。不能使用C语言的形参格式。

1.10.2 方法参数的种类

C#语言的方法可以使用如下四种参数(请注意和参数类型的区别)：

- 值参数，不含任何修饰符。
- 引用参数，以ref修饰符声明。
- 输出参数，以out修饰符声明。
- 数组参数，以params修饰符声明。

1. 值参数

当用值参数向方法传递参数时，程序给实参的值做一份拷贝，并且将此拷贝传递给该方法，被调用的方法不会修改实参的值，所以使用值参数时，可以保证实参的值是安全的。如果参数类型是引用类型，例如是类的引用变量，则拷贝中存储的也是对象的引用，所以拷贝和实参引用同一个对象，通过这个拷贝，可以修改实参所引用的对象中的数据成员。

2. 引用参数

有时在方法中，需要修改或得到方法外部的变量值，C语言用向方法传递实参指针来达到目的，C#语言用引用参数。当用引用参数向方法传递实参时，程序将把实参的引用，即实参在内存中的地址传递给方法，方法通过实参的引用，修改或得到方法外部的变量值。引用参数以ref修饰符声明。注意在使用前，实参变量要求必须被设置初始值。

3. 输出参数

为了把方法的运算结果保存到外部变量，因此需要知道外部变量的引用(地址)。输出参数用于向方法传递外部变量引用(地址)，所以输出参数也是引用参数，与引用参数的差别在于调用方法前无需对变量进行初始化。在方法返回后，传递的变量被认为经过了初始化。值参数、引用参数和输出参数的使用见下例：

```
using System;
class g{public int a=0;}//类定义
class Class1
{   public static void F1(ref char i)//引用参数
    {   i='b';}
    public static void F2(char i)//值参数，参数类型为值类型
    {   i='d';}
    public static void F3(out char i)//输出参数
    {   i='e';}
    public static void F4(string s)//值参数，参数类型为字符串
    {   s="xyz";}
    public static void F5(g gg)//值参数，参数类型为引用类型
```

```
    {   gg.a=20;}
    public static void F6(ref string s)//引用参数，参数类型为字符串
    {   s="xyz";}
    static void Main(string[] args)
    {   char a='c';
        string s1="abc";
        F2(a);//值参数，不能修改外部的a
        Console.WriteLine(a);//因a未被修改，显示c
        F1(ref a);//引用参数,函数修改外部的a的值
        Console.WriteLine(a);//a被修改为b，显示b
        Char j;
        F3(out j);//输出参数，结果输出到外部变量j
        Console.WriteLine(j);//显示e
        F4(s1);//值参数，参数类型是字符串，s1为字符串引用变量
        Console.WriteLine(s1);//显示：abc,字符串s1不被修改
        g g1=new g();
        F5(g1);//值参数，但实参是一个类引用类型变量
        Console.WriteLine(g1.a.ToString());//显示:20，修改对象数据
        F6(ref s1);//引用参数，参数类型是字符串，s1为字符串引用变量
        Console.WriteLine(s1);//显示：xyz，字符串s1被修改
    }
}
```

4. 数组参数

数组参数使用params说明，如果形参表中包含了数组参数，那么它必须是参数表中最后一个参数，数组参数只允许是一维数组。比如string[]和string[][]类型都可以作为数组型参数。最后，数组型参数不能再有ref和out修饰符。见下例：

```
using System;
class Class1
{   static void F(params int[] args)//数组参数，有params说明
    {   Console.Write("Array contains {0} elements:",args.Length);
        foreach (int i in args)
            Console.Write(" {0}",i);
        Console.WriteLine();
    }
    static void Main(string[] args)
    {   int[] a = {1,2,3};
        F(a);//实参为数组类引用变量a
        F(10, 20, 30, 40);//等价于F(new int[] {60,70,80,90});
        F(new int[] {60,70,80,90});//实参为数组类引用
        F();//等价于F(new int[] {});
        F(new int[] {});//实参为数组类引用，数组无元素
    }
}
```

程序输出

```
Array contains 3 elements: 1 2 3
Array contains 4 elements: 10 20 30 40
Array contains 4 elements: 60,70,80,90
Array contains 0 elements:
Array contains 0 elements:
```

方法的参数为数组时也可以不使用params，此种方法可以使用一维或多维数组，见下例：

```
using System;
class Class1
{   static void F(int[,] args)//值参数，参数类型为数组类引用变量，无params说明
    {   Console.Write("Array contains {0} elements:",args.Length);
        foreach (int i in args)
            Console.Write(" {0}",i);
        Console.WriteLine();
    }
    static void Main(string[] args)
    {   int[,] a = {{1,2,3},{4,5,6}};
        F(a);//实参为数组类引用变量a
        //F(10, 20, 30, 40);//此格式不能使用
        F(new int[,] {{60,70},{80,90}});//实参为数组类引用
        //F();//此格式不能使用
        //F(new int[,] {});//此格式不能使用
    }
}
```

程序输出

```
Array contains 3 elements: 1 2 3 4 5 6
Array contains 4 elements: 60,70,80,90
```

1.10.3 静态方法和实例方法

用修饰符 static 声明的方法为静态方法，不用修饰符 static 声明的方法为实例方法。不管类生成或未生成对象，类的静态方法都可以被使用，使用格式为：类名.静态方法名。静态方法只能使用该静态方法所在类的静态数据成员和静态方法。这是因为使用静态方法时，该静态方法所在类可能还没有对象，即使有对象，由于用类名.静态方法名方式调用静态方法，静态方法没有 this 指针来存放对象的地址，无法判定应访问哪个对象的数据成员。在类创建对象后，实例方法才能被使用，使用格式为：对象名.实例方法名。实例方法可以使用该方法所在类的所有静态成员和实例成员。例子如下：

```
using System;
public class UseMethod
{   private static int x=0;//静态字段
    private int y=1;//实例字段
    public static void StaticMethod()//静态方法
    {   x=10;//正确，静态方法访问静态数据成员
        //y=20;//错误，静态方法不能访问实例数据成员
    }
```

```
    public void NoStaticMethod()//实例方法
    {   x=10;//正确，实例方法访问静态数据成员
        y=20;//正确，实例方法访问实例数据成员
    }
}
public class Class1
{   public static void Main()
    {   UseMethod m=new UseMethod();
        UseMethod.StaticMethod();//使用静态方法格式为：类名.静态方法名
        m.NoStaticMethod();//使用实例方法格式为：对象名.实例方法名
    }
}
```

1.10.4 方法的重载

在 C#语言中，如果在同一个类中定义的函数名相同，而参数类型或参数个数不同，认为是不相同的函数，仅返回值不同，不能看作不同函数，这叫做函数的重载。前边 Person 类中定义了多个构造函数就是重载的例子。在 C 语言中，若计算一个数据的绝对值，则需要对不同数据类型求绝对值方法使用不同的方法名，如用 abc()求整型数绝对值，labs()求长整型数绝对值，fabs()求浮点数绝对值。而在 C#语言中，可以使用函数重载特性，对这三个函数定义同样的函数名，但使用不同的参数类型。下面是实现方法：

```
using System;
public class UseAbs
{   public int  abs(int x)//整型数求绝对值
    {   return(x<0 ? -x:x);}
    public long abs(long x)//长整型数求绝对值
    {return(x<0 ? -x:x);}
    public double  abs(double  x)//浮点数求绝对值
    {return(x<0 ? -x:x);}
}
class Class1
{   static void Main(string[] args)
    {   UseAbs m=new UseAbs();
        int x=-10;
        long y=-123;
        double z=-23.98d;
        x=m.abs(x);
        y=m.abs(y);
        z=m.abs(z);
        Console.WriteLine("x={0},y={1},z={2}",x,y,z);
   }
}
```

类的对象调用这些同名方法，在编译时，根据调用方法的实参类型决定调用那个同名方法，计算不同类型数据的绝对值。这给编程提供了极大方便。

1.10.5 操作符重载

操作符重载是将 C#语言中的已有操作符赋予新的功能，但与该操作符的本来含义不冲突，使用时只需根据操作符出现的位置来判别其具体执行哪一种运算。操作符重载，实际是定义了一个操作符函数，操作符函数声明的格式如下：

static public 函数返回类型 operator 重新定义的操作符(形参表)

C#语言中有一些操作符是可以重载的，例如：+ - ! ~ ++ -- true false * / % & | ^ << >> == != > < >= <=等等。但也有一些操作符是不允许进行重载的，例如：=, &&, ||, ?:, new, typeof, sizeof, is等。

下边的例子，定义一个复数类，并且希望复数的加减乘除用符号+,-.*,/来表示。

```
using System;
class Complex//复数类定义
{   private double Real;//复数实部
    private double Imag;//复数虚部
    public Complex(double x,double y)//构造函数
    {   Real=x;
        Imag=y;
    }
    static public Complex operator - (Complex a)//重载一元操作符负号,注意1个参数
    {   return (new Complex(-a.Real,-a.Imag));}
    static public Complex operator +(Complex a,Complex b)//重载二元操作符加号
    {   return (new Complex(a.Real+b.Real,a.Imag+b.Imag));}
    public void Display()
    {   Console.WriteLine("{0}+({1})j",Real,Imag);}
}
class Class1
{   static void Main(string[] args)
    {   Complex x=new Complex(1.0,2.0);
        Complex y=new Complex(3.0,4.0);
        Complex z=new Complex(5.0,7.0);
        x.Display();//显示:1+(2)j
        y.Display();//显示:3+(4)j
        z.Display();//显示:5+(7)j
        z=-x;//等价于z=opeator-(x)
        z.Display();//显示：-1+(-2)j
        z=x+y;//即z=opeator+(x,y)
        z.Display();//显示：4+(6)j
    }
}
```

1.10.6 this 关键字

每个类都可以有多个对象，例如定义 Person 类的两个对象：

```
    Person P1=new Person("李四",30);
    Person P2=new Person("张三",40);
```

因此 P1.Display()应显示李四信息，P2.Display()应显示张三信息，但无论创建多少个对象，只有一个方法 Display()，该方法是如何知道显示那个对象的信息的呢？C#语言用引用变量 this 记录调用方法 Display()的对象，当某个对象调用方法 Display()时，this 便引用该对象(记录该对象的地址)。因此，不同的对象调用同一方法时，方法便根据 this 所引用的不同对象来确定应该引用哪一个对象的数据成员。this 是类中隐含的引用变量，它是被自动被赋值的，可以使用但不能被修改。例如：P1.Display()，this 引用对象 P1，显示李四信息。P2.Display()，this 引用对象 P2，显示张三信息。

1.11 类的多态性

在面向对象的系统中，多态性是一个非常重要的概念。C#支持两种类型的多态性,第一种是编译时的多态性，一个类的对象调用若干同名方法，系统在编译时，根据调用方法的实参类型及实参的个数决定调用那个同名方法，实现何种操作。编译时的多态性是通过方法重载来实现的。C#语言的方法重载以及操作符重载和C++语言的基本一致。

第二种是运行时的多态性，是在系统运行时，不同对象调用一个名字相同，参数的类型及个数完全一样的方法，会完成不同的操作。C#运行时的多态性通过虚方法实现。在类的方法声明前加上了virtual修饰符，被称之为虚方法，反之为非虚方法。C#语言的虚方法和C++语言的基本一致。下面的例子说明了虚方法与非虚方法的区别：

```
using System;
class A
{   public void F()//非虚方法
    {   Console.Write("  A.F");}
    public virtual void G()//虚方法
    { Console.Write("  A.G");}
}
class B:A//A类为B类的基类
{   new public void F()//覆盖基类的同名非虚方法F()，注意使用new
    {   Console.Write("  B.F");}
    public override void G()//覆盖基类的同名虚方法G()，注意使用override
    {   Console.Write("  B.G");}
}
class Test
{   static void F2(A aA)//注意，参数为A类引用变量
    {   aA.G();}
    static void Main()
    {   B b=new B();
        A a1=new A();
        A a2=b;//允许基类引用变量引用派生类对象，a2引用派生类B对象b
        a1.F();//调用基类A的非虚方法F(),显示A.F
        a2.F();//F()为非虚方法，调用基类A的F(),显示A.F
        b.F();//F()为非虚方法，调用派生类B的F(),显示B.F
        a1.G();//G()为虚方法，因a1引用基类A对象，调用基类A的G()，显示A.G
```

```
        a2.G();//G()为虚方法，因a2引用派生类B对象，调用派生类B的G()，显示B.G
        F2(b);//实参为派生类B对象,由于A aA=b，调用派生类B的函数G(),显示B.G
        F2(a1);//实参为基类A对象,调用A类的函数G(),显示A.G
    }
}
```

那么输出应该是：

A.F A.F B.F A.G B.G B.G A.G

注意例子中，不同对象调用同名非虚方法F()和同名虚方法G()的区别。a2虽然是基类引用变量，但它引用派生类对象b。由于G()是虚方法，因此a2.G()调用派生类B的G()，显示G.F。但由于F()是非虚方法，a2.F()仍然调用基类A的F()，显示A.F。或者说，如果将基类引用变量引用不同对象，或者是基类对象，或者是派生类对象，用这个基类引用变量分别调用同名虚方法，根据对象不同，会完成不同的操作。而非虚方法则不具备此功能。

方法F2(A aA)中，参数是A类类型，F2(b)中形参和实参的关系是：A aA=b，即基类引用变量aA引用派生类对象b，aA.G()调用派生类B的函数G(),显示B.G。同理，F2(a1)实参为基类A对象,调用A类的函数G(),显示A.G。

在类的基本概念一节中，定义一个描述个人情况的类Person，其中公有方法Display()用来显示个人信息。在派生雇员类Employee中，覆盖了基类的公有方法Display()，以显示雇员新增加的信息。我们希望隐藏这些细节，希望无论基类还是派生类，都调用同一个显示方法，根据对象不同，自动显示不同的信息。可以用虚方法来实现，这是一个典型的多态性例子。例子

```
using System;
public class Person
{   private String name="张三";//类的数据成员声明
    private int age=12;
    protected virtual void Display()//类的虚方法
    {   Console.WriteLine("姓名:{0},年龄：{1}",name,age);
    }
    public Person(string Name,int Age)//构造函数,函数名和类同名,无返回值
    {   name=Name;
        age=Age;
    }
    static public void DisplayData(Person aPerson)//静态方法
    {   aPerson.Display();//不是静态方法调用实例方法，如写为Display()错误
    }
}
public class Employee:Person//Person类是基类
{   private string department;
    private decimal salary;
    public Employee(string Name,int Age,string D,decimal S):base(Name,Age)
    {   department=D;
        salary=S;
    }
    protected override void Display()//重载虚方法，注意用override
    {   base.Display();//访问基类同名方法
```

```
        Console.WriteLine("部门：{0}  薪金：{1} ", department,salary);
    }
}
class Class1
{   static void Main(string[] args)
    {   Person OnePerson=new Person("李四",30);
        Person.DisplayData(OnePerson);//显示基类数据
        Employee OneEmployee=new Employee("王五",40,"财务部",2000);
        Person.DisplayData(OneEmployee); //显示派生类数据
    }
}
```

运行后，显示的效果是：

姓名：李四,年龄：30

姓名：王五,年龄：40

部门：财务部 薪金：2000

1.12 抽象类和抽象方法

抽象类表示一种抽象的概念，只是希望以它为基类的派生类有共同的函数成员和数据成员。抽象类使用abstract修饰符，对抽象类的使用有以下几点规定：

- 抽象类只能作为其它类的基类，它不能直接被实例化。
- 抽象类允许包含抽象成员，虽然这不是必须的。抽象成员用abstract修饰符修饰。
- 抽象类不能同时又是密封的。
- 抽象类的基类也可以是抽象类。如果一个非抽象类的基类是抽象类，则该类必须通过覆盖来实现所有继承而来的抽象方法，包括其抽象基类中的抽象方法，如果该抽象基类从其它抽象类派生，还应包括其它抽象类中的所有抽象方法。

请看下面的示例：

```
abstract class Figure//抽象类定义
{   protected double x=0,y=0;
    public Figure(double a,double b)
    {   x=a;
        y=b;
    }
    public abstract void Area();//抽象方法，无实现代码
}
class Square:Figure///类Square定义
{   public Square(double a,double b):base(a,b)
    {}
    public override void Area()//不能使用new，必须用override
    {   Console.WriteLine("矩形面积是：{0}",x*y);}
}
class Circle:Figure///类Square定义
{   public Circle(double a):base(a,a)
    {}
```

```
    public override void Area()
    {   Console.WriteLine("园面积是：{0}",3.14*x*y);}
}
class Class1
{   static void Main(string[] args)
    {   Square s=new Square(20,30);
        Circle c=new Circle(10);
        s.Area();
        c.Area();
    }
}
```

程序输出结果为：
矩形面积是：600
园面积是：314

抽象类Figure提供了一个抽象方法Area()，并没有实现它，类Square和Circle从抽象类Figure中继承方法Area()，分别具体实现计算矩形和园的面积。

在类的基本概念一节中，定义一个描述个人情况的类Person，它只是描述了一个人最一般的属性和行为，因此不希望生成它的对象，可以定义它为抽象类。

注意：C++程序员在这里最容易犯错误。C++中没有对抽象类进行直接声明的方法，而认为只要在类中定义了纯虚函数，这个类就是一个抽象类。纯虚函数的概念比较晦涩，直观上不容易为人们接受和掌握，因此C#抛弃了这一概念。

1.13 密封类和密封方法

有时候，我们并不希望自己编写的类被继承。或者有的类已经没有再被继承的必要。C#提出了一个密封类(sealed class)的概念,帮助开发人员来解决这一问题。

密封类在声明中使用sealed修饰符，这样就可以防止该类被其它类继承。如果试图将一个密封类作为其它类的基类，C#编译器将提示出错。理所当然，密封类不能同时又是抽象类，因为抽象总是希望被继承的。

C#还提出了密封方法(sealed method)的概念。方法使用sealed修饰符，称该方法是一个密封方法。在派生类中，不能覆盖基类中的密封方法。

1.14 接口

与类一样，在接口中可以定义一个和多个方法、属性、索引指示器和事件。但与类不同的是，接口中仅仅是它们的声明，并不提供实现。因此接口是函数成员声明的集合。如果类或结构从一个接口派生，则这个类或结构负责实现该接口中所声明的所有成员。一个接口可以从多个接口继承，而一个类或结构可以实现多个接口。由于C#语言不支持多继承，因此，如果某个类需要继承多个类的行为时，只能使用多个接口加以说明。

1.14.1 接口声明

接口声明是一种类型声明，它定义了一种新的接口类型。接口声明格式如下：
属性 接口修饰符 interface 接口名：基接口{接口体}
其中，关键字interface、接口名和接口体时必须的，其它项是可选的。接口修饰符可以是new、public、protected、internal和private。例子：

```
public interface IExample
{//所有接口成员都不能包括实现
    string this[int index] {get;set;}//索引指示器声明
    event EventHandler E;//事件声明
    void F(int value);//方法声明
    string P { get; set;}//属性声明
}
```

声明接口时，需注意以下内容：

- 接口成员只能是方法、属性、索引指示器和事件，不能是常量、域、操作符、构造函数或析构函数，不能包含任何静态成员。
- 接口成员声明不能包含任何修饰符，接口成员默认访问方式是public。

1.14.2 接口的继承

类似于类的继承性，接口也有继承性。派生接口继承了基接口中的函数成员说明。接口允许多继承，一个派生接口可以没有基接口，也可以有多个基接口。在接口声明的冒号后列出被继承的接口名字，多个接口名之间用分号分割。例子如下：

```
using System;
interface IControl
{   void Paint();
}
interface ITextBox:IControl//继承了接口Icontrol的方法Paint()
{   void SetText(string text);
}
interface IListBox:IControl//继承了接口Icontrol的方法Paint()
{   void SetItems(string[] items);
}
interface IComboBox:ITextBox,IListBox
{//可以声明新方法
}
```

上面的例子中，接口ITextBox和IListBox都从接口IControl中继承，也就继承了接口IControl的Paint方法。接口IComboBox从接口ITextBox和IListBox中继承，因此它应该继承了接口ITextBox的SetText方法和IListBox的SetItems方法，还有IControl的Paint方法。

1.14.3 类对接口的实现

前面已经说过，接口定义不包括函数成员的实现部分。继承该接口的类或结构应实现这些函数成员。这里主要讲述通过类来实现接口。类实现接口的本质是，用接口规定类应实现那些函数成员。用类来实现接口时，接口的名称必须包含在类声明中的基类列表中。

在类的基本概念一节中，定义一个描述个人情况的类Person，从类Person可以派生出其它类，例如：工人类、公务员类、医生类等。这些类有一些共有的方法和属性，例如工资属性。一般希望所有派生类访问工资属性时用同样变量名。该属性定义在类Person中不合适，因为有些人无工资，如小孩。如定义一个类作为基类，包含工资属性，但C#不支持多继承。可行的办法是使用接口，在接口中声明工资属性。工人类、公务员类、医生类等都必须实现该接口，也就保证了它们访问工资属性时用同样变量名。例子如下：

```
using System;
public interface I_Salary//接口
{   decimal Salary//属性声明
    {   get;
        set;
    }
}
public class Person
{…//见1.9.2属性节Person类定义，这里不重复了。
}
public class Employee:Person,I_Salary//Person类是基类,I_Salary是接口
{//不同程序员完成工人类、医生类等，定义工资变量名称可能不同
    private decimal salary;
    public new void Display()
    {   base.Display();
        Console.WriteLine("薪金：{0} ",salary);
    }
    //工人类、医生类等都要实现属性Salary，保证使用的工资属性同名
    public decimal Salary
    {   get
        {   return  salary;}
        set
        {   salary=value;}
    }
}
public class Test
{   public static void Main()
    { Employee S=new Employee();
      S.Name="田七";//修改属性Name
      S.Age=20;//修改属性Age
      S.Salary=2000;//修改属性Salary
      S.Display();
```

```
    }
}
```

如果类实现了某个接口，类也隐式地继承了该接口的所有基接口，不管这些基接口有没有在类声明的基类表中列出。因此，如果类从一个接口派生，则这个类负责实现该接口及该接口的所有基接口中所声明的所有成员。

1.15 代表

在这里要介绍的是C#的一个引用类型----代表(delegate)，也翻译为委托。它实际上相当于C语言的函数指针。与指针不同的是C#中的代表是类型安全的。代表类声明格式如下：

属性集 修饰符 delegate 函数返回类型 定义的代表标识符(函数形参列表);

修饰符包括new、public、protected、internal和private。例如我们可以声明一个返回类型为int，无参数的函数的代表MyDelegate：

```
public delegate int MyDelegate();//只能代表返回类型为int，无参数的函数
```

声明了代表类MyDelegate，可以创建代表类MyDelegate的对象，用这个对象去代表一个静态方法或非静态的方法，所代表的方法必须为int类型，无参数。看下面的例子：

```
using System;
delegate int MyDelegate();//声明一个代表，注意声明的位置
public class MyClass
{   public int InstanceMethod()//非静态的方法，注意方法为int类型，无参数
    {   Console.WriteLine("调用了非静态的方法。");
        return 0;
    }
    static public int StaticMethod()//静态方法，注意方法为int类型，无参数
    {   Console.WriteLine("调用了静态的方法。");
        return 0;
    }
}
public class Test
{   static public void Main ()
    {   MyClass p = new MyClass();
//用new建立代表类MyDelegate对象，d中存储非静态的方法InstanceMethod的地址
        MyDelegate d=new MyDelegate(p.InstanceMethod);//参数是被代表的方法
        d();//调用非静态方法
//用new建立代表类MyDelegate对象，d中存储静态的方法StaticMethod的地址
        d=new MyDelegate(MyClass.StaticMethod);//参数是被代表的方法
        d();//调用静态方法
    }
}
```

程序的输出结果是：

```
调用了非静态的方法。
调用了静态的方法。
```

1.16 事件

事件是 C#语言内置的语法，可以定义和处理事件，为使用组件编程提供了良好的基础。

1.16.1 事件驱动

Windows 操作系统把用户的动作都看作消息，C#中称作事件，例如用鼠标左键单击按钮，发出鼠标单击按钮事件。Windows 操作系统负责统一管理所有的事件，把事件发送到各个运行程序。各个程序用事件函数响应事件，这种方法也叫事件驱动。

C#语言使用组件编制 Windows 应用程序。组件本质上是类。在组件类中，预先定义了该组件能够响应的事件，以及对应的事件函数，该事件发生，将自动调用自己的事件函数。例如，按钮类中定义了单击事件 Click 和单击事件函数。一个组件中定义了多个事件，应用程序中不必也没必要响应所有的事件，而只需响应其中很少事件，程序员编制相应的事件处理函数，用来完成需要响应的事件所应完成的功能。现在的问题是，第一，如何把程序员编制的事件处理函数和组件类中预先定义的事件函数联系起来。第二，如何使不需响应的事件无动作。这是本节要节的解决问题。

1.16.2 事件的声明

在C#中，事件首先代表事件本身，例如按钮类的单击事件，同时，事件还是代表类引用变量，可以代表程序员编制的事件处理函数，把事件和事件处理函数联系在一起。下面的例子定义了一个Button组件，这个例子不完整，只是说明问题。实际在C#语言类库中已预定义了Button组件，这里的代码只是想说明Button组件中是如何定义事件的。例子如下：

```
public delegate void EventHandler(object sender,EventArgs e);//代表声明
//EventHandler可以代表没有返回值，参数为(object sender,EventArgs e)的函数
public class Button:Control//定义一个按钮类Button组件
{…//按钮类Button其它成员定义
    public event EventHandler Click;//声明一个事件Click，是代表类引用变量
    protected void OnClick(EventArgs e)//Click事件发生，自动触发OnClick方法
    {   if(Click!=null)//如果Click已代表了事件处理函数，执行这个函数
            Click(this,e);
    }
    public void Reset()
    {   Click=null;}
}
```

在这个例子中，Click事件发生，应有代码保证(未列出)自动触发OnClick方法。Click是类Button的一个事件，同时也是代表EventHandler类的引用变量，如令Click代表事件处理函数，该函数完成Click事件应完成的功能，Click事件发生时，执行事件处理函数。

1.16.3 事件的预订和撤消

在随后的例子中，我们声明了一个使用Button类的登录对话框类，对话框类含有两个按钮：OK和Cancel按钮。

```
public class LoginDialog: Form//登录对话框类声明
{   Button OkButton;
    Button CancelButton;
    public LoginDialog()//构造函数
    {   OkButton=new Button();//建立按钮对象OkButton
        //Click代表OkButtonClick方法，注意+=的使用
        OkButton.Click+=new EventHandler(OkButtonClick);
        CancelButton=new Button();//建立按钮对象OkButton
        CancelButton.Click += new EventHandler(CancelButtonClick);
    }
    void OkButtonClick(object sender, EventArgs e)
    {…//处理OkButton.Click事件的方法
    }
    void CancelButtonClick(object sender, EventArgs e)
    {…//处理CancelButton.Click事件的方法
    }
}
```

在例子中建立了Button类的两个实例，单击按钮事件Click通过如下语句和事件处理方法联系在一起：OkButton.Click+=new EventHandler(OkButtonClick)，该语句的意义是使OkButton.Click代表事件处理方法OkButtonClick，这样只要Click事件被触发，事件处理方法OkButtonClick就会被自动调用。撤消事件和事件处理方法OkButtonClick的联系采用如下语句实现：OkButton.Click-=new EventHandler(OkButtonClick)，这时，OkButton.Click就不再代表事件处理方法，Click事件被触发，方法OkButtonClick就不会被调用了。务必理解这两条语句的用法。使用Visual Studio.Net集成环境可以自动建立这种联系，在自动生成的代码中包括这两条语句。

1.17 索引指示器

在C#语言中，数组也是类，比如我们声明一个整型数数组：int[] arr=new int[5]，实际上生成了一个数组类对象，arr是这个对象的引用(地址)，访问这个数组元素的方法是：arr[下标]，在数组类中，使用索引访问元素是如何实现的呢？是否可以定义自己的类，用索引访问类中的数据成员？索引指示器(indexer)为我们提供了通过索引方式方便地访问类的数据成员的方法。

首先看下面的例子，用于打印出小组人员的名单：

```
using System
class Team
{   string[] s_name = new string[2];//定义字符串数组，记录小组人员姓名
    public string this[int nIndex]//索引指示器声明，this为类Team类的对象
    {   get//用对象名[索引]得到记录小组人员姓名时，调用get函数
```

```
        {   return s_name[nIndex];
        }
        set//用对象名[索引]修改记录小组人员姓名时，调用set函数
        {   s_name[nIndex] =value;//value为被修改值
        }
    }
}
class Test
{   public static void Main()
    {   Team t1 = new Team();
        t1[0]="张三";
        t1[1]="李斯";
        Console.WriteLine("{0}, {1}",t1[0], t1[1]);
    }
}
```

显示结果如下：张三，李斯

1.18 名字空间

一个应用程序可能包含许多不同的部分，除了自己编制的程序之外，还要使用操作系统或开发环境提供的函数库、类库或组件库，软件开发商处购买的函数库、类库或组件库，开发团队中其它人编制的程序，等等。为了组织这些程序代码，使应用程序可以方便地使用这些程序代码，C#语言提出了名字空间的概念。名字空间是函数、类或组件的容器，把它们按类别放入不同的名字空间中，名字空间提供了一个逻辑上的层次结构体系，使应用程序能方便的找到所需代码。这和C语言中的include语句的功能有些相似，但实现方法完全不同。

1.18.1 名字空间的声明

用关键字namespace声明一个名字空间，名字空间的声明要么是源文件using语句后的第一条语句，要么作为成员出现在其它名字空间的声明之中，也就是说，在一个名字空间内部还可以定义名字空间成员。全局名字空间应是源文件using语句后的第一条语句。在同一名字空间中，不允许出现同名名字空间成员或同名的类。在声明时不允许使用任何访问修饰符，名字空间隐式地使用public修饰符。例子如下：

```
using System;
namespace N1//N1为全局名字空间的名称，应是using语句后的第一条语句
{   namespace N2//名字空间N1的成员N2
    {   class A//在N2名字空间定义的类不应重名
        {   void f1(){};}
        class B
        {   void f2(){};}
    }
}
```

也可以采用非嵌套的语法来实现以上名字空间:

```
namespace N1.N2//类A、B在名字空间N1.N2中
{   class A
    {   void f1(){};}
    class B
    {   void f2(){};}
}
```

也可以采用如下格式：

```
namespace N1.N2//类A在名字空间N1.N2中
{   class A
    {   void f1(){};}
}
namespace N1.N2//类B在名字空间N1.N2中
{   class B
    {   void f2(){};}
}
```

1.18.2 名字空间使用

如在程序中，需引用其它名字空间的类或函数等，可以使用语句using，例如需使用上节定义的方法f1()和f2(),可以采用如下代码：

```
using N1.N2;
class WelcomeApp
{   A a=new A();
    a.f1();
}
```

using N1.N2实际上是告诉应用程序到哪里可以找到类A。请读者重新看一下1.2.1节中的例子。

1.19 非安全代码

在C和C++的程序员看来，指针是最强有力的工具之一，同时又带来许多问题。因为指针指向的数据类型可能并不相同，比如你可以把int类型的指针指向一个float类型的变量，而这时程序并不会出错。如果你删除了一个不应该被删除的指针，比如Windows中指向主程序的指针，程序就有可能崩溃。因此滥用指针给程序带来不安全因素。正因为如此，在C#语言中取消了指针这个概念。虽然不使用指针可以完成绝大部分任务，但有时在程序中还不可避免的使用指针，例如调用Windows操作系统的API函数，其参数可能是指针，所以在C#中还允许使用指针，但必须声明这段程序是非安全(unsafe)的。可以指定一个方法是非安全的，例如：unsafe void F1(int＊p){…}。可以指定一条语句是非安全的，例如：unsafe int* p2=p1;还可以指定一段代码是非安全的，例如：unsafe{ int* p2=p1;int* p3=p4;}。在编译时要采用如下格式：csc 要编译的C#源程序 /unsafe。

习题

1. 从键盘输入姓名，在显示器中显示对输入姓名的问候。(提示：string 为字符串类型，用语句 string s=Console.ReadLine()输入姓名)
2. 构造函数和析购函数的主要作用是什么？它们各有什么特性？
3. 定义点类，数据成员为私有成员，增加有参数和无参数构造函数，在主函数中生成点类对象，并用字符显示点类对象的坐标。
4. 定义矩形类，数据成员为私有成员，增加有参数和无参数构造函数，在主函数中生成矩形类对象，并用字符显示矩形类对象的长、宽和矩形左上角的坐标。
5. 设计一个计数器类，统计键入回车的次数，数据成员为私有成员，在主程序中使用此类统计键入回车的次数。
6. 说明值类型和引用类型的区别，并和 C 语言相应类型比较。
7. 定义点结构，在主函数中生成点结构变量，从键盘输入点的位置，并重新显示坐标。
8. 定义整型一维数组，从键盘输入数组元素数值后，用循环语句显示所有元素的值。
9. 输入字符串，将字符串第一个字母和每个空格后的字母变为大写，其余字母为小写后输出。
10. 输入 5 个数，在每两个数之间增加 3 个空格后输出。
11. 编一个猜数程序，程序设定一个 1 位十进制数，允许用户猜 3 次，错了告诉比设定数大还是小，用 switch 语句实现。
12. C#语言 for 语句可以这样使用：for(int i;i<10;i++),请问，i 的有效使用范围。
13. 用字符*在 CRT 上显示一个矩形。
14. 输入一个字符串，用 foreach 语句计算输入的字符串长度，并显示长度。
15. 输入两个数相加，并显示和。用异常语句处理输入错误。
16. 将 1.6.3 节中 try－catch-finally 语句例子改为 try-finally 和 try－catch 语句。
17. 定义点类，从点类派生矩形类，数据成员为私有成员，增加有参数和无参数构造函数，在主函数中生成矩形类对象，并用字符显示矩形类对象的长、宽和矩形左上角的坐标。
18. 重做 12 题，将数据成员用属性表示。
19. 定义一个类，将类外部的 char 数组元素都变为大写。主程序输入一个字符串，将其变为 char 数组，变为大写后输出每一个 char 数组元素。分别用类对象和静态函数实现。
20. 定义分数类，实现用符号+，-，*，/完成分数的加减乘除。在主函数中输入两个数，完成运算后输出运算结果。
21. 建立一个 sroot()函数，返回其参数的二次根。重载它，让它能够分别返回整数、长整数和双精度参数的二次根。
22. 重新设计 complex 类，完成复数的+、-、*、/四则运算。
23. 定义点类，从点类派生矩形类和园类，主程序实现用同一个方法显示矩形和园的面积。
24. 重做 19 题，将点类定义为抽象类。
25. 重做 19 题，改为接口实现，即将点类改为接口。

第二章 Windows 编程的基础知识

2.1 窗口

Windows 应用程序一般都有一个窗口，窗口是运行程序与外界交换信息的界面。一个典型的窗口包括标题栏，最小化按钮，最大/还原按钮，关闭按钮，系统菜单图标，菜单，工具条，状态栏，滚动条，客户区等。程序员的工作之一是设计符合自己要求的窗口，C#用控件的方法设计界面。编程另一个工作是在用户区显示数据和图形。

2.2 Windows 的消息系统

2.2.1 消息驱动（事件驱动）

Windows 应用程序和 dos 程序(控制台程序)的最大区别是事件驱动,也叫消息驱动。dos 程序运行时如要读键盘，则要独占键盘等待用户输入，如用户不输入，则 CPU 一直执行键盘输入程序，等待用户输入，即 dos 程序独占外设和 CPU。

Windows 操作系统是一个多任务的操作系统，允许同时运行多个程序，它不允许任何一个程序独占外设，如键盘，鼠标等，所有运行程序共享外设和 CPU，各个运行程序都要随时从外设接受命令，执行命令。

因此必须由 Windows 操作系统统一管理各种外设。Windows 把用户对外设的动作都看作事件(消息)，如单击鼠标左键，发送单击鼠标左键事件，用户按下键盘，发送键盘被按下的事件等。Windows 操作系统统一负责管理所有的事件，把事件发送到各个运行程序，而各个运行程序用一个函数响应事件，这个函数叫事件响应函数。这种方法叫事件驱动。每个事件都有它自己的事件响应函数，当接到 Windows 事件后，自动执行此事件的事件响应函数。程序员编程的主要工作就是编制这些事件的处理函数，完成相应的工作。

2.2.2 事件队列

Windows 把用户的动作都看作事件，Windows 操作系统负责管理所有的事件，事件发生后，这些事件被放到系统事件队列中，Windows 操作系统从系统事件队列中逐一取出事件，分析各个事件，分送事件到相应运行程序的事件队列中。而每个运行程序，则利用消息循环方法（既循环取得自己事件队列中的事件）得到事件，并把他们送到当前活动窗口，由窗口中的事件函数响应各个事件(消息)。因此，每个运行程序都有自己的事件队列。

2.2.3 注视窗口

Windows 操作系统允许多个程序同时运行，每个程序可能拥有多个窗口，但其中只有一

个窗口是活动的，我们能从窗口的标题栏的颜色来识别一个活动窗口，这个窗口接收 Windows 系统发来的大部分的事件。这个应用程序的窗口被称为注视(活动)窗口。

2.3 Windows 编程接口和类库

操作系统为了方便应用程序设计，一般都要提供一个程序库，一些设计应用程序的共用代码都包含在这个程序库中。程序员可以调用这些代码，以简化编程。这节介绍一些常用程序库。

2.3.1 Windows 编程接口(API)

API(Application Programming Interface)是 Windows98、2000 和 XP 操作系统中提供的一组函数，这些函数采用 C 语言调用格式，是为程序员编制 Windows 应用程序提供的编程接口。程序员用 C 语言直接调用 API 也可以编制 Windows 应用程序，但大量的程序代码必须由程序员自己编写，而 API 函数非常庞大，给编程者带来很大的困难。

2.3.2 MFC 类库

由于 API 函数十分庞大复杂，看不到函数之间的关系，使程序员不易使用。用 C 语言使用 API 函数编写 Windows 应用程序是十分困难的。微软的 VC++6.0 用类对 API 函数进行了封装，为编程提供了 MFC 类库。使用 MFC 类库简化了 Windows 应用程序的编制。但是，MFC 类库的使用还是比较复杂的，因此，VC++一直是一些专业人员的编程工具。

2.3.3 组件库

为了简化 Windows 应用程序的设计，提出了组件(控件)的概念，组件也是类，按钮、菜单、工具条等都可以封装为组件，组件采用属性、事件、方法来描述，其中属性描述组件的特性，如按钮的标题，标签字体的颜色和大小。方法是组件类提供的函数，通过调用这些方法，可以控制组件的行为。组件通过事件和外界联系，一个组件可以响应若干个事件，可以为事件增加事件处理函数，以后每当发生该事件，将自动调用该事件处理函数处理此事件。很多组件在设计阶段是可见的，支持可视化编程，这些组件又被叫做控件。用控件编制 Windows 应用程序很象搭积木，将控件放到窗体中，设置好属性，漂亮的界面就设计好了。组件编程的工具有很多，例如：VB6.0、VB.Net、C#、C++Builder、Java、Delphi 等快速开发工具(RAD)。这些工具都有自己的组件库。

2.3.4 .NET 框架类库

.NET 系统为编制 Windows 应用程序、Web 应用程序、Web 服务，在.Net 框架(.Net FrameWork)中提供了基础类库(Base Class Library)。它是一个统一的、面向对象的、层次化的、可扩展的类库，统一了微软当前各种不同的框架和开发模式，无论开发 Windows 应用程序，还是开发 Web 应用程序，采用相同的组件名称，组件具有相同的属性、方法和事件，

开发模式也类似,方便程序员学习。.Net 框架类库支持控件可视化编程,.Net 中的 VC++.Net、VB.Net、C#语言都使用这个类库，消除了各种语言开发模式的差别。该类库包括以下功能：基础类库(基本功能，象字符串、数组等)、网络、安全、远程化、诊断和调试、I/O、数据库、XML、Web 服务、Web 编程、Windows 编程接口等等。

Windows98、2000 和 XP 操作系统并不包含.NET 框架类库，为了运行 C#程序，必须安装.Net FrameWork。

2.4 Windows 应用程序的基本结构

Windows 应用程序和控制台应用程序的基本结构基本一样，程序的执行总是从 Main()方法开始，主函数 Main()必须在一个类中。但 Windows 应用程序使用图形界面，一般有一个窗口(Form)，采用事件驱动方式工作。本节介绍 Windows 应用程序的基本结构。

2.4.1 最简单的 Windows 应用程序

最简单的 Windows 应用程序如下：

```
using System;//引入名字空间
using System.Windows.Forms;
public class Form1:Form//类定义
{   static void Main()//主函数
    {   Application.Run(new Form1());
    }
}
```

自定义类Form1以Form类为基类。Form类是.Net系统中定义的窗体类，Form类对象具有Windows应用程序窗口的最基本功能，有标题栏、系统菜单、最大化按钮、最小化按钮和关闭按钮、用户区。Form类对象还是一个容器，在Form窗体中可以放置其它控件，例如菜单控件，工具条控件等等。System.Application类中的静态方法Run负责完成一个应用程序的初始化，运行，终止等功能，其参数是本程序使用的窗体Form1类对象，Run方法还负责从操作系统接受事件，并把事件送到窗体中响应。窗体关闭，方法Run退出，Windows应用程序结束。假设已经将文件保存在d:\Charp目录下，文件名为：e1.cs。启动命令行提示符，在屏幕上输入一行命令：d:回车，cd Charp回车，键入命令：

```
C:\WINNT\Microsoft.NET\Framework\v1.0.3705\csc /t:winexe
/r:system.dll,System.Windows.Forms.dll e1.cs
```

命令中的/t:winexe 表示要建立一个 Windows 应用程序，/r 表示要引入的命名空间。也可以用记事本建立一个批处理文件 g.bat，将以上命令内容拷贝到文件中，运行 g.bat，和在命令行提示符键入命令效果相同。以上方法在 FrameWork SDK 2000 中实现。如果一切正常 e1.cs 文件将被编译，编译后生成可执行文件 e1.exe。运行可执行文件 e1.exe，CRT 上出现一个窗口如右图。

可以在 Form1 类中定义新的变量，由于主窗体关闭，程序也就结束了，因此定义在主窗体 Form1 中的变量的生命周期和程序的生命周期是相同的，从这个意义上说，这些变量是全局变量。可以为 Form1 类定义构造函数，在构造函数中做一些初始化的工作，例如修改 Form1 标题栏中的标题。还可以在 Form1 中定义控件类的对象，这些控件将在 Form1 的

用户区显示出来，换句话讲，在 Form1 中生成控件对象，也就是把控件放到窗体中。如在窗体中增加了一个按钮(Button)控件，单击按钮，将产生单击按钮事件，完成一定功能，下例说明了如何在窗体中增加控件，如何修改控件属性，如何增加控键的事件处理函数。

```
using System;
using System.Windows.Forms;
public class Form1:Form
{   Button button1;//生成Button类引用变量，和应用程序有相同生命周期
    public Form1()//构造函数
    {//下句修改主窗体标题，不指明属性(方法)所属对象，默认为 Form1 类的属性(方法)
        Text="我的第一个程序";//也可写为：this.Text="我的第一个程序";
        button1=new Button();//生成 Button 类对象
        button1.Location=new Point(25,25);//修改 button1 属性 location 即按钮位置
        button1.Text="确定";//修改 button1 属性 Text，即按钮的标题
        //下句指定 button1_Click 函数是按钮单击事件的单击事件处理函数
        button1.Click+=new System.EventHandler(button1_Click);
        this.Controls.Add(button1);//按钮增加到窗体中，将在主窗体用户区显示出来
    }
    static void Main()
    {   Application.Run(new Form1());
    }
    private void button1_Click(object sender, System.EventArgs e)
    {//事件处理函数
        this.button1.Text="单击了我";//单击按钮事件执行的语句
    }
}
```

请注意在窗体中增加控件类的对象的步骤，首先生成一个引用变量 button1，和主窗体 Form1 有相同的生命周期，第二步在构造函数中用 new 生成 Button 类对象，第三步在构造函数中修改 button1 的属性，增加 button1 的事件函数。这些步骤对于定义任何一个控件都是相同的。编译运行结果如右图：

2.4.2 用 Visual Studio.Net 建立 Windows 应用程序框架

以上所做的工作，都是一些固定的工作，可以使用 Visual Studio.Net 自动建立，下面介绍使用 Visual Studio.Net 创建 Windows 应用程序的具体步骤。

(1) 运行 Visual Studio.Net 程序，出现如图 1.2.2A 界面。

(2) 单击新建项目按钮，出现如图 1.2.2B 对话框。在项目类型(P)编辑框中选择 Visual C#项目，在模板(T)编辑框中选 Windows 应用程序，在名称(N)编辑框中键入 e2，在位置(L)编辑框中键入 D:\csarp。也可以单击浏览按钮，在打开文件对话框中选择文件夹。单击确定按钮，创建项目。出现如图 2.4.2A 界面。生成一个空白窗体(Form1)。

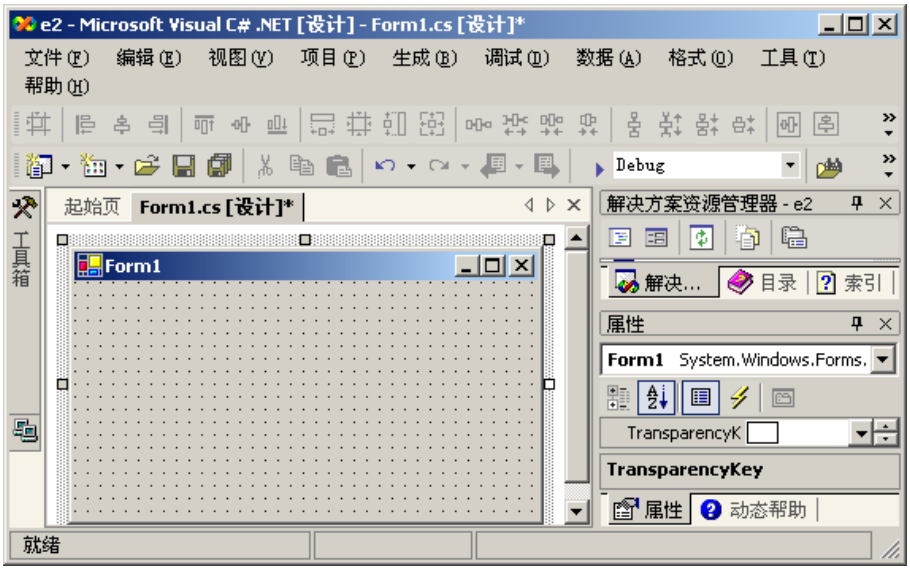


图 2.4.2A

(3) 在 e2 文件夹中下有两个文件夹和 8 个文件，一般只修改 Form1.cs 文件。右击 Form1 窗体，在快捷菜单中选择菜单项查看代码(C)，可打开 Form1.cs 文件。Visual Studio.Net 生成的 Foem1.cs 文件如下，这是使用 Visual Studio.Net 创建 Windows 应用程序的最基本的形式。底色为黑色的字是作者增加的注解。

```
using System;//引入名字空间
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace e2//定义名字空间，///为解释
{   //此处可定义其它类
    /// <summary>
    /// Form1 的摘要说明。
    /// </summary>
    public class Form1 : System.Windows.Forms.Form//Forme1类定义
    {   //此处可定义自己的变量，这些变量和运行程序同生命周期
        /// <summary>
        /// 必需的设计器变量。
        /// </summary>
        private System.ComponentModel.Container components = null;
        public Form1()//构造函数
        {
            //
            // Windows 窗体设计器支持所必需的
            //
```

```
            InitializeComponent();//此函数系统自动生成，不要修改，该函数做一些初始化工作
            //
            // TODO: 在 InitializeComponent 调用后添加任何构造函数代码
            //在构造函数增加自己的初始化代码，必须放在InitializeComponent()之后
        }
        /// <summary>
        /// 清理所有正在使用的资源。
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }
        #region Windows Form Designer generated code
        /// <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()
        {    //此函数系统自动生成，不要修改函数内容，函数做一些初始化工作
            //
            // Form1
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(6, 14);
            this.ClientSize = new System.Drawing.Size(292, 273);
            this.Name = "Form1";//this 是Form1窗体对象
            this.Text = "Form1";
        }
        #endregion
        /// <summary>
        /// 应用程序的主入口点。
        /// </summary>
        [STAThread]
        static void Main()//程序入口函数 ，一般不修改
        {
            Application.Run(new Form1());//主程序建立窗体运行
        }//程序入口函数之后可定义自己的方法、属性等
}
```

}

(4) 下边在窗体中增加一个按钮，并为按钮增加单击事件函数。单击图 2.4.2A 中标题为 Forms.cs[设计]的窗口标签，返回标题为 Forms.cs[设计]的窗口。向项目中添加控件需要使用工具箱窗口，若看不到，可以用菜单命令视图/工具箱打开这个窗口(见图 2.4.2B 左图)。选中工具箱窗口中 Windows 窗体类型下的 Button 条目，然后在标题为 Forms.cs[设计]的窗口的 Form1 窗体中按下鼠标左键，拖动鼠标画出放置 Button 控件的位置，抬起鼠标左键，就将 Button 控件放到 Form1 窗体中。选中按钮控件，属性窗口(见图 2.4.2B 中图)显示按钮属性，其中左侧为属性名称，右侧为属性值，用属性窗口修改 Button 的 Text 属性值为：确定。单击属性窗体上的第 4 个图标，打开事件窗口(见图 2.4.2B 右图)，显示 Button 控件所能响应的所有事件，其中左侧为事件名称，右侧为事件处理函数名称，如果为空白，表示还没有事件处理函数，选中 Click 事件，双击右侧空白处，增加单击事件处理函数。

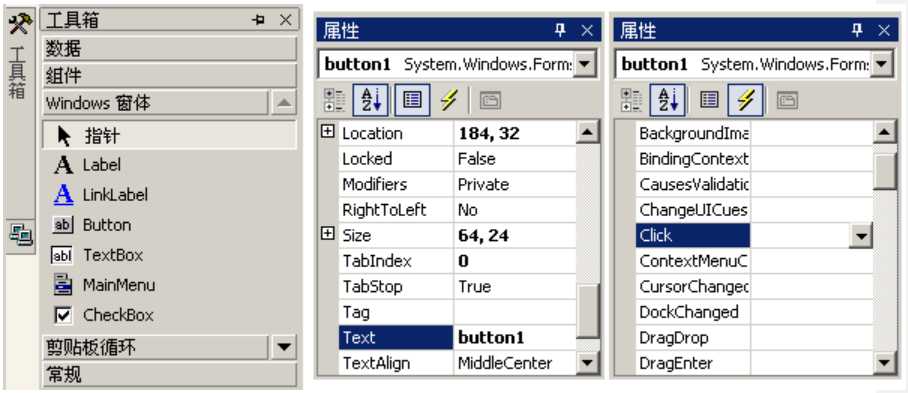


图 2.4.2B

完成以上设计后，集成环境生成的 Foem1.cs 文件如下，底色为黑色的代码是新增代码。

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace e2
{
    /// <summary>
    /// Form1 的摘要说明。
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button button1;//定义Button类引用变量
        /// <summary>
        /// 必需的设计器变量。
        /// </summary>
        private System.ComponentModel.Container components = null;
```

```
public Form1()
{
    //
    // Windows 窗体设计器支持所必需的
    //
    InitializeComponent();
    //
    // TODO: 在 InitializeComponent 调用后添加任何构造函数代码
    //
}
/// <summary>
/// 清理所有正在使用的资源。
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}
#region Windows Form Designer generated code
/// <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.button1 = new System.Windows.Forms.Button();//生成对象
        this.SuspendLayout();
        //
        // button1
        //
        this.button1.Location = new System.Drawing.Point(96, 56);//修改属性
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(72, 32);
        this.button1.TabIndex = 0;
        this.button1.Text = "确定";
        this.button1.Click += new System.EventHandler(this.button1_Click);//增加事件
        //
        // Form1
```

```
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(6, 14);
            this.ClientSize = new System.Drawing.Size(292, 273);
            this.Controls.AddRange(new System.Windows.Forms.Control[] {this.button1});
            this.Name = "Form1";
            this.Text = "Form1";
        }
        #endregion
        /// <summary>
        /// 应用程序的主入口点。
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }
        private void button1_Click(object sender, System.EventArgs e)
        {//事件处理函数
        }
    }
}
```

请注意按钮放到窗体后，集成环境自动增加的语句。分析这些增加的语句，可知在窗体中增加Button类对象的步骤：首先定义Button类变量button1，这是Form1类的一个字段，由于主窗体关闭，程序也就结束了，因此定义在主窗体Form1中的变量的生命周期和程序的生命周期是相同的，从这个意义上说，这样的变量是全局变量。因此变量button1和主窗体Form1有相同的生命周期。第二步在构造函数中用new生成Button类对象，第三步在构造函数中修改button1的属性，第四步增加button1的事件函数，函数button1_Click()是事件处理函数，语句this.button1.Click += new System.EventHandler(this.button1_Click)把按钮Button1的事件Click和事件处理函数button1_Click()联系到一起。程序员应在事件处理函数button1_Click()中增加具体的事件处理语句。这些步骤对于增加任何控件都是相同的。可以比较一下2.4.1节中的步骤，它们基本是相同的。应熟悉以上操作步骤，学会在窗体中增加控件，修改控件属性，增加事件函数。

2.4.3 方案(Solution)和项目(Project)

一个应用(Application)可能包括一个或多个可执行程序，例如，学生信息管理系统，可能包括客户端程序和服务器端程序，所有这些可执行程序的集合叫做一个应用解决方案。为了生成一个可执行程序，可能需要有一个或多个文件，例如，一般需要一个窗体文件，有时还需要一个资源文件，若干图形或图像文件。所有这些文件的集合叫一个项目，因此项目是为了创建一个可执行程序所必需的所有的文件的集合。而一个方案中可能包括多个项目。为了方便管理项目和项目所在的方案，Visual Studio.Net 为开发人员提供了解决方案资源管理器窗口(图 2.4.3)。它可以为我们显示一个方案的树形结构，以及它所包含的项目及项目中的文件。

一个项目一般要放在一个文件夹中，例如上边的例子，项目 e2 的所有文件都在文件夹

e2 中，共有两个文件夹和 8 个文件，它们的用途如下：

- bin 文件夹：包含 debug 子文件夹，存储生成带调试信息的可执行 C#程序。
- obj 文件夹：包含编译过程中生成的中间代码。
- AssemblyInfo.cs：创建项目自动添加。包含各种属性设置，例如，项目最终创建的可执行文件或 DLL 文件中的信息，如标题、描述、公司名等。一般用工具修改该程序，不要直接修改。
- Form1.cs：窗体文件，程序员一般只修改该文件。
- Form1.resx：资源文件。程序员用集成环境提供的工具修改，不要直接修改。
- e2.suo：解决方案用户选项文件，记录用户关于解决方案的选项。
- e2.csproj：项目文件，记录用户关于项目的选项。
- e2.sln：解决方案文件。

图 2.4.3

为了以后重新用 Visual Studio.Net 打开该解决方案，必须保存除了两个文件夹以外的所有文件，实际上，由于文件夹 e2 不太大，可以保存整个 e2 文件夹。如果重新开始一个解决方案，首先用菜单项文件/关闭解决方案，关闭当前项目，再新建一个项目。为了用 Visual Studio.Net 修改以前的程序，必须打开保存的项目文件(扩展名为 csproj)，或者使用菜单项文件/打开项目，打开保存的项目，同时打开项目所在的解决方案。

习题

(1)Windows 应用程序和 dos 程序有那些不同。
(2)以键盘操作为例说明什么是事件驱动。
(3)那些 Windows 操作系统提供了.NET 框架类库，那些提供了 API。
(4)运行 C#程序，应首先安装那些软件。
(5)定义一个和应用程序同生命周期的变量，该变量应定义在何处，说明该变量的使用范围。
(6)在窗体中增加一个控件，应如何操作，集成环境增加了那些代码。
(7)为控件增加事件函数，应如何操作，集成环境增加了那些代码。
(8)如何为窗体文件增加一个方法，说明该方法的使用范围。

第三章 常用控件和类的使用

Visual Studio.Net(简称 VS.NET)使用控件(组件)设计 Windows 应用程序。将 VS.NET 工具箱窗口中的控件放到窗体中，使用属性窗口改变控件的属性，或在程序中用语句修改属性，为控件增加事件函数，完成指定的功能。

3.1 控件通用属性

大部分控件，例如 Label、Button、TextBox 等，都是 Control 类的派生类。Control 类中定义了这些派生类控件通用的一组属性和方法，这些属性是：

- Name：控件的名称，区别控件类不同对象的唯一标志，例如建立一个 Button 控件类对象，可用如下语句，Button button1=new Button()，那么 Name 属性的值为 button1。
- Location：表示控件对象在窗体中的位置。本属性是一个结构，结构中有两个变量，x 和 y，分别代表控件对象左上角顶点的 x 和 y 坐标，该坐标系以窗体左上角为原点，x 轴向左为正方向，y 轴向下为正方向，以像素为单位。修改 Location，可以移动控件的位置，例如：button1.Location=new Point(100,200)语句移动按钮 button1 到新位置。
- Left 和 Top：属性值等效于控件的 Location 属性的 X 和 Y。修改 Left 和 Top，可以移动控件的位置，例如：button1.Left=100 语句水平移动按钮 button1。
- Size：本属性是一个结构，结构中有两个变量，Width 和 Height 分别代表控件对象的宽和高，例如可用语句 button1.Size.Width=100 修改 Button 控件对象 button1 的宽。
- BackColor：控件背景颜色。
- Enabled：布尔变量，为 true 表示控件可以使用，为 false 表示不可用，控件变为灰色。
- Visible：布尔变量，为 true 控件正常显示，为 false 控件不可见。
- Modifier：定义控件的访问权限，可以是 private、public、protected 等。默认值为 private。
- Cursor：鼠标移到控件上方时，鼠标显示的形状。默认值为 Default，表示使用默认鼠标形状，即为箭头形状。

3.2 Form 类

Form 类是.Net 系统中定义的窗体类(WinForm)，它属于 System.Windows.Forms 名字空间。Form 类对象具有 Windows 应用程序窗口的最基本功能。它可以是对话框、单文档或多文档应用程序窗口的基类。Form 类对象还是一个容器，在 Form 窗体中可以放置其它控件，例如菜单控件，工具条控件等等，还可以放置子窗体。

1. Form 类常用属性

- AutoScroll：布尔变量，表示窗口是否在需要时自动添加滚动条。
- FormBorderStyle：窗体边界的风格，如有无边界、单线、3D、是否可调整等。
- Text：字符串类对象，窗体标题栏中显示的标题。
- AcceptButton：记录用户键入回车时，相当于单击窗体中的那个按钮对象。
- CanceButton：记录用户键入 ESC 键时，相当于单击窗体中的那个按钮对象。以上两个

属性多用于对话框，例如打开文件对话框，用户键入回车，相当于单击确定按钮。

- MaxiMizeBox：窗体标题栏右侧最大化按钮是否可用，设置为 false，按钮不可用。
- MiniMizeBox：窗体标题栏右侧最小化按钮是否可用，设置为 false，按钮不可用。如果属性 MaxiMizeBox 和 MiniMizeBox 都设置为 false，将只有关闭按钮。在不希望用户改变窗体大小时，例如对话框，将两者都设置为 false。

2. Form 类常用方法

- Close()：窗体关闭，释放所有资源。如窗体为主窗体，执行此方法，程序结束。
- Hide()：隐藏窗体，但不破坏窗体，也不释放资源，可用方法 Show()重新打开。
- Show()：显示窗体。

3. Form 类常用事件

- Load：在窗体显示之前发生，可以在其事件处理函数中做一些初始化的工作。

3.3 标签(Label)控件

标签控件用来显示一行文本信息，但文本信息不能编辑，常用来输出标题、显示处理结果和标记窗体上的对象。标签一般不用于触发事件。

1. Label 控件常用属性

- Text：显示的字符串
- AutoSize：控件大小是否随字符串大小自动调整，默认值为 false，不调整。
- ForeColor：Label 显示的字符串颜色。
- Font：字符串所使用的字体，包括所使用的字体名，字体的大小，字体的风格等等，具体修改方法见下边的例子。

2. 例子 e3_3：我的第一个程序

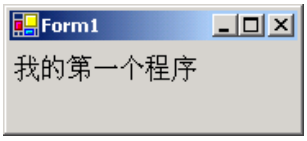
下面的例子在窗口中显示一行文本，该例虽然简单，但包括了用 Visual Studio.Net 建立 C# Windows 应用程序的基本步骤。具体实现步骤如下：

(1)建立一个新项目，生成一个空白窗体(Form1)，见图 2.4.2A。可以用属性窗口(图 2.4.2B 中图)修改窗体的属性，例如修改 Form1 的属性 Text，可以修改窗体的标题。用鼠标拖动窗体的边界小正方形，可以修改窗体打开时的初始大小。

(2)双击工具箱窗口(图 2.4.2B 左图)中 Windows 窗体类型下的 Label 条目，在窗体 Form1 放置一个 Label 控件。该控件用来显示一行文本。可以用鼠标拖放 Label 到窗体的任意位置，并可拖动 Label 边界改变控件的大小。

(3)选中 Label 控件，在属性窗口中找到属性 text，把它的值由“Label1”修改为“我的第一个程序”。接着在属性窗口中选中 Font 属性，单击 Font 属性右侧的标题为...的按钮，打开对话框，在对话框中可以修改 Label 控件显示字符串的字体名称和字号等，也可以单击 Font 属性左边的+号，在出现的子属性中编辑。编辑完成后，单击 Font 属性左边的-号，隐藏 Font 的子属性。修改 ForeColor 属性可以修改 Label 控件显示字符串的颜色。这是在设计阶段修改属性。

(4)编译，运行，可以看到窗口中按指定字体大小和颜色显示：我的第一个程序。运行效果如右图。



(5)保存项目。生成一个可执行程序需要多个文件，这些文件组成一个项目。一般把一个项目存到一个子目录中。单击文件/存所有文件菜单项，保存所有文件。

(6) 关掉 VS.NET，再启动。用文件/打开项目菜单项打开刚才关闭的项目文件(扩展名为 sln)。应能看到刚才关闭的设计界面。必须打开项目，才能完成编译工作。

3.4 按钮(Button)控件

用户单击按钮，触发单击事件，在单击事件处理函数中完成相应的工作。

1. Button 控件的常用属性和事件

- 属性 Text：按钮表面的标题
- 事件 Click：用户单击触发的事件，一般称作单击事件。

2. 例子 e3_4

本例说明如何用程序修改属性，如何使用方法，增加事件函数。该例在窗口中显示一行文字，增加 2 个按钮，单击标题为红色的按钮把显示的文本颜色改为红色，单击标题为黑色的按钮把显示的文本颜色改为黑色。实现步骤如下：

(1)继续上例，放三个 Button 控件到窗体，修改属性 Text，使标题分别为红色，黑色，退出。设计好的界面如右图。

(2)选中标题为红色的按钮，打开事件窗口(见图 2.4.2B 右图)，显示该控件所能响应的所有事件，其中左侧为事件名称，右侧为事件处理函数名称，如果为空白，表示还没有事件处理函数，选中 Click 事件，双击右侧空白处，增加单击(Click)标题为红色的按钮的事件处理函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{    label1.ForeColor=Color.Red;//运行阶段修改属性
}//注意 label1 是控件的名字(label 的 Name 属性)，用它来区分不同的控件。
```

(3)单击(Click)标题为黑色的按钮的事件处理函数如下：

```
private void button2_Click(object sender, System.EventArgs e)
{    label1.ForeColor=Color.Black;}
```

(4)单击(Click)标题为退出的按钮的事件处理函数如下：

```
private void button3_Click(object sender, System.EventArgs e)
{    Close();}
```

Close()为窗体(Form)的方法，作用是关闭注窗体。由于关闭了主窗体，程序也就结束了。注意，引用窗体的方法和属性时可不用指定对象名，换句话讲，如不指定属性或方法的对象名，默认为窗体的属性或方法。而使用其它组件的属性及方法要指明所属组件对象，例如 label1.ForeColor=Color.Red;

(5)编译，运行，单击标题为红色的按钮，窗体显示字符串颜色变为红色，单击标题为黑色的按钮，窗体显示字符串颜色变为黑色，单击标题为退出的按钮，结束程序。

3.5 事件处理函数的参数

事件处理函数一般有两个参数，第一个参数(object sender)为产生该事件的对象的属性 Name 的值，例如上例单击标题为红色的按钮，第一个参数 sender 的值为 button1。如上例标题为红色的按钮和标题为黑色的按钮使用同一个单击事件处理函数，其事件处理如下：

```
private void button1_Click(object sender,System.EventArgs e)
{    if(sender==button1)
         label1.ForeColor=Color.Red;
     else
         label1.ForeColor=Color.Black;
```

}

事件处理函数第二个参数(System.EventArgs e)代表事件的一些附加信息，事件不同，所代表的信息也不相同，例如在后边的例子中可以看到，按下鼠标的事件处理函数中，e.X 和 e.Y 分别为发生事件时鼠标位置的 x 坐标和 y 坐标，e.Button 表示用户单击了鼠标那个键，如为 MouseButtons.Left，表示单击了鼠标左键。

为了使这两个按钮使用相同的单击事件处理函数，首先为标题为红色的按钮增加单击事件处理函数，即是上边的代码，事件函数名称为：button1_Click。选中标题为黑色的按钮，打开事件窗体(见图 2.4.2B 右图)，选中 Click 事件，从其右侧下拉列表中选择事件处理函数为 button1_Click，这样两个按钮就使用相同的单击事件处理函数了。

3.6 文本框(TextBox)控件

TextBox 控件是用户输入文本的区域，也叫文本框。

1. TextBox 控件属性和事件

- 属性 Text：用户在文本框中键入的字符串
- 属性 MaxLength：单行文本框最大输入字符数。
- 属性 ReadOnly：布尔变量，为 true，文本框不能编辑。
- 属性 PasswordChar：字符串类型，允许输入一个字符，如输入一个字符，用户在文本框中输入的所有字符都显示这个字符。一般用来输入密码。
- 属性 MultiLine：布尔变量，为 true，多行文本框，为 false，单行文本框。
- 属性 ScrollBars：MultiLine=true 时有效，有 4 种选择：=0，无滚动条，=1，有水平滚动条，=2，有垂直滚动条，=3，有水平和垂直滚动条。
- 属性 SelLength：可选中文本框中的部分或全部字符，本属性为所选择的文本的字符数。
- 属性 SelStart：所选中文本的开始位置。
- 属性 SelText：所选中的文本
- 属性 AcceptsReturn：MultiLine=true 时有效，布尔变量，为 true，键入回车，换行，为 false，键入回车键，相当于单击窗体中的默认按钮。
- 事件 TextChanged：文本框中的字符发生变化时，发出的事件。

2. 例子 e3_6

本例要求用户在编辑框中输入两个乘数，单击按钮把相乘的结果在编辑框中显示出来。

(1)建立一个新的项目。放四个 Label 控件到窗体，Text 属性分别为：被乘数，乘数，积，*,=。

(2)放三个 textBox 控件到窗体，属性 Name 从左到右分别为：textBox1、textBox2、textBox3，属性 Text 都为空。

(3)放三个 Button 控件到窗体，Text 属性分别修改为求积，清空，退出。设计的界面如上图。

(4)标题为求积的按钮的单击事件处理函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   float ss,ee;
    ss=Convert.ToSingle(textBox1.Text);
    ee=Convert.ToSingle(textBox2.Text);
    textBox3.Text=Convert.ToString(ss*ee);
}
```

(5)标题为清空的按钮的单击事件处理函数如下：

```
private void button2_Click(object sender,System.EventArgs e)
{   textBox1.Text="";
    textBox2.Text="";
    textBox3.Text="";
}
```

(6)标题为退出的按钮的单击事件处理函数如下:

```
private void button3_Click(object sender, System.EventArgs e)
{   Close();}
```

(7) 编译，运行，在文本框 textBox1，textBox2 分别输入 2 和 3，单击标题为求积的按纽，textBox3 中显示 6，单击标题为清空的按钮，三个文本框被清空，单击标题为退出的按纽，结束程序。

3.7 Convert 类

Convert 类中提供了一些静态方法，用来把一种类型数据转换为另一种类型数据。例如，Convert.ToSingle(textBox1.Text) 把字符串 textBox1.Text 转换为单浮点数。Convert.ToString(3.14)把单浮点数 3.14 转换为字符串。其它转换函数还有：ToInt、ToInt16 等等。

3.8 单选按钮(RadioButton)和 GroupBox 控件

RadioButton 是单选按钮控件，多个 RadioButton 控件可以为一组，这一组内的 RadioButton 控件只能有一个被选中。GroupBox 控件是一个容器类控件，在其内部可放其它控件，表示其内部的所有控件为一组，其属性 Text 可用来表示此组控件的标题。例如把 RadioButton 控件放到 GroupBox 控件中，表示这些 RadioButton 控件是一组。有一些特性是互斥的，例如性别，选择这类特性可用 RadioButton 和 GroupBox 控件。

1. GroupBox 控件常用属性

GroupBox 控件常用属性只有一个，属性 Text，指定 GroupBox 控件顶部的标题。

2. RadioButton 控件属性和事件

- 属性 Text：单选按钮控件旁边的标题。
- 属性 Checked：布尔变量，为 true 表示按钮被选中，为 false 表示不被选中。
- 事件 CheckedChanged：单选按钮选中或不被选中状态改变时产生的事件。
- 事件 Click：单击单选按钮控件时产生的事件。

3. 例子 e3_8

该例用 RadioButton 控件修改 Label 控件字符串的字体为：宋体、黑体、楷体。具体实现步骤如下：

(1) 建立一个新的项目。

(2) 放 Label 控件到窗体,属性 Text="不同的字体"。字体为宋体。

(3) 放 GroupBox 控件到窗体,其属性 Text="选择字体"。

(4) 放三个 RadioButton 控件到 GroupBox 中,其属性 Text 分别为：宋体、黑体、楷体。宋体 RadioButton 控件的属性 Checked=true。设计好的界面如右图。

(5) 为三个 RadioButton 控件的 CheckedChanged 事件增加事件处理函数如下：

```
private void radioButton1_CheckedChanged(object sender, System.EventArgs e)
{   if(radioButton1.Checked)
    label1.Font=new Font("宋体",label1.Font.Size);
}//label1显示的字体变为宋体，字体大小不变
private void radioButton2_CheckedChanged(object sender, System.EventArgs e)
{   if(radioButton2.Checked)
    label1.Font=new Font("黑体",label1.Font.Size);
}
private void radioButton3_CheckedChanged(object sender, System.EventArgs e)
{   if(radioButton3.Checked)
    label1.Font=new Font("楷体_GB2312",label1.Font.Size);
}
```

(6) 编译，运行，单击 RadioGroup1 中的三个 RadioButton 按钮，可以改变字体。注意三个按钮只能选一个，既只能选一种字体。考虑一下，是否可用 Click 事件。

3.9 Font 类

Font 类有两个构造函数：第一个是 new Font(字体名称,字号)，例如，label1.Font=new Font("黑体",9),用法还可参考例 e3_8。第二个是 new Font(字体名称,字号，字体风格)，其中第三个参数是枚举类型，具体定义如下：

```
enum FontStyle{
    Regular     =0,//正常字体
    Bold        =1,//黑体
    Italic      =2,//斜体
    BoldItalic  =3,//黑斜体
    Underline   =4,//下划线，5=黑体下划线，6=斜体下划线，7=黑斜体下划线
    Strikeout   =8}//删除线，9=黑体删除线，10=斜体删除线，依此类推。
```

例如修改标签控件字体为斜体：

```
label1.Font=new Font("黑体",9,label1.Font.Style|FontStyle.Italic);
```

或者：label1.Font=new Font("黑体",9,label1.Font.Style|(FontStyle)2);

修改标签控件字体不为斜体：

```
label1.Font=new Font("黑体",9,label1.Font.Style&~FontStyle.Italic);
```

或者：label1.Font=new Font("黑体",9,label1.Font.Style&(FontStyle)(~2));

用法还可参考例 e3_11。

3.10 多选框(CheckBox)控件

CheckBox 是多选框控件，可将多个 CheckBox 控件放到 GroupBox 控件内形成一组，这一组内的 CheckBox 控件可以多选，不选或都选。可用来选择一些可共存的特性，例如一个人的爱好。

1. CheckBox 控件属性和事件

- 属性 Text：多选框控件旁边的标题。
- 属性 Checked：布尔变量，为 true 表示多选框被选中，为 false 不被选中。

- 事件 Click：单击多选框控件时产生的事件。
- 事件 CheckedChanged：多选框选中或不被选中状态改变时产生的事件。

2. 例子 e3_10A

在窗口中增加 2 个 CheckBox 控件，分别用来选择是否爱好音乐和是否爱好文学，用鼠标单击 CheckBox 控件，改变爱好选择，用 Label 控件显示所选择的爱好。实现步骤如下：

(1)建立新项目。放 Label 控件到窗体，属性 Text="你的爱好是："。

(2)放 GroupBox 控件到窗体，属性 Text="爱好"。放两个 CheckBox 控件到 GroupBox 中，属性 Text 分别为：音乐、文学。设计界面如下图。

(3)标题为音乐的多选框控件的 CheckedChanged 事件处理函数如下：

```
private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
{   String text1="你的爱好是：";
    if(checkBox1.Checked)
        text1=text1+checkBox1.Text;
    if(checkBox2.Checked)
        text1+=checkBox2.Text;
    label1.Text=text1;
}
```

(4)将标题为文学的多选框控件的 CheckedChanged 事件处理函数，设置为标题为音乐的多选框控件的 CheckedChanged 事件处理函数，具体步骤见 3.5 节。

(5)编译，运行。选中音乐将在标签控件中显示：你的爱好是：音乐，再选中文学显示：你的爱好是：音乐文学，...。

3. 例子 e3_10B

该例同上例，但按选中音乐和文学的顺序在标签中显示爱好，实现步骤如下：

(1)建立一个新项目。为 Form1 类增加私有变量 String s="你的爱好是："。

(2)放 Label 控件、GroupBox 控件、两个 CheckBox 到窗体，属性设置同上例。

(4)标题为音乐的多选框控件 CheckBox1 的 CheckedChanged 事件处理函数如下：

```
private void checkBox1_CheckedChanged(object sender,System.EventArgs e)
{   int n=s.IndexOf("音乐");//s中有字符串"音乐"吗?n=-1表示没有
    if(n==-1)//n=-1，表示上次没选此项，此次选中，应增加"音乐"
        s+="音乐";
    else//否则，表示上次已选此项，此次不选中，应删除"音乐"
        s=s.Remove(n,2);
    label1.Text=s;
}
```

(5)标题为文学的多选框控件 CheckBox2 的 CheckedChanged 事件处理函数如下：

```
private void checkBox2_CheckedChanged(object sender,System.EventArgs e)
{   int n=s.IndexOf("文学");//s中有字符串"文学"吗?=-1表示没有
    if(n==-1)//=-1，表示上次没选此项，此次选中，应增加"文学"
        s+="文学";
    else//否则，表示上次已选此项，此次不选中，应删除"文学"
        s=s.Remove(n,2);
    label1.Text=s;
}
```

(6)编译，运行。选中音乐在标签中显示：你的爱好是：音乐，再选中文学显示：你的爱好

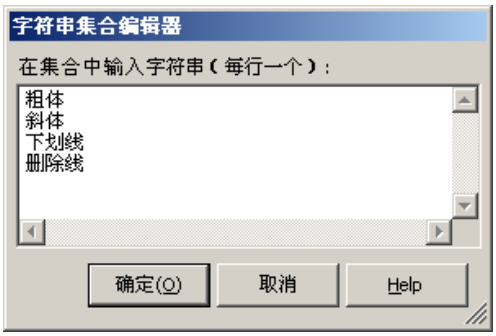
是：音乐文学，不选音乐显示：你的爱好是：文学，再选音乐显示：你的爱好是：文学音乐。

3.11 列表选择控件(ListBox)

列表选择控件列出所有供用户选择的选项，用户可从选项中选择一个或多个选项。

1. 列表选择控件的常用属性、事件和方法

- 属性 Items：存储 ListBox 中的列表内容，是 ArrayList 类对象，元素是字符串。
- 属性 SelectedIndex：所选择的条目的索引号，第一个条目索引号为 0。如允许多选，该属性返回任意一个选择的条目的索引号。如一个也没选，该值为-1。
- 属性 SelectedIndices：返回所有被选条目的索引号集合，是一个数组类对象。
- 属性 SelectedItem：返回所选择的条目的内容，即列表中选中的字符串。如允许多选，该属性返回选择的索引号最小的条目。如一个也没选，该值为空。
- 属性 SelectedItems：返回所有被选条目的内容，是一个字符串数组。
- 属性 SelectionMode：确定可选的条目数，以及选择多个条目的方法。属性值可以使：none(可以不选或选一个)、one(必须而且必选一个)、MultiSimple(多选)或 MultiExtended(用组合键多选)。
- 属性 Sorted：表示条目是否以字母顺序排序，默认值为 false，不允许。
- 方法 GetSelected()：参数是索引号，如该索引号被选中，返回值为 true。
- 事件 SelectedIndexChanged：当索引号(即选项)被改变时发生的事件。



2. 例子 e3_11

根据列表框的选择，为字符串加下划线、删除线、变斜体、变粗体。具体步骤如下：

(1) 建立一个新项目。放 Label 控件到窗体,其属性 Text="字体风格"。

(2) 放置 ListBox 控件到窗体中,属性 Name=listBox1。选中 ListBox 控件，在属性窗口中，单击 Items 属性右侧的三个小点，打开字符串集合编辑器对话框，在其中输入四项：粗体、斜体、下划线、删除线，注意每一项要换行。如上图。

(3) 设置列表选择控件 ListBox1 属性 SelectionMode 为 MultiExtended，允许多选。

(4) 为列表选择控件的事件 SelectedIndexChenged 增加事件处理函数如下：

```
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{   int Style=0,k=1;//Style=0正常字体，1=黑体，2=斜体，3=黑斜体等，参见3.9节
    for(int i=0;i<listBox1.Items.Count;i++)//此例Count=4，为什么？
    {if(listBox1.GetSelected(i))//例如此例GetSelected(0)=true表示粗体被选中
        Style=Style|k;//增加指定风格
     else
        Style=Style&(~k);//取消指定风格
     k=k*2;
    }
    FontStyle m=new FontStyle();
    m=(FontStyle)Style;
    label1.Font=new Font(label1.Font.Name,9,m);}
```

(5) 编译，运行，单选或用 Ctrl 键多选，看一下效果。运行效果如上图。

3.12 下拉列表组合框(ComboBox)控件

控件 ComboBox 中有一个文本框，可以在文本框输入字符，其右侧有一个向下的箭头，单击此箭头可以打开一个列表框，可以从列表框选择希望输入的内容。现介绍该控件用法。

1. ComboBox 控件的常用属性、事件和方法

- 属性 DropDownStyle：确定下拉列表组合框类型。为 Simple 表示文本框可编辑，列表部分永远可见。为 DropDown 是默认值，表示文本框可编辑，必须单击箭头才能看到列表部分。为 DropDownList 表示文本框不可编辑，必须单击箭头才能看到列表部分。
- 属性 Items：存储 ComboBox 中的列表内容，是 ArrayList 类对象，元素是字符串。
- 属性 MaxDropDownItems：下拉列表能显示的最大条目数(1—100)，如果实际条目数大于此数，将出现滚动条。
- 属性 Sorted：表示下拉列表框中条目是否以字母顺序排序，默认值为 false，不允许。
- 属性 SelectedItem：所选择条目的内容，即下拉列表中选中的字符串。如一个也没选，该值为空。其实，属性 Text 也是所选择的条目的内容。
- 属性 SelectedIndex：编辑框所选列表条目的索引号，列表条目索引号从 0 开始。如果编辑框未从列表中选择条目，该值为-1。
- 事件 SelectedIndexChanged：被选索引号改变时发生的事件。

2. 例子 e3_12 选择 Windows 操作系统提供的所有字体

增加一个 ComboBox 控件，用来选择字符串使用的字体名。本例提供方法使控件 ComboBox 的下拉列表中显示 Windows 操作系统中使用的所有字体名。运行效果如右图。实现步骤如下：

(1) 建立新项目。放 Label 控件到窗体, 其属性 Text="选择不同字体"。

(2) 放 ComboBox 控件到窗体中，属性 Name=comboBox1，属性 DropDownStyle=DropDownList，不能在编辑框中输入字体名，只能从下拉列表中选取。

(3) 为窗体 Form1 的事件 Load 增加事件处理函数如下：

```
private void Form1_Load(object sender, System.EventArgs e)
{//Families 是类 FontFamily 的一个静态属性，得到操作系统中所使用的所有字体名
  FontFamily[] families=FontFamily.Families;//静态属性没有类的对象也可使用
  foreach (FontFamily family in families)
        comboBox1.Items.Add(family.Name);//注意 Add 方法的使用
}
```

(4) 为 comboBox1 的事件 SelectedIndexChenged 增加事件处理函数如下：

```
private void comboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{label1.Font=new Font(comboBox1.Text,9);}
```

(5)编译，运行，在下拉列表中选择不同字体名，标签的字体变为选择的字体。从下拉列表中可以看到操作系统中的所有字体名称已经在列表中。

3.13 ToolTip 控件

在一些 Windows 应用程序中，例如 Word 程序，当鼠标在工具条的按钮上停留一段时间后，会在旁边出现提示，ToolTip 控件就是为实现此功能的。可以用 ToolTip 控件为任何控

件增加提示，本节介绍该控件的使用方法。

例子 e3_13 为 Button 控件增加提示

(1) 建立一个新项目。放 Button 控件到窗体，Name 属性为 Button1。

(2) 把 toolTip 控件放到窗体中，属性 Name=ToolTip1。

(3) 在 Form1 的构造函数中，增加语句如下：

```
toolTip1.SetToolTip(button1,"这是一个按钮");
```

(4) 编译，运行，当鼠标在 Button 上停留一段时间后，会在旁边出现提示：这是一个按钮。

3.14 超级链接(LinkLable)控件

控件 LinkLable 是控件 Label 的派生类，和控件 Label 不同的是显示的字符有下划线，可以为 LinkLable 控件的 LinkClicked 事件增加事件处理函数，当鼠标指向 LinkLable 控件，鼠标形状变为手形，单击该控件，调用这个事件处理函数，可以打开文件或网页。

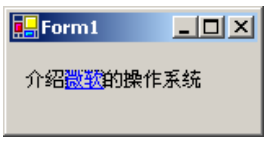
1. 超级链接控件的属性、方法和事件

- 属性 LinkColor：用户未访问过的链接的字符颜色，默认为蓝色。
- 属性 VisitedLinkColor：用户访问链接后的字符颜色。
- 属性 LinkVisited：如果已经访问过该链接，则为 true；否则为 false。
- 属性 LinkArea：是一个结构，变量 LinkArea.Start 表示字符串中开始加下划线的字符位置，LinkArea.Length 表示字符串中加下划线字符的个数。
- 事件 LinkClicked：单击控件 LinkLable 事件。

2. 例子 e3_14:用 LinkLabel 控件超级链接到微软网站。

(1) 建立一个新工程。放 LinkLabel 控件到窗体，属性 Text="介绍微软的操作系统"。

(2) 修改 LinkLabel 控件属性 LinkArea.Length=2，LinkArea.Start=2。也可在构造函数用语句修改：linkLabel1.LinkArea=new LinkArea(2,2);



(3) 为 LinkLabel 控件的事件 LinkClicked 增加事件处理函数：

```
private void linkLabel1_LinkClicked(object sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
{   linkLabel1.LinkVisited=true;
    System.Diagnostics.Process.Start("http://www.micosoft.com.cn");
}
```

(4) 运行，效果如右图，注意只有字符微软带下划线。单击微软，打开浏览器访问微软主页。

(5) 如果要打开一个窗口，列出 C 盘根目录下的文件及文件夹，LinkLabel 控件事件 LinkClicked 事件处理函数修改如下：

```
linkLabel1.LinkVisited=true;
System.Diagnostics.Process.Start("C:/");
```

(6) 如果要打开指定程序，例如打开记事本程序，修改 LinkClicked 事件处理函数如下：

```
linkLabel1.LinkVisited=true;
System.Diagnostics.Process.Start("notepad");
```

3.15 定时(Timer)控件

定时控件(Timer)也叫定时器或计时器控件，是按一定时间间隔周期性地自动触发事件的控件。在程序运行时，定时控件是不可见的。

3. 定时控件的属性、方法和事件

- 属性 Interval：周期性地自动触发事件的时间间隔，单位为毫秒。
- 属性 Enabled：为 true,启动定时器。调用方法 Start()也可启动定时器。
- 方法 Start()和 Stop()：启动和停止定时器。设置属性 Enabled=false 也可停止定时器。
- 事件 Tick：每间隔属性 Interval 指定的时间，产生事件 Tick。

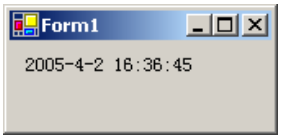
4. 例子 e3_15 用标签控件显示当前日期和时间

(1) 建立一个新项目。放 Timer 组件到窗体，Name 属性为 timer1。

(2) 放 Label 控件到窗体，Name 属性为 label1。

(3) 为窗体 Form1 的事件 Load 增加事件处理函数如下：

```
private void Form1_Load(object sender, System.EventArgs e)
{    this.timer1.Interval=100;
     this.timer1.Enabled=true;
     label1.Text=DateTime.Now.ToString();
}
```



(4) 为 Timer1 的 Tick 事件增加事件处理函数如下：

```
private void timer1_Tick(object sender, System.EventArgs e)
{    label1.Text=DateTime.Now.ToString();
}
```

(5) 编译，运行，标签控件位置显示日期和时间。运行效果如上图。

3.16 DateTime 类

DateTime 类中提供了一些静态方法，可以用来得到日期、星期和时间，下面是一些常用的方法。

- 得到日期和时间，并转换为字符串。

```
String s=DateTime.Now.ToString();//或 DateTime.Today.ToString()
```

- 得到年、月和日期

```
int y=DateTime.Now.Year;//得到年
int m=DateTime.Now.Month;//得到月
int d=DateTime.Now.Day;//得到日期
String s=DateTime.Now.DayOfWeek.ToString();//英文表示的星期
```

- 得到小时、分和秒

```
int h=DateTime.Now.Hour;//得到小时
int m=DateTime.Now.Minute;//得到分
int s=DateTime.Now.Second;//得到秒
```

- 定义一个 DateTime 类对象，表示 1999 年 1 月 13 日 3 时 57 分 32.11 秒

```
System.DateTime moment=new System.DateTime(1999,1,13,3,57,32,11);
```

- 加法和减法(减法请读者自己完成)

```
System.DateTime dTime=new System.DateTime(1980,8,5);//1980 年 8 月 5 日
//时间间隔，17 天 4 小时 2 分 1 秒
System.TimeSpan tSpan=new System.TimeSpan(17,4,2,1);
System.DateTime result=dTime+tSpan;//结果是：1980 年 8 月 22 日 4:2:1 AM.
```

3.17 菜单

Windows 应用程序一般都有一个菜单，通过选择菜单中的不同菜单项，完成指定的功能。使用主菜单控件 MainMenu 可以很容易建立 windows 应用程序的主菜单。

1. 菜单的组成及功能

放主菜单控件 MainMenu 到窗体中，可以为窗体增加一个主菜单。主菜单一般包括若干顶级菜单项，例如，文件、编辑、帮助等。单击顶级菜单项，可以出现弹出菜单，弹出菜单中包含若干菜单项，例如单击文件顶级菜单项，其弹出菜单一般包括打开文件、存文件、另存为等菜单项，用鼠标单击菜单项，可以执行菜单项命令。有的菜单项还包括子菜单。

所有菜单项都可以有快捷键，即菜单项中带有下划线的英文字符，当按住 ALT 键后，再按顶级菜单项的快捷键字符，可以打开该顶级菜单项的弹出菜单。弹出菜单出现后，按菜单项的快捷键字符，可以执行菜单项命令。增加快捷键的方法是在菜单项的标题中，在要设定快捷键英文字符的前边增加一个字符&，例如，菜单项的标题为：打开文件(&O)，菜单项的显示效果为：打开文件(O)。菜单项可以有加速键，一般在菜单项标题的后面显示，例如，菜单项打开文件的加速键一般是 Ctrl+O，不打开菜单，按住 Ctrl 键后，再按 O 键，也可以执行打开文件命令。设定加速键的方法是修改菜单项的 ShortCut 属性。

2. 用程序生成菜单

放主菜单控件 MainMenu 到窗体中，可以为该窗体增加一个主菜单，Visual Studio.Net 自动添加如下语句：

```
MainMenu mainMenu1=new MainMenu();
This.Menu=mainMenu1;//指定主窗口的主菜单是 mainMenu1。
```

可以建立多个 MainMenu 类对象，用第二条语句修改使主窗口使用不同的主菜单。有了主菜单对象，用如下语句为主菜单增加顶级菜单项：

```
MenuItem myFile=mainMenu1.MenuItem.Add(“文件(&F)”);//顶级菜单项：文件
```

有了顶级菜单项对象，用如下语句为顶级菜单项的弹出菜单增加菜单项：

```
myFile.MenuItem.Add(“打开(&O)”);//文件顶级菜单项的弹出菜单的菜单项：打开
```

实际上，这些都可以用 Visual Studio.Net 自动生成。

3. 菜单项的属性和事件

- 属性 Checked：布尔变量，=true，表示菜单项被选中，其后有标记：√。
- 属性 ShortCut：指定的加速键，可以从下拉列表中选择。
- 属性 ShowShortCut：布尔变量，true(默认值)，表示显示加速键，false，不显示。
- 属性 Text：菜单项标题。如为字符-，为分隔线。如指定字符前加&，例如：颜色(&c)，增加快捷键，即用 Alt+c 访问颜色菜单。
- 常用事件 Click：单击菜单项事件。

4. 例子 e3_17 增加菜单

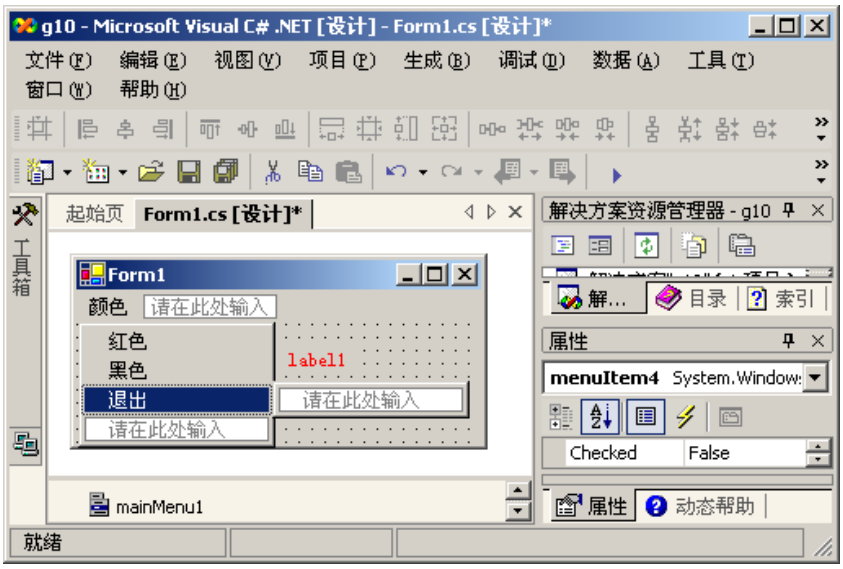
本例在窗体中建立主菜单，主菜单包括一个顶级菜单项：颜色，其弹出菜单包括两个菜单项：红色、黑色，单击标题为红色的菜单项，把窗体中显示的字符串变为红色，单击标题为黑色的菜单项，把窗体中显示的字符串变为黑色。实现步骤如下：

(1)建立一个新项目。放 Label 控件到窗体。

(2)双击工具箱中 Mainmenu 控件，在窗体中增加主菜单。右下角有一主菜单图标，在左上角有一方框，其中有文字：请在此处输入，在此方框中输入菜单标题。

(3)在方框内输入字符”颜色”,在其下部方框内输入字符”红色”为一菜单项，在”红色”下输入字符”黑色”为另一菜单项，再输入”退出”菜单项。如希望在选中某一菜单项后出现下一

级子菜单，可在菜单项右侧方框中输入子菜单项名。如果菜单项属性 Text 的值为-，则菜单项为分隔符。可以用鼠标拖动菜单项移动菜单项的位置。集成环境设计界面如下图。



(4)标题为红色的菜单项的单击(Click)事件处理函数如下：

```
private void menuItem2_Click(object sender,System.EventArgs e)
{label1.ForeColor=Color.Red;}//改变字体颜色为红色
```

(5)标题为黑色的菜单项的单击(Click)事件处理函数如下：

```
private void menuItem3_Click(object sender, System.EventArgs e)
{label1.ForeColor=Color.Black;}//改变字体颜色为黑色
```

(6)标题为退出的菜单项的单击(Click)事件处理函数如下：

```
private void menuItem4_Click(object sender, System.EventArgs e)
{   Close();}//退出程序
```

(7) 编译，运行，单击红色和黑色菜单项，能改变字符串的颜色。效果如上图。

3.18 工具条

一般 Windows 应用程序都有一个工具条，可以认为工具条上的按钮为菜单的某一菜单项的快捷按钮，单击工具条按钮相当于单击相应菜单项，完成同样的功能。

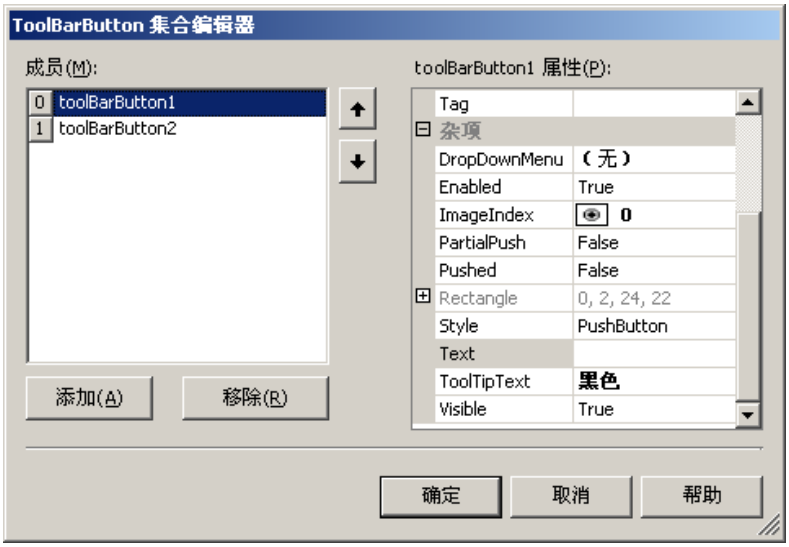
1. 工具条的组成及功能

放工具条控件 ToolBar 到窗体中，可以为该窗体增加一个工具条。在工具条中可以增加 Button 按钮和其它控件，例如象 Word 程序的工具条中用下拉列表控件(ComboBox)选择字号、字体等。一般工具条按钮上都有一个图标，提示用户该按钮的使用功能。按钮的所有图标存放到 ImageList 类对象中。单击任何一个按钮，都产生工具条控件的 ButtonClick 事件，在这个事件处理事件函数中，要用语句区分用户单击了那一个按钮，以完成相应的功能。

2. 控件 ToolBar 的属性、事件和方法

- 属性 BorderStyle：边界风格，=None(默认值)，无边界；=FixedSingle，单线边界；=Fixed3D，立体风格边界。
- 属性 Button：集合属性，存储 ToolBar 的按钮对象。单击其后的按钮，可以打开

ToolBarButton 集合编辑器对话框(见下图)，增加或删除按钮，修改按钮属性。



- 属性 ImageList：指定一个 ImageList 类对象，该对象中可以存储若干图标，这些图标作为 ToolBar 控件按钮的图标。
- 属性 Wrappable：布尔变量，=true(默认值)，当窗体 Form 水平尺寸小于工具条的水平尺寸时，一行不能显示所有按钮，允许下一行显示；=false，不允许。
- 事件 ButtonClick：ToolBar 控件的单击事件。在 ButtonClick 事件处理事件函数中，要用语句区分用户单击了那一个按钮，以完成相应的功能。
- 属性 ShowToolTips：布尔变量，=true，允许显示提示信息。
- 方法 IndexOF()：参数为 ToolBar 控件中按钮的属性 Name，返回其索引值。

3. ToolBar 控件中 ToolBarButton 按钮的属性

ToolBar 控件中 ToolBarButton 按钮可以看作独立的控件，它有自己独立的属性。下面介绍 ToolBar 控件中 ToolBarButton 按钮的属性。

- 属性 ImageIndex：ToolBar 控件属性 ImageList 指定一个 ImageList 类对象，该对象中的图标作为 ToolBar 控件按钮的图标。这个属性指定本按钮使用 ImageList 类对象中存储的第几个图标。
- 属性 Style：有 4 个值，=PushButton，为普通按钮；=Separator，为一分割符，再左边和右边的两个按钮中间增加一个间隙；=ToggleButton，开关按钮，单击该按钮，按钮被按下，不抬起，再单击，抬起。=DropDownButton，下拉按钮，按钮右侧有一个下拉箭头，单击下拉箭头，可以弹出下拉列表。
- 属性 Text：ToolBar 控件中按钮除了有图标外，还可以有属性 Text 指定的文字。
- 属性 ToolTipText：当鼠标在工具条按钮上停留一段时间后，将在工具条按钮旁边出现此属性指定的提示。

4. 例子 e3_18

现为上例的菜单增加工具条，有两个按钮，单击按钮分别使字体变红、变黑。步骤如下：

(1) 继续菜单的例子，放 ImageList 控件到窗体。

(2) 放 ToolBar 控件到窗体。修改属性 ImageList=ImageList1。

(3) 单击 ImageList 属性 Images 后按钮，打开 Image 集合编辑器，单击添加按钮，打开选择文件对话框。按指定路径选择图标的文件后，单击确定按钮，增加图标到 ImageList

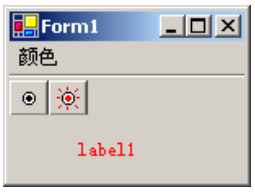
对象中。在 C:\Program Files\Microsoft Office\Office\forms\2052 文件夹和 C:\program files\Microsoft Visual Studio.Net\Common7\Graphics\Icon\Misc 文件夹中有若干图标。也可用画笔程序自己设计图标，图标的宽和高应比工具条按钮的宽和高略小，存为.ico 文件。也可以用抓图软件抓其它程序的图标。任选以上方法，为 ImageList 对象增加两个图标。

(4) 单击 ToolBar 控件属性 Buttons 后按钮，打开 ToolBarButton 集合编辑器(见上图)，单击添加按钮，增加一个按钮，从其属性 ImageIndex 后的下拉列表中选择按钮使用的图标，设置按钮的 ToolTipText 属性为：改变字体为红色，为工具按钮增加提示。同样方法增加第二个按钮，按钮的 ToolTipText 属性为：改变字体为黑色。

(5) 设定 ToolBar 控件属性 ShowToolTips 为 true。

(6) 为 ToolBar 控件的 ButtonClick 事件增加事件函数如下：

```
private void toolBar1_ButtonClick(object sender,
                    System.Windows.Forms.ToolBarButtonClickEventArgs e)
{   int n=toolBar1.Buttons.IndexOf(e.Button);//n为工具条中被单击按钮的序号
    switch(n)
    {   case 0://第一个按钮，调用相应的菜单项的事件处理函数。
            this.menuItem3_Click(sender,e);
            break;
        case 1://第二个按钮
            this.menuItem2_Click(sender,e);
            break;
    }
 }
```



(7) 编译，运行，单击两个工具条按钮，可以分别使字体变为红色或黑色。见上图。

3.19 状态栏(StatusBar)控件

Windows 应用程序的状态栏一般用来显示一些信息，如时间，鼠标位置等。

1. 状态栏控件的属性

- 属性 Panels：集合属性，存储状态栏中的各个分栏对象。单击其后标题为...的按钮，可以打开 StatusBarPanels 集合编辑器对话框，增加或删除分栏，修改分栏属性。
- 属性 ShowPanel：布尔变量，=true，允许显示多栏；=false，不允许。

2. 状态栏(StatusBar)控件分栏的属性

状态条可以为单栏，也可以为多栏。属性 Text，表示在状态栏中显示的内容。如为单栏，在单栏中显示字符串的语句是：statusBar1.Text="在单栏中显示的文本"，如为多栏，在第 2 栏中显示字符串的语句是：statusBar1.Panels[1].Text="在第 2 栏中显示的文本"。

- 属性 Alignment：对齐方式，可以为左对齐、右对齐和中间对齐。
- 属性 Text：表示在状态栏中显示的内容。
- 属性 Width：栏的宽度。
- 属性 BorderStyle：指定状态栏控件上 每个分栏的边框外观。边界风格，=None(默认值)，不显示边框；=**Raised**，三维凸起边框；=**Sunken**，三维凹陷边框显示。

3. 例子 e3_19 为窗体增加状态条，在状态条内显示时间和鼠标位置。

(1) 建立新项目。放 StatusBar 控件到窗体。单击 StatusBar 控件属性 Panels 后按钮，打开 StatusBarPanels 集合编辑器(如下图)，单击添加按钮，增加若 2 栏。其序号为 0、1。

(2) 修改 StatusBar 控件属性 ShowPanel=true。

(3) 放 Timer 组件到窗体，Name=Timer1，属性 Interval=1000，Enabled=true。

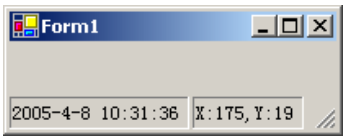
(4) 为 Timer1 的 Tick 事件增加事件处理函数如下：

```
private void timer1_Tick(object sender, System.EventArgs e)
{   statusBar1.Panels[0].Text=DateTime.Now.ToString();
}
```

(5) 为 Form1 的 MouseMove 事件增加事件处理函数如下：

```
private void Form1_MouseMove(object sender,System.Windows.Forms.MouseEventArgs e)
{statusBar1.Panels[1].Text="X:"+e.X.ToString()+",Y:"+e.Y.ToString();
}
```

(6) 编译，运行，如右图，在第 1 栏中可以看到当前时间，在窗口中移动鼠标，在第 2 栏中可以看到鼠标的位置不断变化。



3.20 鼠标事件

从类 System.Windows.Forms.Control 派生的控件都有鼠标事件，控件的 Click 事件本质上也是鼠标事件。一些控件还有单独的鼠标事件，例如 Form。鼠标事件有：

- MouseDown：如果鼠标位于控件区域，按下鼠标按键时产生该事件。
- MouseUp：如果鼠标位于控件区域，抬起鼠标按键时产生该事件。
- MouseMove：如果鼠标在控件区域移动，产生该事件。
- MouseEnter：鼠标进入控件区域，产生该事件。
- MouseLeave：鼠标离开控件区域，产生该事件。

鼠标事件处理函数一般有两个参数，第一个参数(object sender)是产生该事件的对象的属性 Name 的值，例如，为 Form1 的 MouseDown 事件增加事件函数，单击 Form1，第一个参数 sender 代表 Form1 对象。(System.Windows.Forms.MouseEventArgs e)是事件处理函数第二个参数，代表事件的一些信息，事件不同，所代表的信息也不相同，鼠标按下事件处理函数中，e.X 为发生事件时鼠标位置的 x 坐标，e.Y 为发生事件时鼠标位置的 y 坐标，e.Button

为 MouseButtons.Left，表示单击了鼠标左键等等，Right 和 Middle 则分别代表右键和中间键。e.Clicks 为鼠标单击的次数，如果大于 2 次，则为双击。

例子 e3_20：在窗体中的指定区域，双击鼠标左键，用 Label 控件显示双击鼠标的位置。指定区域的左上角坐标为(20,20)，宽为 200，高为 200。

(1) 建立一个新项目。放 Label 控件到窗体。属性 Name=label1。

(2) Panel 控件可以将窗体分为多个区域。放 Panel 控件到窗体，属性 Location.X=20，Location.Y=20，属性 Width=200，Height=200，属性 Name=p1。

(3) 为 Panel 的 MouseDown 事件增加事件函数如下：

```
private void p1_MouseDown(object sender,System.Windows.Forms.MouseEventArgs e)
{   if(e.Button==MouseButtons.Left&&e.Clicks>1)//如果是双击左键
        label1.Text="X:"+e.X.ToString()+",Y:"+e.Y.ToString();
}
```

(4) 编译，运行，分别在指定区域和区域外双击鼠标左键，看一下效果。分别在指定区域和区域外双击鼠标右键，看一下效果。

3.21 快捷菜单(ContextMenu)

使用过 Word 程序的人都知道，在其程序窗口的不同位置单击右键，会出现不同弹出菜单，这个弹出菜单叫快捷菜单，这节介绍如何在应用程序中增加快捷菜单。快捷菜单和主菜单的属性、事件和方法基本一致，只是快捷菜单没有顶级菜单项，因此这里就不多介绍了。

例子 e3.21

例子在窗口中显示一行字符串，加入两个按纽，单击按纽 button1 把字符串变为红色，单击按纽 button2 把字符串变为黑色。为两个按钮建立快捷菜单，快捷菜单中有 2 个菜单项，单击菜单项把字符串变为红色或黑色。为窗体建立快捷菜单，菜单中仅有 1 个退出菜单项，单击退出菜单项，退出程序。具体实现步骤如下：

(1) 建立一个新项目。放 Label 控件到窗体。

(2) 放 2 个 Button 控件到窗体，标题（属性 Text）分别为红色，黑色。

(3) 标题为红色的按钮的单击事件处理函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   label1.ForeColor=Color.Red;}
```

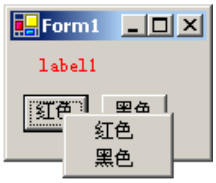
(4) 标题为黑色的按钮的单击事件处理函数如下：

```
private void button2_Click(object sender, System.EventArgs e)
{   label1.ForeColor=Color.Black;}
```

(5) 放 2 个 ContextMenu 控件到窗体，属性 Name 分别为 contextMenu1，contextMenu2。

(6) 选中 contextMenu1 控件，在菜单编辑器中增加两个标题分别为红色和黑色的菜单项，它们的单击事件处理函数分别是单击红色按钮和单击黑色按钮的事件处理函数。

(7) 选中 contextMenu2 控件，在菜单编辑器中增加标题为退出的菜单项，并为其增加单击事件处理函数，为事件处理函数增加语句：Close();



(8) 将红色按钮和黑色按钮的属性 ContextMenu 指定为 contextMenu1。Form 的属性 ContextMenu 指定为 contextMenu2。

(9) 编译，运行，右击标题为红色的按钮，快捷菜单 contextMenu1 打开，单击快捷菜单中标题为红色的菜单项，将使窗体显示的字符串颜色变为红色，右击标题为黑色的按钮，快捷菜单 contextMenu1 打开，单击快捷菜单中标题为黑色的菜

单项，将使窗体显示的字符串颜色变为黑色，右击窗体，快捷菜单 contextMenu2 打开，单击快捷菜单中标题为退出的菜单项，将退出应用程序。运行效果如上图。

3.22 综合例子：计算器

具体步骤如下：

(1) 建立一个新项目。Form 属性 MaxiMizeBox=false，属性 MiniMizeBox=false。属性 FormBorderStyle=FixedDialog，窗口不能修改大小。

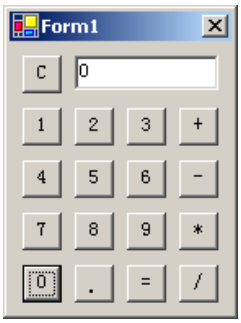
(2) 放 textBox 控件到窗体，属性 Name=textBox1，属性 Text="0"，属性 ReadOnly=true。

(3) 增加 10 个 Button 控件，前 9 个按钮属性 Name 分别为：Button1-Button9，最后一个为 Button0，属性 Text 分别为：1、2、3、4、5、6、7、8、9、0。

(4) 增加 7 个 Button 控件,属性 Name 分别为:btn_dot、btn_equ、btn_add、btn_sub、btn_mul、btn_div、btn_C，属性 Text 分别为：.、=、+、-、*、/、C。设计界面如下图。

(5) 控件 Button0 单击事件处理函数如下：

```
private void button0_Click(object sender, System.EventArgs e)
{   if(sender==button0) append_num(0);
    if(sender==button1) append_num(1);
    if(sender==button2) append_num(2);
    if(sender==button3) append_num(3);
    if(sender==button4) append_num(4);
    if(sender==button5) append_num(5);
    if(sender==button6) append_num(6);
    if(sender==button7) append_num(7);
    if(sender==button8) append_num(8);
    if(sender==button9) append_num(9);
}
```



(6) 为 Form1 类增加方法如下：

```
public void append_num(int i)
{   if(textBox1.Text!="0")
        textBox1.Text+=Convert.ToString(i);
    else
        textBox1.Text=Convert.ToString(i);
}
```

(7) 将 Button1-Button9 的单击事件处理函数设定为 Button0 单击事件处理函数。

(8) 为标题为.按钮增加事件处理函数如下：

```
private void btn_dot_Click(object sender, System.EventArgs e)
{   int n=textBox1.Text.IndexOf(".");
    if(n==-1)//如果没有小数点，增加小数点，否则不增加
        textBox1.Text=textBox1.Text+".";
}
```

(9) 编译，单击数字按钮，在 textBox1 可以看到输入的数字，也可以输入小数。

(10) 先实现加法，必须定义一个浮点类型变量 sum，初始值为 0，记录部分和。

(11) 输入了第一个加数，然后输入任一运算符(+、-、*、\或=)，应首先清除编辑框中显示的第一个加数，才能输入第二个加数。为实现此功能，必须定义一个布尔变量 blnClear，

初始值为 false，表示输入数字或小数点前不清除编辑框中显示，输入运算符(+、-、*、\或=)后，blnClear=true，表示再输入数字或小数点先清除编辑框中显示。修改前边程序，输入数字或小数点前，要判断变量 blnClear，如为 true，清除编辑框中显示的内容后，再显示新输入的数字或小数点，同时修改 blnClear=false。为此修改 append_num 方法如下：

```
public void append_num(int i)
{   if(blnClear)//如果准备输入下一个加数，应先清除textBox1显示内容
    {   textBox1.Text="0";//阴影部分为新增语句
        blnClear=false;
    }
    if(textBox1.Text!="0")
        textBox1.Text+=Convert.ToString(i);
    else
        textBox1.Text=Convert.ToString(i);
}
```

(12) 修改 btn_dot_Click 方法如下：

```
private void btn_dot_Click(object sender, System.EventArgs e)
{   if(blnClear) //如果准备输入下一个数，应先清除textBox1显示内容
    {   textBox1.Text="0";//阴影部分为新增语句
        blnClear=false;
    }
    int n=textBox1.Text.IndexOf(".");
    if(n==-1)//如果没有小数点，增加小数点，防止多次输入小数点
        textBox1.Text=textBox1.Text+".";
}
```

(13) 如果计算 1+2-3 的运算结果，先单击按钮 1，编辑框中显示 1，再单击按钮+,执行运算 sum=sum+1(注意此时 sum=0)，显示 sum 到编辑框中(实际显示不变)，记住此次输入的运算符，这里为+号。单击按钮 2，编辑框中显示 2，再单击按钮-,按记录的运算符(这里是+)计算 sum=sum+2，显示 sum 到编辑框中，记住此次输入的运算符，这里为-号，依此类推。为实现此功能，必须定义一个字符串变量 strOper，记录输入的运算符，初始值为"+"，保证输入第一个运算符后，执行运算 sum=sum+第一个加数，由于初始 sum=0，也就是 sum=第一个加数。标题为+的按钮的单击事件处理函数如下：

```
private void btn_add_Click(object sender, System.EventArgs e)
{   double dbSecond=Convert.ToDouble(textBox1.Text);
    if(!blnClear)//如果未输入第二个操作数，不运算
        switch(strOper)//按记录的运算符号运算
        {   case "+":
            sum+=dbSecond;
                break;
        //在此增加其它运算符-、*、\代码
        }
    if(sender==btn_add)
        strOper="+";
    //在此增加运算符-、*、\、=代码
```

```
    textBox1.Text=Convert.ToString(sum);
    blnClear=true;
}
```

(14) =号处理语句和+号处理基本一致，修改标题为+按钮的事件函数如下：

```
private void btn_add_Click(object sender, System.EventArgs e)
{   double dbSecond=Convert.ToDouble(textBox1.Text);
    if(!blnClear)//如果未输入第二个操作数，不运算
        switch(strOper)//按记录的运算符号运算
        {   case "+":
            sum+=dbSecond;
                break;
        //在此增加运算符-、*、\代码
        }
    if(sender==btn_add)
        strOper="+";
    if(sender==btn_equ)//为=号处理增加的语句
        strOper="=";
    textBox1.Text=Convert.ToString(sum);
    blnClear=true;
}
```

将 btn_equ 按钮的单击事件函数设定为+按钮的单击事件函数。

(15) 为标题为 C 按钮增加事件函数如下：

```
private void btn_C_Click(object sender, System.EventArgs e)
{   textBox1.Text="0";
    sum=0;
    blnClear=false;
    strOper="+";
}
```

(16) 请读者自己补上减法，乘法，除法运算的语句。

习题：

(1) 在窗口中显示一行字符串，加入两个按钮，单击按钮 1 把字符串改为红色，单击按钮 2 把字符串改为黑色。使字符串为红色时红色按钮不能使用，字符串为黑色时黑色按钮不能使用。(提示：可以修改按钮的属性 Enabled 为 false 使其不能使用。)

(2) 将上题改为用按扭修改字体的大小，分别为大字体和小字体。(参见 3.9 节)

(3) 加一文本框控件和一按纽，单击按纽将文本框控件输入内容显示标签控件上。(提示：单击按钮事件处理函数中加语句 label1.Text=textBox1.Text)。

(4) 修改上题，使文本框控件和标签控件文本同步显示(提示：文本框控件的 TextChanged 事件处理函数中加语句 label1.Text=textBox1.Text)。

(5) 加一文本框控件和一按纽，单击按纽将文本框控件输入的文本中选中的内容显示在标签控件上（提示：单击按钮事件处理函数中加语句 label1.Text=textBox1.SelText。）

(6) 加一文本框控件和一按纽，单击按纽将文本框控件输入的文本的字符、选中的内容的字符数和选中的内容的开始位置显示在标签控件上。

(7) 用控件 RadioButton 选择性别，把选择的结果用 Label 控件显示出来。
(8) 例子 e3_8 中如改为响应单击事件 Click，可能出现什么问题？
(9) 用控件 ComboBox 修改标签控件字体的大小。(用属性 Item 在下拉列表中输入大小)。
(10)放 ListBox 控件到窗体中,属性 Name=listBox1。列表框有三项分别为：苹果，梨子，香蕉。允许多选。标签控件同步显示 ListBox 控件所做的选择。提示：为 ListBox 控件的 SelectedIndexChenged 事件增加事件函数，

```
label1.Text="所选择的是：";
for(int i=0;i<listBox1.SelectedIndices.Count;i++)
    label1.Text+=listBox1.SelectedItems[i].ToString()+", ";
```

(11) 放 ListBox、TextBox 和 3 个 Button 控件到窗体中,属性 Name 分别为 listBox1、textBox1、Button1、Button2、Button3。Button 控件属性 Text 分别为：增加、删除、清空。单击增加按钮，把 textBox 中输入的内容作为一个条目增加到 listBox1 中，单击删除按钮，删除 listBox1 中所选择的条目，单击清空按钮，清除 listBox1 所有条目。提示：增加用语句：listBox1.Items.Add(textBooooox1.Text)。删除所选择的条目用语句：listBox1.Items.RemoveAt(listBox1.SelectedIndex)。清除 listBox1 所有条目用语句：listBox1.Items.Clear()。
(12) 在窗体中显示字符，每隔 1 秒字体变大些，变到一定尺寸后，每隔 1 秒字体变小些，如此循环。增加一个按钮，可以启动和停止字符串字体大小变化，按钮标题给出正确提示。
(13) 在窗体中显示字符，每隔 1 秒字符移动一定距离，先右移，移到右边界，再左移，移到左边界，又一次右移，如此循环。(提示：修改 Label 的 Left 属性值。)
(14) 修改例子 e3_17，使显示字符串为红色时，标题为红色的菜单项无效；使显示字符串黑色时，标题为黑色的菜单项无效。
(15) 修改例子 e3_17，使显示字符串为红色时，标题为红色的菜单项前增加选中标志；使显示字符串黑色时，标题为黑色的菜单项前增加选中标志。
(16) 为例 e3_17 的菜单项增加加速键，键入 Alt+c 打开顶级菜单项颜色的弹出菜单，弹出菜单打开后,键入 B 执行标题为黑色的菜单项命令,键入 R 执行标题为红色的菜单项命令。
(17) 为例子 e3_17 的菜单项定义加速键（属性 Shortcut），键入 ctrl+r 使显示字符串为红色，键入 ctrl+b 使显示字符串黑色。
(18) 为例子 e3_17 顶级菜单项颜色增加单击事件处理函数,在事件处理函数中判断显示的字符串的颜色，决定是否为相应的菜单项增加选中标志。
(19) 拖动鼠标左键时，在状态栏中显示鼠标的位置。
(20) 模拟画笔程序，在左侧增加工具按钮，在下部增加颜色按钮。
~~(21)~~在工具栏中加三个按钮，单击按钮时，按钮保持按下状态，再单击按钮，按钮抬起。在按下状态，使标签控件中字符串加下画线、斜体或加粗，抬起则取消。

带格式的：项目符号和编号

~~(21) 工具栏中按钮的属性 Style 设置为 ToolBarButtonStyle.DropDownButton，按钮可有一个下拉菜单。首先创建一个 ContextMenu 菜单，指定工具栏中按钮的属性 DropDownMenu 的值为创建的 ContextMenu 菜单对象，将在按下按钮时显示这个菜单。请用工具栏中按钮的下拉菜单实现使标签控件字符的颜色变为红色、黑色。~~(提示：工具栏中按钮的属性 Style 设置为 ToggleButton。属性 Pushed 是一个布尔变量，表示工具栏按钮当前是否处于按下状态)

(22)用工具栏中按钮的下拉菜单实现使标签控件字符的颜色变为红色、黑色。(提示：如工具栏中按钮的属性 Style 设置为 DropDownButton，按钮可有一个下拉菜单。首先创建一个 ContextMenu 菜单，指定工具栏中按钮的属性 DropDownMenu 的值为创建的 ContextMenu 菜单对象，将在按下按钮时显示这个菜单。)

(23) 完成计算器的减法和乘除程序。增加求平方，对数等功能。(例如 Math.Sqrt())

第四章 文本编辑器的实现

本章的目的是建立一个文本编辑器，同时继续介绍控件的用法。有两类文本编辑器：单文档文本编辑器和多文档文本编辑器，单文档文本编辑器一次只允许打开一个文件，如果要打开另一个文件，必须关闭当前打开的文件，微软的写字板程序就是一个典型的单文档字处理程序。多文档文本编辑器同时允许打开多个文件，每个文件占用一个子窗口，微软的 Word 程序就是一个典型的多文档字处理程序。本章首先介绍建立一个单文档文本编辑器的方法，然后介绍建立多文档文本编辑器的方法。

4.1 用 RichTextBox 控件实现文本编辑器

RichTextBox 控件可以用来输入和编辑文本，该控件和 TextBox 控件有许多相同的属性、事件和方法，但比 TextBox 控件的功能多，除了 TextBox 控件的功能外，还可以设定文字的颜色、字体和段落格式，支持字符串查找功能，支持 rtf 格式等。这里只介绍在 TextBox 控件中没有介绍的属性、事件和方法，相同部分就不介绍了，可参见 TextBox 控件。RichTextBox 控件的属性、事件和方法如下：

- 属性 Dock：很多控件都有此属性，它设定控件在窗体中的位置，可以是枚举类型 DockStyle 的成员 None、Left、Right、Top、Bottom 或 Fill，分别表示在窗体的任意位置、左侧、右侧、顶部、底部或充满客户区。在属性窗口中，属性 DOCK 的值用周边 5 个矩形，中间一个矩形的图形来表示。
- 属性 SelectedText：获取或设置 RichTextBox 控件内的选定文本。
- 属性 SelectionLength：获取或设置 RichTextBox 控件中选定文本的字符数。
- 属性 SelectionStart：获取或设置 RichTextBox 控件中选定的文本起始点。
- 属性 SelectionFont：如果已选定文本，获取或设置选定文本字体，如果未选定文本，获取当前输入字符采用字体或设置以后输入字符采用字体。
- 属性 SelectionColor：如果已选定文本，获取或设置选定文本的颜色，如果未选定文本，获取当前输入字符采用的颜色或设置以后输入字符采用的颜色。
- 属性 Lines：记录 RichTextBox 控件中所有文本的字符串数组，每两个回车之间字符串是数组的一个元素。
- 属性 Modified：指示用户是否已修改控件的内容。为 true，表示已修改。
- 事件 SelectionChange：RichTextBox 控件内的选定文本更改时发生的事件。
- 事件 TextChanged：RichTextBox 控件内的文本内容改变时发生的事件。
- 方法 Clear()：清除 RichTextBox 控件中用户输入的所有内容，即清空属性 Lines。
- 方法 Copy()、Cut()、Paste()：实现 RichTextBox 控件的拷贝、剪贴、粘贴功能。
- 方法 SelectAll()：选择 RichTextBox 控件内的所有文本。
- 方法 Find()：实现查找功能。从第二个参数指定的位置，查找第一个参数指定的字符串，并返回找到的第一个匹配字符串的位置。返回负值，表示未找到匹配字符串。第三个参数指定查找的一些附加条件，可以是枚举类型 RichTextBoxFinds 的成员：MatchCase(区分大小写)、Reverse(反向查找)等。允许有 1 个、2 个或 3 个参数。
- 方法 SaveFile()：存文件，它有 2 个参数，第一个参数为要存文件的全路径和文件名，

第二个参数是文件类型，可以是：纯文本，RichTextBoxStreamType.PlainText；Rtf 格式流，RichTextBoxStreamType.RichText；采用 Unicode 编码的文本流，RichTextBoxStreamType.UnicodePlainText。

- 方法 LoadFile()：读文件，参数同方法 SaveFile()，注意存取文件的类型必须一致。
- 方法 Undo()：撤消 RichTextBox 控件中的上一个编辑操作。
- 方法 Redo()：重新应用 RichTextBox 控件中上次撤消的操作。

4.2 实现文本编辑器的剪贴板功能

许多程序都支持剪贴板功能。通过剪贴板可以完成数据的剪贴（Cut），复制（Copy），粘贴（Paste）等功能。剪贴板可以理解为一块存储数据的公共区域，用户可以把数据复制或剪贴到剪贴板中，本任务或其它任务要用剪贴板中的数据时，可以用粘贴功能从剪贴板中把数据取出。存入剪贴板中的数据，可以是字符，位图，或者其它格式数据。实现文本编辑器的编辑和剪贴板功能的具体步骤如下：

(1) 新建项目。放 RichTextBox 控件到窗体。属性 Name=richTextBox1，Dock=Fill，Text=""。

(2) 放 Mainmenu 控件到窗体中。增加顶级菜单项：编辑，为其弹出菜单增加菜单项：剪切、复制、粘贴、撤销和恢复，属性 Name 分别为：mainMenuEdit、menuItemEditCut、menuItemEditCopy、menuItemEditPaste、menuItemEditUndo、menuItemEditRedo。为各个菜单项增加事件处理函数如下：

```
private void menuItemEditCut_Click(object sender, System.EventArgs e)
{   richTextBox1.Cut();}            //剪切
private void menuItemEditCopy_Click(object sender, System.EventArgs e)
{   richTextBox1.Copy();}           //拷贝
private void menuItemEditPaste_Click(object sender, System.EventArgs e)
{   richTextBox1.Paste();}          //粘贴
private void menuItemEditUndo_Click(object sender, System.EventArgs e)
{   richTextBox1.Undo();}           //撤销
private void menuItemEditRedo_Click(object sender, System.EventArgs e)
{   richTextBox1.Redo();}           //恢复
```

(3) 编译，运行，输入一些字符后，选中一些字符，试验一下剪切、复制、粘贴等功能，并查看一下在剪贴板中字符是否能粘贴到其它字处理软件中，例如写字板。查看一下撤销和恢复功能是否可用。

4.3 实现文本编辑器的存取文件功能

文本编辑器都具有文件存取功能，顶级菜单项文件的弹出菜单中一般包括如下菜单项：新建、打开、关闭、保存和另存为等。本节实现以上菜单项。

4.3.1 OpenFileDialog 和 SaveFileDialog 控件

OpenFileDialog 对话框用来选择要打开的文件路径及文件名，SaveFileDialog 对话框用来选择要存储文件的路径及文件名。两个对话框的外观如下图，它们的属性和方法基本相同，

这里在一起介绍。

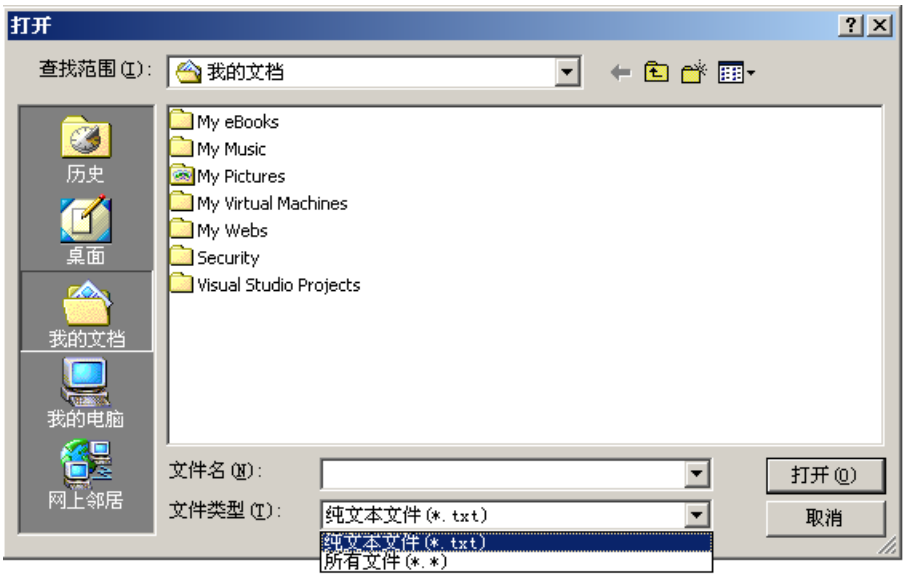


图 4.3.1A 打开文件对话框

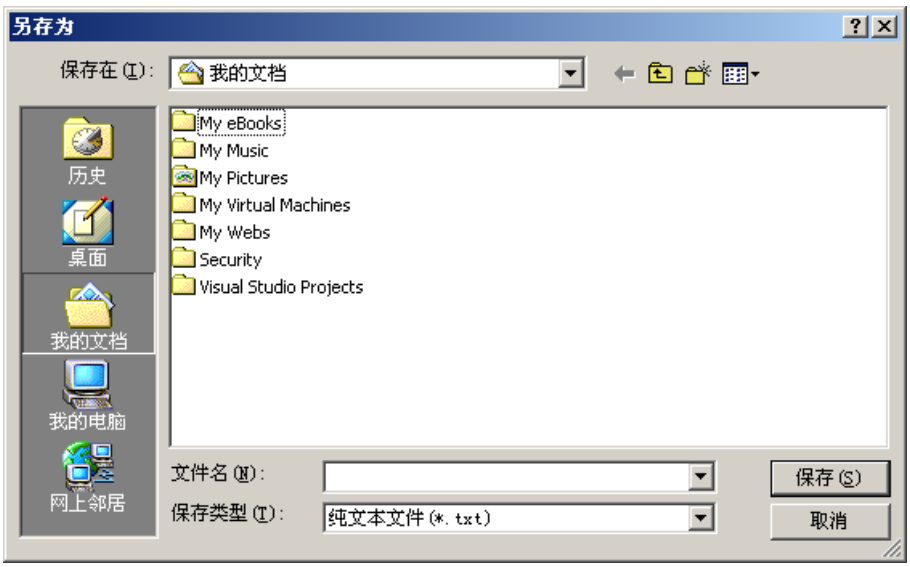


图 4.3.1B 文件另存为对话框

- 属性 Filter：字符串类型，选择在对话框中显示的文件类型。属性 Filter 有多项，中间用|分开，每两项是一组，每组的第一项将出现在对话框保存类型(T)下拉列表编辑框的下拉列表中(见图 4.3.1A)，供用户选择，第二项表示如第一项被选中，对话框实际列出的文件。例如 Filter="纯文本文件(*.txt)|*.txt|所有文件(*.*)|*.*"，表示打开对话框，对话框的文件类型(T)下拉列表编辑框的下拉列表有两项：纯文本文件(*.txt)和所有文件(*.*)，供用户选择。如果从文件类型下拉列表编辑框的下拉列表中选中"纯文本文件(*.txt)"，表示打开对话框，只列出所有扩展名为.txt 的文件，如果选中"

所有文件(*.*)"，表示打开对话框，将列出所有文件。

- 属性 FilterIndex：表示打开对话框后，对话框的文件类型(T)下拉列表编辑框的下拉列表中首先被选中的项的索引号。可以在设计阶段在属性窗口修改属性 FilterIndex 和 Filter，也可在程序中用下列语句修改：openFileDialog1.Filter="纯文本文件(*.txt)|*.txt|所有文件(*.*)|*.*",openFileDialog1.FilterIndex=1。
- 属性 FileName：用户选取的文件的路径和文件名。
- 属性 InitialDirectory：打开对话框首先显示该属性指定的文件夹中的文件。
- 属性 DefaultExt：如果用户未指定扩展名，自动增加属性指定的文件扩展名。
- 方法 ShowDialog()：打开对话框，根据方法的返回值确定用户单击了那个按钮，如返回 DialogResult.Cancle，用户单击了忽略按钮，如返回 DialogResult.OK，用户单击了打开或保存按钮。

4.3.2存取文件功能的实现

(4) 把 OpenFileDialog 和 SaveFileDialog 控件放到窗体中。属性 Name 分别是 openFileDialog1 和 saveFileDialog1。

(5) 增加顶级菜单项：文件，为其弹出菜单增加菜单项：新建、打开...、保存...、另存为...、退出。修改 Name 属性分别为：mainMenuFile、menuItemFileNew、menuItemFileOpen、menuItemFileSave、menuItemFileSaveAs、menuItemFileExit。

(6) 为 Form1 类增加 string 类型变量记录当前编辑的文件名：string s_FileName=""，如果为空，表示还未记录文件名，即编辑的文件还没有名字。当单击菜单项保存，保存文件时，必须请用户输入文件名。

(7) 为新建菜单项增加事件处理函数如下：

```
private void menuItemFileNew_Click(object sender, System.EventArgs e)
{   richTextBox1.Text="";//或richTextBox1.Clear();
    s_FileName="";//新建文件没有文件名。
}
```

(8) 为打开文件菜单项增加事件处理函数如下：

```
private void menuItemFileOpen_Click(object sender, System.EventArgs e)
{   if(openFileDialog1.ShowDialog()==DialogResult.OK)
    {   s_FileName=openFileDialog1.FileName;
        richTextBox1.LoadFile(openFileDialog1.FileName,
                              RichTextBoxStreamType.PlainText);
    }
}
```

(9) 为另存为菜单项增加事件处理函数如下：

```
private void menuItemFileSaveAs_Click(object sender, System.EventArgs e)
{   if(saveFileDialog1.ShowDialog()==DialogResult.OK)
    {   s_FileName=saveFileDialog1.FileName;
        richTextBox1.SaveFile(saveFileDialog1.FileName,
                              RichTextBoxStreamType.PlainText);
    }//注意存取文件类型应一致。
}
```

(10)为保存文件菜单项增加事件处理处理函数如下：

```
private void menuItemSaveFile_Click(object sender, System.EventArgs e)
{   if(s_FileName.Length!=0)
      richTextBox1.SaveFile(s_FileName,RichTextBoxStreamType.PlainText);
    else
      menuItemFileSaveAs_Click(sender,e);//调用另存为菜单项事件处理函数
}
```

(11)把 SaveFileDialog 控件放到窗体中，将自动创建控件对象，其生命周期等于窗体生命周期，将长期占用存储空间。实际上 SaveFileDialog 控件对象只在存文件菜单项事件处理函数中有用，其它时间无用。为了节约存储空间，可以在存文件菜单项事件处理函数中建立 SaveFileDialog 控件对象，退出该事件处理函数时，自动释放该对象。修改另存为菜单项事件处理函数如下(首先删除增加的控件 SaveFileDialog)：

```
private void menuItemFileSaveAs_Click(object sender, System.EventArgs e)
{   SaveFileDialog saveFileDialog1=new SaveFileDialog();
    saveFileDialog1.Filter="纯文本文件(*.txt)|*.txt|所有文件(*.*)|*.*";
    saveFileDialog1.FilterIndex=1;
    if(saveFileDialog1.ShowDialog()==DialogResult.OK)
    {   s_FileName=saveFileDialog1.FileName;
        richTextBox1.SaveFile(saveFileDialog1.FileName,
                                    RichTextBoxStreamType.PlainText);
    }//也可以用此方法修改打开文件菜单项事件处理函数。
}
```

(12)为退出菜单项增加事件处理函数如下：

```
private void menuItemExit_Click(object sender, System.EventArgs e)
{   Close();}
```

(13)编译，运行，可以存取文件。

4.4 修改字体属性

为修改字体属性，首先打开字体对话框 FontDialog，选择指定字体。可以按两种方式修改字体，如果未选中字符，表示以后键入的字符将按选定字体输入。如果选中字符，则仅修改选定字符的字体。修改字符颜色也根据同样原则。

4.4.1 FontDialog 控件属性和方法

用户可以用 FontDialog 对话框选定指定字体，FontDialog 控件和 OpenDialog 控件的属性和方法基本相同，这里只介绍不同部分。属性 Font：用户用 FontDialog 对话框选定的字体。FontDialog 对话框显示效果如图 4.3.1。

4.4.2 修改字体属性的实现方法

(14)放 FontDialog 控件到窗体，属性 Name=fontDialog1。增加顶级菜单项：格式，为格式顶级菜单项的弹出菜单增加菜单项：字体，属性 Name 分别为 mainMenuModel 和

menuItemModelFont，为字体菜单项增加事件处理函数如下：

```
private void menuItemModelFont_Click(object sender, System.EventArgs e)
{   if(fontDialog1.ShowDialog()==DialogResult.OK)
        richTextBox1.SelectionFont=fontDialog1.Font;
}
```

(15)编译，运行，在选中字符和不选中字符两种情况下，用字体菜单项修改字体，看是否能实现写字板中同样的功能。

图 4.3.1 字体对话框

4.5 实现 About 对话框

前边介绍的 SaveDialog、OpenDialog 和 FontDialog 都是类库中预先定义的对话框，本节介绍如何创建满足一定要求的自制对话框。对话框其实就是窗体，其基类和主窗体一样，是 System.Windows.Forms.Form。只是一般对话框只有关闭按钮，没有最大化和最小化按钮，对话框的边界是固定的，不能改变。设计自己的对话框是经常遇到的工作。

(16)选择菜单项项目/添加 Windows 窗体，弹出对话框(见图 4.5)，在模板(T)编辑框中选择 Windows 窗体，在名称栏(N)编辑框中输入窗体文件名称：formAbout.cs，单击打开按钮，可以见到一个新窗体。从文件 formAbout.cs 可以看到新建窗体类名也为 formAbout。

(17)修改 formAbout 属性 StartPosition=CenterParent，表示打开对话框时，对话框在父窗口的中间。修改属性 MaximizeBox=False，MinimizeBox=False，表示没有最大化和最小化按钮，既不能最大化和最小化。属性 FormBorderStyle=FixedDialog，窗口不能修改大小。属性 Text="关于记事本"。可以在窗体中增加各种控件，例如，小图标，Label 控件等。本例仅增加 Label 控件表示版权信息，其属性 Text="版权所有"。一个按钮，属性 Text="确定"，按钮单击事件处理函数如下：

```
private void button1_Click(object sender,System.EventArgs e)
{Close();}
```

(18)为 Form1 窗体增加顶级菜单项：帮助，为帮助顶级菜单项弹出菜单增加菜单项：关于…，属性 Name 为 menuItemAbout。关于…菜单项单击事件处理函数如下：

```
private void menuItemAbout_Click(object sender, System.EventArgs e)
{   formAbout AboutDialog=new formAbout();
    AboutDialog.ShowDialog(this);
}//注意不能使用 Show()函数
```

(19)编译，运行，单击关于…菜单项，将出现一个 formAbout 对话框(如右图)，并且不关闭此对话框，不能回到主窗口，一般把这样的对话框叫做模式对话框。

图 4.5

4.6 实现文本编辑器查找替换功能

本节首先介绍模式对话框和非模式对话框的概念。并用非模式对话框实现文本编辑器程序的查找和替换功能。

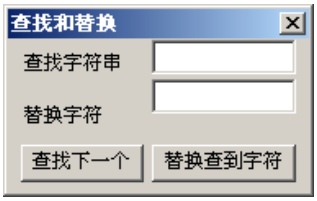
4.6.1 模式对话框和非模式对话框

模式对话框和非模式对话框的区别是：打开模式对话框后，只有关闭该模式对话框，才能转到其他窗口，例如前边讲到的 SaveDialog 和 OpenDialog 都是典型的模式对话框。而打开非模式对话框后，不必退出该模式对话框，就可以转到其他窗口，例如查找和替换对话框都是典型的非模式对话框。两类对话框本质上都是窗体，是 System.Windows.Forms.Form 类的派生类，只是打开时使用的方法不一样，打开模式对话框，使用方法 ShowDialog()，而打开非模式对话框，使用方法 Show()。文本编辑器程序中，查找和替换对话框一般是非模式对话框。

4.6.2 写字板查找替换功能的实现

(20)建立查找替换对话框。对话框其实就是窗体，其基类是 System.Windows.Forms.Form。选择菜单项项目/添加 Windows 窗体，弹出对话框(如图 4.5)，选择 Windows 窗体，在名称栏输入窗体文件名称：formFindReplace.cs，单击打开按钮，可以见到一个新窗体。其属性 Name=formFindReplace。

(21)修改 formFindReplace 窗体属性 StartPosition=CenterParent，表示打开对话框时，对话框在父窗口的中间。修改属性 MaximizeBox=False，MinimizeBox=False，表示没有最大化和最小化按钮，既不能最大化和最小化。FormBorderStyle=FixedDialog，窗口不能修改大小。属性 Text="查找和替换"。在窗体中增加两个 Label 控件，属性 Text 分别为"查找字符串"和"替换字符串"。两个 TextBox 控件，属性 Text=""。两个按钮，属性 Text 分别为"查找下一个"和"替换查到字符"。修改属性 TopMost=true，使该窗口打开时总在其它窗体的前边。对话框界面如右图。



(22)为 formFindReplace 窗体增加变量:Form1 MainForm1;

(23)修改 formFindReplace 类构造函数如下(阴影部分是所做的修改)：

```
public formAbout(Form1 form1)//增加参数
{
//Windows窗体设计器支持所必需的
    InitializeComponent();
//TODO:在InitializeComponent调用后添加任何构造函数代码
    MainForm1=form1;//新增语句,这里Form1是主窗体的属性Name的值
}//有了 Form1，可以在 formFindReplace 窗体中调用主窗体的公有方法
```

(24)为主窗体 Form1 增加方法如下，该方法将被 formFindReplace 窗体类调用。

```
public void FindRichTextBoxString(string FindString)
{}    //以后步骤将在此方法中增加查找语句
```

(25)formFindReplace 窗体中查找下一个按钮单击事件处理函数如下：

```
private void buttonFind_Click(object sender, System.EventArgs e)
{   if(textBox1.Text.Length!=0)//如果查找字符串不为空,调用主窗体查找方法
        MainForm1.FindRichTextBoxString(textBox1.Text);//上步增加的方法
    else
        MessageBox.Show("查找字符串不能为空","提示",MessageBoxButtons.OK);
}//MessageBox 时对话框，使用方法见 4.7.1 节
```

(26)为主窗体 Form1 增加方法如下，该方法将被 formFindReplace 窗体类调用。

```
public void ReplaceRichTextBoxString(string ReplaceString)
{}     //以后步骤将在此方法中增加替换语句
```

(27)为替换查到字符按钮单击事件增加事件处理函数如下：

```
private void buttonReplace_Click(object sender, System.EventArgs e)
{ if(textBox2.Text.Length!=0)//如果查找字符串不为空,调用主窗体替换方法
    MainForm1.ReplaceRichTextBoxString(textBox1.Text,textBox2.Text);
  else//方法MainForm1.ReplaceRichTextBoxString见(26)中定义
    MessageBox.Show("替换字符串不能为空","提示", MessageBoxButtons.OK);
}
```

(28)为 Form1 窗体增加变量：int FindPostion=0，记录查找位置。

(29)为 Form1 窗体顶级菜单项编辑的弹出菜单增加菜单项：查找和替换。为查找和替换菜单项单击事件增加事件处理函数如下：

```
private void menuItemFindReplace_Click(object sender, System.EventArgs e)
{   FindPostion=0;
    formAbout FindReplaceDialog=new formAbout(this);//注意this
    FindReplaceDialog.Show();//打开非模式对话框使用Show()方法
}
```

(30)为在前边定义的 Form1 主窗体的 FindRichTextBoxString 方法增加语句如下：

```
public void FindRichTextBoxString(string FindString)
{   if(FindPostion>=richTextBox1.Text.Length)//已查到文本底部
    {   MessageBox.Show("已到文本底部,再次查找将从文本开始处查找",
                                              "提示",MessageBoxButtons.OK);
        FindPostion=0;
        return;
    }//下边语句进行查找，返回找到的位置，返回-1，表示未找到，参数1是要找的字符串
    //参数2是查找的开始位置，参数3是查找的一些选项，如大小写是否匹配，查找方向等
    FindPostion=richTextBox1.Find(FindString,
                            FindPostion,RichTextBoxFinds.MatchCase);
    if(FindPostion==-1)//如果未找到
    {   MessageBox.Show("已到文本底部,再次查找将从文本开始处查找",
                                              "提示", MessageBoxButtons.OK);
        FindPostion=0;//下次查找的开始位置
    }
    else//已找到
    {   richTextBox1.Focus();//主窗体成为注视窗口
        FindPostion+=FindString.Length;
    }//下次查找的开始位置在此次找到字符串之后
}
```

(31)为在前边定义的 Form1 主窗体的 ReplaceRichTextBoxString 方法增加语句如下：

```
public void ReplaceRichTextBoxString(string ReplaceString)
{   if(richTextBox1.SelectedText.Length!=0)//如果选取了字符串
    richTextBox1.SelectedText=ReplaceString;//替换被选的字符串
}
```

(32)编译，运行，输入若干字符，选中菜单项：编辑/查找和替换，打开对话框，注意该对话框可以在不关闭的情况下，转到主窗体，并且总是在其它窗体的前边，因此它是一个典型的非模式对话框。在对话框中输入查找和替换的字符，单击标题为查找下一个的按钮，可以找到所选字符，并被选中，单击标题为替换所选字符按钮，可以看到查找到的字符被替换。运行效果如右图：

4.7 提示用户保存修改的文件

用户在新建文本，打开其他文本或者退出文本编辑器时，如果编辑内容发生了改变，应提示用户是否保存已修改的文本内容。因此就需要在用户关闭当前文件前，弹出提示对话框，提醒用户是否保存当前文件。本节实现此功能。

4.7.1 对话框 MessageBox

使用 MessageBox 可以打开一个对话框，用法如下：

```
MessageBox.Show(this,"要保存当前更改吗？","保存更改吗？",
                MessageBoxButtons.YesNoCancel,MessageBoxIcon.Question);
```

第一个参数是父窗口，第二个参数是提示信息，第三个参数是标题栏的内容，第四个参数是有那些按钮，此例有 YES,NO,CANCEL 按钮，还可以使用 AbortRetryIgnore(中止、重试和忽略按钮)、OK(确定按钮)、OKCancel(确定和取消按钮)、RetryCance(重试和忽略按钮)、YesNo(是和否按钮)等选项。第五个参数是使用那一个图标，此例是一个问号图标，还可以是 Asterisk、Error、Exclamation、Hand、Stop、Warning 等图标，如为 None 则无图标。返回值是 System.Windows.Forms.DialogResult 变量，代表用户按了那一个按钮。如果返回值是 System.Windows.Forms.DialogResult.Yes，则表示按了 YES 键，表示要存修改的文件。如果返回值是 System.Windows.Forms.DialogResult.Cancel，按 Cancel 键，表示忽略此次操作。如果返回值是 System.Windows.Forms.DialogResult.No，则表示按了 No 键，表示不存修改的文件。以上设计的对话框 MessageBox 如下图：

4.7.2 提示用户保存修改的文件的实现

(33)为 Form1 类增加一个 bool 变量 bSave=false 作为标记，用来跟踪 RichTextBox 中文本内容改变的情况。在程序开始运行、建立和打开一个新文件时，bSave=false，表示不必保存当前文本。RichTextBox 控件有一个 TextChanged 事件，当文本发生改变的时候，这个事件就会被激活，在该事件处理函数中，使 bSave=true。

(34)首先增加一个函数，其功能是判断是否需要将已修改的文件存盘，之所以要增加这个函数是因为有三处要用到此函数。该函数返回 true，表示继续操作，该函数返回 false，表示忽略此次操作，该函数定义如下：

```
public bool IfSaveOldFile()
{ bool ReturnValue=true;
  if(bSave)
  { System.Windows.Forms.DialogResult dr;
    dr=MessageBox.Show(this,"要保存当前更改吗？","保存更改吗？",
```

```
                    MessageBoxButtons.YesNoCancel,MessageBoxIcon.Question);
    switch(dr)
    {  case System.Windows.Forms.DialogResult.Yes://单击了yes按钮，保存修改
          bSave=false;
          if(s_FileName.Length!=0)
            richTextBox1.SaveFile(s_FileName,RichTextBoxStreamType.PlainText);
          else
          { SaveFileDialog saveFileDialog1=new SaveFileDialog();
            saveFileDialog1.Filter="纯文本文件(*.txt)|*.txt|所有文件(*.*)|*.*";
            saveFileDialog1.FilterIndex=1;
            if(saveFileDialog1.ShowDialog()==DialogResult.OK)
            { s_FileName=saveFileDialog1.FileName;
              richTextBox1.SaveFile(saveFileDialog1.FileName,
                                    RichTextBoxStreamType.PlainText);
            }
          }
          ReturnValue=true;
          break;
       case System.Windows.Forms.DialogResult.No://单击了no按钮，不保存
          bSave=false;
          ReturnValue=true;
          break;
       case System.Windows.Forms.DialogResult.Cancel://单击了Cancel按钮
          ReturnValue=false;
          break;
    }
  }
  return ReturnValue;
}
```

(35)在新建和打开菜单项的事件处理函数的头部增加如下语句：

```
if(!IfSaveOldFile())//如果忽略，退出。
   return;
```

(36)修改存文件菜单项单击事件处理函数如下：

```
private void menuItemSaveFile_Click(object sender, System.EventArgs e)
{   if(s_FileName.Length!=0)
    {   bSave=false;//阴影为增加的语句
        richTextBox1.SaveFile(s_FileName,RichTextBoxStreamType.PlainText);
    }
    else
        menuItemSaveAs_Click(sender,e);
}
```

(37)修改另存为菜单项单击事件处理函数如下：

```
private void menuItemSaveAs_Click(object sender, System.EventArgs e)
{   SaveFileDialog saveFileDialog1=new SaveFileDialog();
```

```
    saveFileDialog1.Filter="纯文本文件(*.txt)|*.txt|所有文件(*.*)|*.*";
    saveFileDialog1.FilterIndex=1;
    if(saveFileDialog1.ShowDialog()==DialogResult.OK)
    {   s_FileName=saveFileDialog1.FileName;
        richTextBox1.SaveFile(saveFileDialog1.FileName,
                                RichTextBoxStreamType.PlainText);
        bSave=false;//阴影为增加的语句
    }
}
```

(38)为 RichTextBox 控件 TextChanged 事件增加事件处理函数如下：

```
private void richTextBox1_TextChanged(object sender, System.EventArgs e)
{   bSave=true;}
```

(39)为 Form1 窗体 Closing 事件是在关闭窗口之前发送的事件，此时，窗体中的控件还存在，还可以保存修改的内容，也可以不退出。增加它的事件处理函数如下：

```
private void Form1_Closing(object sender,System.ComponentModel.CancelEventArgs e)
{   if(!IfSaveOldFile())
        e.Cancel=true;//不退出
}
```

(40)编译，运行，键入若干字符，选中菜单项新建或打开，或退出，将看到提示信息，问是否保存修改的文件。有三种选择：存文件，不存文件，忽略此次操作，试验单击不同按钮的效果。

4.8 打印和打印预览

打印和打印预览是一个编辑器必须具有的功能，本节介绍实现打印和打印预览的方法。一般要实现如下菜单项：打印、打印预览、页面设置。

4.8.1 PrintDocument 类

PrintDocument 组件是用于完成打印的类，其常用属性、方法和事件如下：

- 属性 DocumentName：字符串类型，记录打印文档时显示的文档名（例如，在打印状态对话框或打印机队列中显示）。
- 方法 Print：开始文档的打印。
- 事件 BeginPrint：在调用 Print 方法后，在打印文档的第一页之前发生。
- 事件 PrintPage：需要打印新的一页时发生。
- 事件 EndPrint：在文档的最后一页打印后发生。

若要打印，首先创建 PrintDocument 组件的对象。然后使用页面设置对话框 PageSetupDialog 设置页面打印方式，这些设置作为要打印的所有页的默认设置。使用打印对话框 PrintDialog 设置对文档进行打印的打印机的参数。在打开两个对话框前，首先设置对话框的属性 Document 为指定的 PrintDocument 类对象，修改的设置将保存到 PrintDocument 组件对象中。第三步是调用 PrintDocument.Print 方法来实际打印文档。当调用该方法后，引发下列事件：BeginPrint、PrintPage、EndPrint。其中每打印一页都引发 PrintPage 事件，打印多页，要多次引发 PrintPage 事件。完成一次打印，可以引发一个

或多个 PrintPage 事件。

程序员应为这 3 个事件编写事件处理函数。BeginPrint 事件处理函数进行打印初始化，一般设置在打印时所有页的相同属性或共用的资源，例如所有页共同使用的字体、建立要打印的文件流等。PrintPage 事件处理函数负责打印一页数据。EndPrint 事件处理函数进行打印善后工作。这些处理函数的第 2 个参数 System.Drawing.Printing.PrintEventArgs e 提供了一些附加信息，主要有：

- e.Cancel：布尔变量，设置为 true，将取消这次打印作业。
- e.Graphics：所使用的打印机的设备环境，参见第五章。
- e.HasMorePages：布尔变量。PrintPage 事件处理函数打印一页后，仍有数据未打印，退出事件处理函数前设置 HasMorePages=true，退出 PrintPage 事件处理函数后，将再次引发 PrintPage 事件，打印下一页。
- e.MarginBounds：打印区域的大小，是 Rectangle 结构，元素包括左上角坐标：Left 和 Top，宽和高：Width 和 Height。单位为 1/100 英寸。
- e.MarginBounds：打印纸的大小，是 Rectangle 结构。单位为 1/100 英寸。
- e.PageSettings：PageSettings 类对象，包含用对话框 PageSetupDialog 设置的页面打印方式的全部信息。可用帮助查看 PageSettings 类的属性。

下边为这 3 个事件编写事件处理函数，具体步骤如下：

(41)在最后一个 using 语句之后增加语句：

```
using System.IO;
using System.Drawing.Printing;
```

(42)本例打印或预览 RichTextBox 中的内容，增加变量：StringReader streamToPrint=null。如果打印或预览文件，改为：StreamReader streamToPrint，流的概念参见第六章。增加打印使用的字体的变量：Font printFont。

(43)放 PrintDocument 控件到窗体，属性 name 为 printDocument1。

(44)为 printDocument1 增加 BeginPrint 事件处理函数如下：

```
private void printDocument1_BeginPrint(object sender,
                                       System.Drawing.Printing.PrintEventArgs e)
{   printFont=richTextBox1.Font;//打印使用的字体
    streamToPrint=new StringReader(richTextBox1.Text);//打印richTextBox1.Text
}//如预览文件改为：streamToPrint=new StreamReader("文件的路径及文件名");
```

(45)printDocument1 的 PrintPage 事件处理函数如下。streamToPrint.ReadLine()读入一段数据，可能打印多行。本事件处理函数将此段数据打印在一行上，因此方法必须改进。

```
private void printDocument1_PrintPage(object sender,
                                      System.Drawing.Printing.PrintPageEventArgs e)
{   float linesPerPage=0;//记录每页最大行数
    float yPos=0;//记录将要打印的一行数据在垂直方向的位置
    int count=0;//记录每页已打印行数
    float leftMargin=e.MarginBounds.Left;//左边距
    float topMargin=e.MarginBounds.Top;//顶边距
    string line=null;//从RichTextBox中读取一段字符将存到line中
    //每页最大行数=一页纸打印区域的高度/一行字符的高度
    linesPerPage=e.MarginBounds.Height/printFont.GetHeight(e.Graphics);
    //如果当前页已打印行数小于每页最大行数而且读出数据不为null，继续打印
    while(count<linesPerPage&&((line=streamToPrint.ReadLine())!=null))
```

```
    {   //yPos为要打印的当前行在垂直方向上的位置
        yPos=topMargin+(count*printFont.GetHeight(e.Graphics));
        e.Graphics.DrawString(line,printFont,Brushes.Black,
                leftMargin,yPos,new StringFormat());//打印，参见第五章
        count++;//已打印行数加1
    }
    if(line!=null)//是否需要打印下一页
        e.HasMorePages=true;//需要打印下一页
    else
        e.HasMorePages=false;//不需要打印下一页
}
```

(46)为 printDocument1 增加 EndPrint 事件处理函数如下：

```
private void printDocument1_EndPrint (object sender,
                                      System.Drawing.Printing.PrintEventArgs e)
{   if(streamToPrint!=null)
        streamToPrint.Close();//释放不用的资源
}
```

4.8.2 打印设置对话框控件 PageSetupDialog

Windows 窗体的 PageSetupDialog 控件是一个页面设置对话框，用于在 Windows 应用程序中设置打印页面的详细信息，对话框的外观如图 4.8.2。

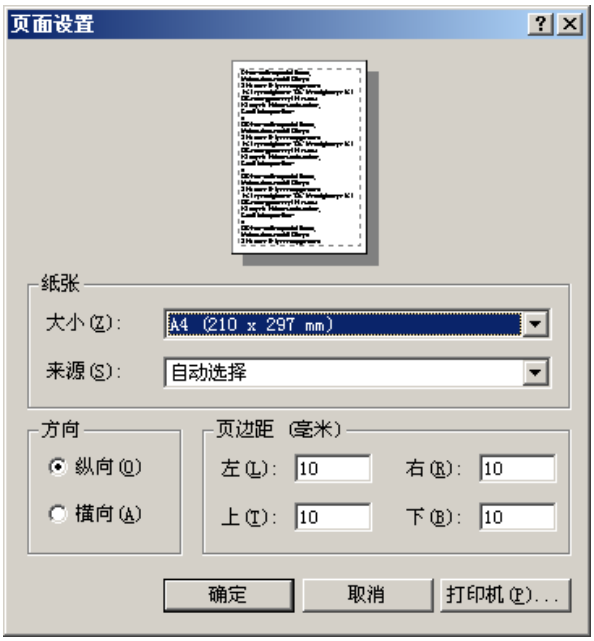


图 4.8.2

用户使用此对话框能够设置纸张大小(类型)、纸张来源、纵向与横向打印、上下左右的页边距等。在打开对话框前，首先设置其属性 Document 为指定的 PrintDocument 类对象，

用来把页面设置保存到 PrintDocument 类对象中。为文本编辑器增加页面设置功能的具体步骤如下：

(47)为文件顶级菜单项的弹出菜单增加菜单项：页面设置。

(48)放 PageSetupDialog 控件到窗体，属性 name 为 pageSetupDialog1。

(49)为页面设置菜单项增加单击事件处理函数如下：

```
private void menuItem5_Click(object sender,System.EventArgs e)
{   pageSetupDialog1.Document=printDocument1;
    pageSetupDialog1.ShowDialog();
}
```

(50)打开对话框 pageSetupDialog1 后，如果单击了确定按钮，PageSetupDialog 对话框中所做的的页面设置被保存到 PrintDocument 类对象 printDocument1 中，如果单击了取消按钮，不保存这些修改，维持原来的值。当调用 PrintDocument.Print 方法来实际打印文档时，引发 PrintPage 事件，该事件处理函数的第二个参数 e 提供了这些设置信息。

4.8.3 打印预览

用 PrintPreviewDialog 类可以在屏幕上显示 PrintDocument 的打印效果，既打印预览。实现打印预览的具体步骤如下：

(51)为文件顶级菜单项的弹出菜单增加菜单项：打印预览。

(52)放 PrintPreviewDialog 控件到窗体，属性 name 为 printPreviewDialog1。

(53)为打印预览菜单项增加单击事件处理函数如下：

```
private void menuItemPrintView_Click(object sender,System.EventArgs e)
{   printPreviewDialog1.Document=printDocument1;
    printPreviewDialog1.ShowDialog();
}
```

(54)编译，运行，输入若干字符，试验一下预览的效果，预览的效果如图 4.8.3。

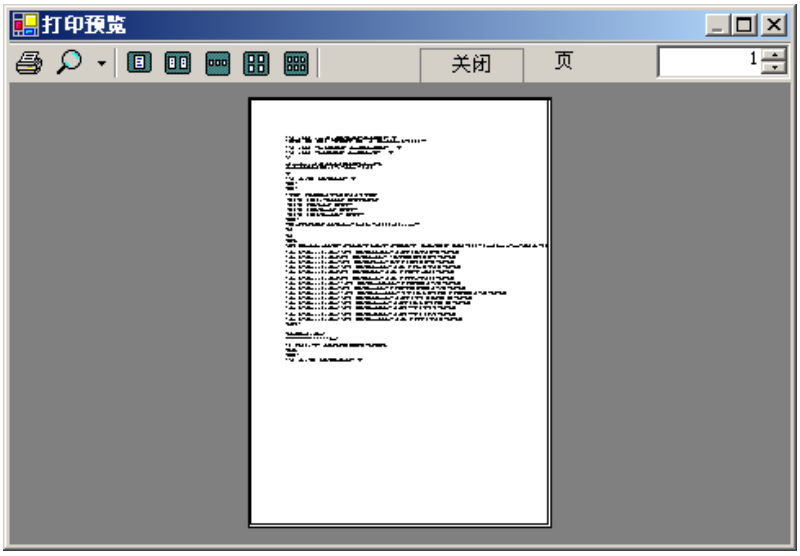


图 4.8.3

4.8.4 用打印对话框 PrintDialog 实现打印

PrintDialog 组件是类库中预先定义的对话框,用来设置对文档进行打印的打印机的参数,包括打印机名称、要打印的页(全部打印或指定页的范围)、打印的份数以及是否打印到文件等。在打开对话框前，首先设置其属性 Document 为指定的 PrintDocument 类对象，打开 PrintDialog 对话框后，修改的设置将保存到 PrintDocument 类的对象中。PrintDialog 对话框的外观如图 4.8.4。

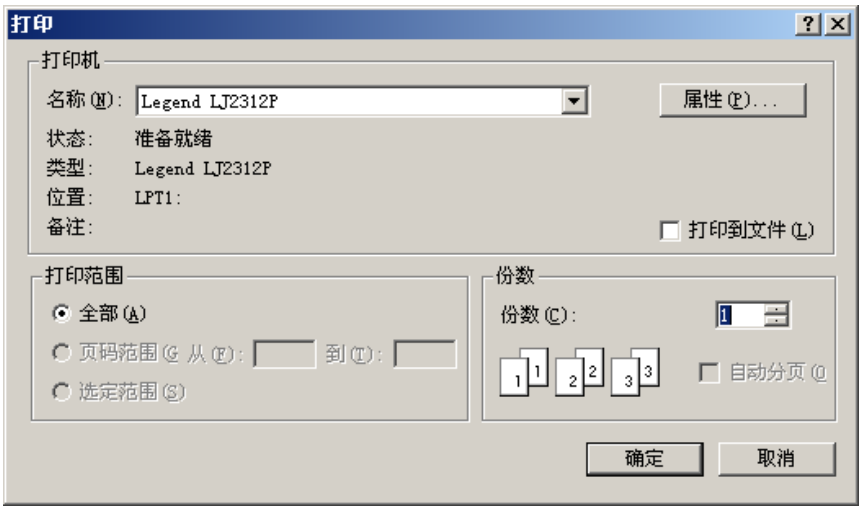


图 4.8.4

增加打印功能的具体步骤如下：

(55)放 PrintDialog 控件到窗体属性 Name=printDialog1。

(56)为文件顶级菜单项的弹出菜单增加菜单项：打印。

(57)为打印菜单项增加单击事件处理函数如下：（不能打印？）

```
private void menuItemPrint_Click(object sender, System.EventArgs e)
{   printDialog1.Document=printDocument1;
    if(printDialog1.ShowDialog(this)==DialogResult.OK)
        printDocument1.Print();
}
```

(58)编译，运行，输入若干字符，试验一下打印效果。

4.9 编写多文档界面应用程序

本节首先介绍如何建立类似 Microsoft Word 的文本编辑器，然后介绍如何建立类似 Visualstudio.Net 的编辑器那样的文本编辑器，有多个选项卡页。

4.9.1建立类似 Microsoft Word 的编辑器

建立一个类似 Microsoft Word 的编辑器，可以有多页，每页处理一个文档。多文档界面(MDI)应用程序具有一个主窗体(父窗体)，主窗体在其工作区内包含一组窗体(子窗体)。每个子窗体都是一个限制为只能在该父窗体内出现的窗体。这些子窗体通常共享父窗体界面的菜单栏、工具栏以及其他部分。创建多文当编辑器的具体步骤如下：

(1) 新建项目。修改主窗体属性 IsMdiContainer=true，表示主窗体是一个子窗体容器。

(2) 放主菜单控件 Mainmenu 到主窗体。增加顶级菜单项：文件，属性 Name=menuItemFile。为文件菜单增加菜单项：新建、打开、另存为、关闭当前窗口、退出，属性 Name 分别为 menuItemNew、menuItemOpen、menuItemSaveAs、menuItemCloseChild、menuItemExit。增加顶级菜单项：窗口，属性 Name=menuItemWindow，属性 MdiList=true,该属性将在窗口菜单下增加子窗口列表。为窗口菜单增加菜单项：水平平铺、层叠、垂直平铺，属性 Name 分别为 menuItemTileH、menuItemCascade、menuItemTileV。

(3) 创建子窗体,选择菜单项:项目/添加Windows窗体,弹出对话框(见图4.5),选择Windows窗体，在名称栏输入窗体文件名称：FormChild.cs，单击打开按钮，可以见到一个新窗体。定义新窗体的类名也为 FormChild。此窗体作为主窗体的子窗体。

(4) 放 RichTextBox1 控件到子窗体。修改属性 Dock=Fill，Text="",Modifiers=public,使 RichTextBox1 为公有成员,在主窗体可以访问 RichTextBox1。

(5) 为主窗体菜单项新文件增加单击事件处理函数如下：

```
private void menuItemNew_Click(object sender,System.EventArgs e)
{   FormChild formChild=new FormChild();
    formChild.MdiParent=this;
    formChild.Show();
}
```

(6) 把 OpenFileDialog 控件放到窗体中。单击打开文件菜单项事件处理函数如下：

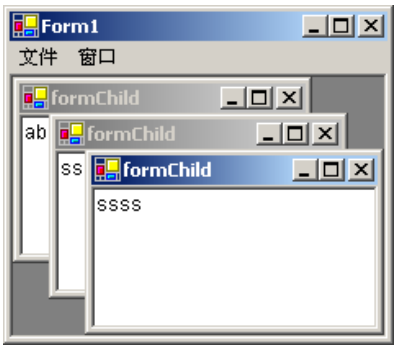
```
private void menuItemOpen_Click(object sender, System.EventArgs e)
{   if(openFileDialog1.ShowDialog(this)==DialogResult.OK)
    {   FormChild ChildForm=new FormChild();
        ChildForm.MdiParent=this;
        ChildForm.richTextBox1.LoadFile(openFileDialog1.FileName,
                                    RichTextBoxStreamType.PlainText);
        ChildForm.Show();
    }
}
```

(7) 把 SaveFileDialog 控件放到子窗体中。另存为菜单项事件处理函数如下：

```
private void menuItemChildSaveAs_Click(object sender, System.EventArgs e)
{   if(saveFileDialog1.ShowDialog(this)==DialogResult.OK)
    {   FormChild ChildForm=(FormChild)this.ActiveMdiChild;
        ChildForm.richTextBox1.SaveFile(saveFileDialog1.FileName,
                                        RichTextBoxStreamType.PlainText);
    }
}
```

(8) 为主窗体菜单项关闭当前窗口增加单击事件函数如下：

```
private void menuItemCloseChild_Click(object sender, System.EventArgs e)
{   this.ActiveMdiChild.Close();}
```
(9) 为主窗体菜单项退出增加单击事件函数如下：
```
private void menuItemExit_Click(object sender, System.EventArgs e)
{   Close();}
```
(10)为主窗体菜单项水平平铺增加单击事件函数如下：
```
private void menuItemTileH_Click(object sender, System.EventArgs e)
{   this.LayoutMdi(MdiLayout.TileHorizontal);}
```
(11)为主窗体菜单项层叠增加单击事件函数如下：
```
private void menuItemCascade_Click_1(object sender, System.EventArgs e)
{   this.LayoutMdi(MdiLayout.Cascade);}
```
(12)为主窗体菜单项垂直平铺增加单击事件函数如下：
```
private void menuItemTileV_Click(object sender, System.EventArgs e)
{   this.LayoutMdi(MdiLayout. TileVertical);}
```
(13)运行，运行效果如下，子窗体为层叠排列。



4.9.2主窗口和子窗口的菜单的融合

在许多多文档编辑器应用程序中，在没有子窗体打开时，菜单比较简单，而有子窗体打开后，菜单增多。实现这种功能一般是在主窗体中创建一个简单菜单，子窗体没打开时，只显示这个简单菜单。在子窗体中也创建一个菜单，包含主窗体菜单中没有的菜单项。打开子窗体后，子窗体的菜单和主窗体菜单合并成为一个菜单，这个功能叫做主窗口和子窗口的菜单的融合。创建具有这种功能的多文档编辑器应用程序可以按下列步骤：

(1) 新建项目。修改主窗口属性 IsMdiContainer 为 true。

(2) 把 Mainmenu 控件放到主窗体中。增加顶级菜单项：文件。其属性 MergeType=MergeItems，表示打开子窗体后，主窗体和子窗体中属性 MergeOrder 相同的顶级菜单项的弹出菜单中的菜单项合并为一个弹出菜单。属性 MergeOrder=0。子窗体的顶级菜单项文件的属性 MergeType 也应为 MergeItems，MergeOrder 属性也应为 0，这样打开子窗口后，才能合并为一个弹出菜单。属性 Name=menuItemFile。为文件菜单增加菜单项：新建、打开、退出,属性 Name 分别为 menuItemNew、menuItemOpen、menuItemExit，属性 MergeType 都为 Add，属性 MergeOrder 依次为 1、2、6，目的是打开子窗口后，在新建和打开菜单项后加入子窗口菜单栏中的文件菜单的另存为菜单项。增加菜单：帮助，其属性 MergeType=Add，属性 MergeOrder=7，属性 Name=menuItemHelp。为帮助菜单增加菜单

项：关于…，属性 Name=menuItemAbout。其余菜单在子窗口中实现。注意属性 MergeOrder 分别为 0、7，打开子窗口后，子窗口中的菜单将按顺序插入到主窗口的菜单中，例如，子窗口有菜单：编辑，其属性 MergeOrder=3，合并后，菜单排列顺序为：文件、编辑、帮助。

(3) 创建子窗体，选择菜单项：项目/添加 Windows 窗体，弹出对话框，选择 Windows 窗体，在名称栏输入窗体文件名称：formChild.cs，单击打开按钮，可以见到一个新窗体。定义新窗体的类名也为 formChild。

(4) 为 formChild 窗体增加变量：Form1 MainForm1;

(5) 修改 formChild 类构造函数如下(阴影部分是所做的修改)：

```
public formChild(Form1 form1)//增加参数
{
//Windows 窗体设计器支持所必需的
    InitializeComponent();
//TODO:在 InitializeComponent 调用后添加任何构造函数代码
    MainForm1=form1;//新增语句,这里 Form1 是主窗体的属性 Name 的值
}//有了 Form1，可以在 formChild 窗体中调用主窗体的公有方法
```

(6) 把 Mainmenu 控件放到子窗体中。增加顶级菜单项：文件，其属性 MergeType=MergeItems，属性 MergeOrder=0。为文件顶级菜单项弹出菜单增加菜单项：另存为…，属性 MergeType=Add，属性 MergeOrder=3，菜单合并后，另存为…菜单项将出现在主窗口文件菜单的新建和打开菜单项之后。增加菜单项：关闭当前文件，属性 MergeType=Add，属性 MergeOrder=4。

(7) 增加顶级菜单项：编辑，其属性 MergeType=Add，属性 MergeOrder=3。注意属性 MergeOrder=3，菜单合并后，编辑菜单将出现出现菜单文件之后。为编辑顶级菜单项弹出菜单增加菜单项：拷贝、剪贴、粘贴。

(8) 增加顶级菜单项：窗口，其属性 MergeType=Add，属性 MergeOrder=6，属性 Name=menuItemWindow，属性 MdiList=true，该属性将在窗口菜单下增加子窗口列表。为窗口菜单增加菜单项：层叠，属性 Name 为 menuItemCascade。

(9) 放 RichTextBox1 控件到子窗体。修改属性 Dock=Fill，Text=””，属性 Modifiers=public，使 RichTextBox1 为公有成员，在主窗体可以访问 RichTextBox1。

(10)为主窗体菜单项新文件增加单击事件函数如下：

```
private void menuItemNew_Click(object sender, System.EventArgs e)
{   formChild ChildForm=new formChild(this);
    ChildForm.MdiParent=this;
    ChildForm.Show();
}
```

(11)把 OpenFileDialog 控件放到主窗体中。单击打开文件菜单项事件处理函数如下：

```
private void menuItemOpen_Click(object sender, System.EventArgs e)
{   if(openFileDialog1.ShowDialog(this)==DialogResult.OK)
    {   formChild ChildForm=new formChild(this);
        ChildForm.MdiParent=this;
        ChildForm.richTextBox1.LoadFile(openFileDialog1.FileName);
        ChildForm.Show();
    }
}
```

(12)为主窗体菜单项退出增加单击事件处理函数如下：

```
private void menuItemExit_Click(object sender, System.EventArgs e)
{   Close();}
```

(13)为子窗体菜单项层叠增加单击事件处理函数如下：

```
private void menuItemCascade_Click_1(object sender, System.EventArgs e)
{   MainForm1.LayoutMdi(MdiLayout.Cascade);}
```

(14)把 SaveFileDialog 控件放到子窗体中。为子窗体菜单项另存为增加单击事件函数如下：

```
private void menuItemChildSaveAs_Click(object sender, System.EventArgs e)
{   if(saveFileDialog1.ShowDialog(this)==DialogResult.OK)
    {   richTextBox1.SaveFile(saveFileDialog1.FileName,
                              RichTextBoxStreamType.PlainText);
    }
}
```

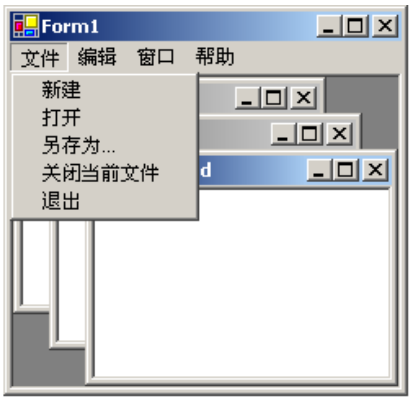
(15)为子窗体菜单项关闭当前窗口增加单击事件函数如下：

```
private void menuItemCloseChild_Click(object sender, System.EventArgs e)
{   Close();}
```

(16)为子窗体菜单项拷贝、剪贴和粘贴增加单击事件函数如下：

语句分别为：richTextBox1.Cut();richTextBox1.Copy();richTextBox1.Paste();

(17)运行效果如下图。



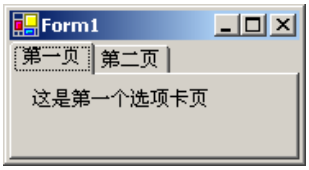
4.9.3建立类似 Visualstudio.Net 的编辑器

Visualstudio.Net 的编辑器有多个选项卡页，可以编辑多个文件。建立选项卡页数固定，每选项卡页显示一行文本，类似 Visualstudio.Net 的编辑器的文本编辑器的具体实现步骤如下：

(1) 新建项目。放 TabControl 控件到子窗体。修改属性 Dock=Fill。

(2) 单击 TabControl 属性 TabPages 后按钮，打开 TabPage 集合编辑器，单击添加按钮，增加 1 个选项卡页。修改属性 Text 分别为：第一页，第二页。如图 4.9.3。

(3) 选中第一页，可以在页中放置控件，例如放置 Label 控件，属性 Text=”这是第一个选项卡页”。同样在第二页中也放置 Label 控件，属性 Text=”这是第二个选项卡页”。如果放置 RichTextBox



控件，可以做成多文档编辑器。

(4) 运行，可以看到多页，单击每页的标题，可以转换选项卡页。运行效果如右图：

图 4.9.3

如可以有多个选项卡页，每选项卡页处理一个文档，并能动态增加新选项卡页，关闭当前选项卡页。实现步骤如下：

(1) 新建项目。放 TabControl 控件到子窗体。修改属性 Dock=Fill。

(2) 把 Mainmenu 控件放到主窗体中。增加顶级菜单项：文件，为其弹出菜单增加 4 个菜单项：新页、关闭当前页、打开、另存为。属性 Name 分别为：menuItemFile、menuItemFileNew、menuItemFileClose、menuItemFileOpen、menuItemFileSaveAs。

(3) 增加一个新方法 MakeNewTbpage()如下：

```
private object  MakeNewTbpage()
{   //增加选项卡页TabPage
    TabPage tabPage1=new TabPage();
    tabControl1.Controls.Add(tabPage1);//将tabPage1放到tabControl1中
    tabPage1.Location=new Point(4, 21);
    tabPage1.Size=new Size(284, 248);
    tabPage1.Text="第"+tabPage1.TabIndex.ToString()+"页";
    //增加RichTextBox
    RichTextBox richTextBox1=new RichTextBox();
    richTextBox1.Dock=DockStyle.Fill;
    richTextBox1.Size=new Size(284, 248);
    richTextBox1.Text="";
    tabPage1.Controls.Add(richTextBox1);//将richTextBox1放到tabPage1中
    return (object)richTextBox1;
}
```

(4) 为菜单项新页增加事件处理函数如下：

```
private void menuItemFileNew_Click(object sender, System.EventArgs e)
{ MakeNewTbpage();}
```

(5) 为菜单项关闭当前页增加事件处理函数如下：

```
private void menuItemFileClose_Click(object sender, System.EventArgs e)
{   TabPage tabPage1=tabControl1.SelectedTab;//得到当前选定的选项卡页
    tabControl1.Controls.Remove(tabPage1);//从tabControl1中移走该页
    //得到当前选定的选项卡页中第0个控件，即RichTextBox控件
    RichTextBox richTextBox1=(RichTextBox)tabPage1.Controls[0];
    if(richTextBox1!=null)
        richTextBox1.Dispose();//删除当前选定选项卡页中RichTextBox控件对象
    if(tabPage1!=null)
        tabPage1.Dispose();//删除当前选定的选项卡页
}
```

(6) 把 OpenFileDialog 控件放到子窗体中。为菜单项打开增加事件处理函数如下：

```
private void menuItemFileOpen_Click(object sender, System.EventArgs e)
{   if(openFileDialog1.ShowDialog()==DialogResult.OK)
    {   RichTextBox richTextBox1=(RichTextBox)MakeNewTbpage();
        richTextBox1.LoadFile(openFileDialog1.FileName,
                                    RichTextBoxStreamType.PlainText);
    }
}
```

(7) 把 SaveFileDialog 控件放到子窗体中。为菜单项另存为增加事件处理函数如下：

```
private void menuItemFileSaveAs_Click(object sender, System.EventArgs e)
{   if(saveFileDialog1.ShowDialog()==DialogResult.OK)
    {   TabPage tabPage1=tabControl1.SelectedTab;
        RichTextBox richTextBox1=(RichTextBox)tabPage1.Controls[0];
        richTextBox1.SaveFile(saveFileDialog1.FileName,
                                    RichTextBoxStreamType.PlainText);
    }
}
```

(8) 编译，运行，建立新文件，关闭当前选项卡页，打开新文件，存文件，开是否正常。

习题

(1) RichTextBox 控件 Lines 属性记录控件中所有文本的字符串数组，每两个回车之间字符串是数组的一个元素。定义一个数组，将属性 Lines 中的内容存到这个数组中。(提示：string[] s=new string [richTextBox1.Lines.Length];s= richTextBox1.Lines)

(2) 为设计的单文档写字板增加工具栏，实现建新文件，打开文件，存文件等功能。

(3) 在工具栏中，增加 2 个下拉列表文本框，一个选择使用的字体，一个选择字体的字号。

(4) 在工具栏中，增加 3 个按钮，分别设定字符为黑体，斜体，增加下划线。

(5) 在工具栏中，增加 1 个按钮，用 ColorDialog 对话框选择字体的颜色。

(6) 如何实现全选菜单项。

(7) RichTextBox 控件的属性 Modified 可以指示用户是否修改文本框控件的内容。请修改 4.7 节程序，使用属性 Modified 判断用户是否修改了 RichTextBox 控件中文本内容。

(8) 在查找对话框中，增加两个多选框，选择是否允许反向查和区分大小写，并实现反向查找和不区分大小写查找。

(9) 在实现打开文件和另存为功能时，使用属性 InitialDirectory 和属性 DefaultExt。

~~(11)~~(10) RichTextBox 控件的属性 SelectionAlignment 表示段落的对齐方式，在工具栏中增加三个按钮，分别实现段落的左对齐(HorizontalAlignment.Left)、右对齐(HorizontalAlignment.Right)、中间对齐(HorizontalAlignment.Center)。(提示：3 个按钮应是互斥的，用分隔符表示 3 个按钮是一组。工具条按钮属性 Style 设置为 ToolBarButtonStyle.Separator，则按钮将显示为一个按钮分隔符，而不是按钮。)

(11) RichTextBox 控件的属性 SelectionCharOffset 可以设定选中的字符为上下标，可以为一个整数，为 0 表示正常字符，负数表示下标，正数表示上标。请在工具条中增加三个按钮，分别实现上下标功能。

(12) 请实现完整的单文档编辑器，具有前边介绍的单文档编辑器的功能。

(13) 请实现完整的多文档编辑器，具有前边介绍的单文档编辑器的功能。

(14) 请实现完整的选项卡式多文档文本编辑器，具有前边介绍的单文档编辑器的功能。

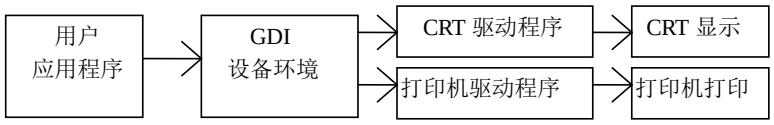
带格式的：项目符号和编号

第五章 图形图像编程

本章的目的是介绍图形图像编程的方法，希望在学了本章以后，能编制象 Windows 画图那样的程序。本章的重点是学习 Graphics 类中对象(象笔、刷子等)及各种方法的使用，以及 Bitmap 类的应用。

5.1 图形设备环境接口(GDI)

为了在 Windows 窗口输出数据(字符或图形)，Windows 操作系统提供了一些工具和函数，例如提供笔用来定义图形外轮廓线的颜色及粗细，提供刷子定义添充封闭图形内部的颜色和格式，提供不同输出字体，提供函数用来输出字符或绘制图形等等。所有这些工具和函数被放在图形设备接口函数库中(GDI32.DLL)，它负责 CRT 显示及打印。根据设备不同，可以构造不同的设备环境(GDI)，使输出图形或字符与设备无关，既无论是在 CRT 显示还是在打印机上打印同一个图形或字符，都用相同的函数。GDI 所扮演的角色如下图所示：



用户应用程序根据是在 CRT 显示还是在打印机打印，首先选择 CRT 显示设备环境或打印设备环境，然后调用 GDI 中的同名函数实现在 CRT 显示或在打印机上打印。而 GDI 设备环境根据选择的不同设备，调用不同的设备驱动程序，在 CRT 上显示或在打印机上打印。而这些驱动程序都是各个设备制造厂商提供的。这样做的最大好处是应用程序和设备无关，应用程序不必为不同的设备编制不同的程序。为使用不同的设备，无论是不同的显卡，还是不同的打印机，只要安装该设备的驱动程序，应用程序就可以使用该设备，微软的 Word 程序可以使用不同的打印机就是使用了这个原理。

.NET 系统的基础类库(.Net FrameWork)对 Windows 操作系统的图形设备接口函数库(GDI32.DLL)进行了扩充，并用类进行了封装，一般叫做 GDI+。使用 GDI+绘图更加方便快捷。为了使用 GDI+图形功能，必须引入以下命名空间：System.Drawing，System.Drawing.Priniting， System.Drawing.Imaging， System.Drawing.Drawing2D，System.Drawing.Design，System.Drawing.Text。

5.2 Graphics 类

System.Drawing.Graphics 类对 GDI+进行了封装，Graphics 类提供一些方法完成各种图形的绘制。Graphics 类对象与特定的设备关联，为了在不同的设备上用完全相同的代码完成同样的图形，应根据不同的设备建立不同的 Graphics 类对象。Graphics 类是密封类，不能有派生类。

5.2.1使用 Graphics 类绘图的基本步骤

GDI+大部分功能被封装在 Graphics 类中，Graphics 类提供了一些工具和函数，例如提供笔用来定义图形外轮廓线的颜色及粗细，提供刷子定义添充封闭图形内部的颜色和格式，提供不同输出字体，提供函数用来输出字符或绘制图形等等。为了在窗体中或其它控件中使用这些工具和函数绘图，必须首先得到这些窗体或控件的使用的 Graphics 类对象。下面的例子，在窗体中增加了一个按钮，单击按钮将在窗体中画一个边界为红色，内部填充蓝色的圆。该程序段说明了使用 Graphics 类绘图的基本步骤。按钮的单击事件处理函数如下：

```
private void button1_Click(object sender,System.EventArgs e)
{   Graphics g=this.CreateGraphics();//得到窗体使用的 Graphics 类对象
    Pen pen1=new Pen(Color.Red);//创建红色笔对象
    SolidBrush brush1=new SolidBrush(Color.Blue);//创建蓝色刷子对象
    g.DrawEllipse(pen1,10,10,100,100);//用红色笔画圆的边界
    g.FillEllipse(brush1,10,10,100,100);//用蓝色刷子填充圆的内部
}
```

运行后，单击按钮，出现边界为红色，内部填充为蓝色的圆。

5.2.2窗体的 Paint 事件

运行上例，单击按钮，出现边界为红色，内部填充蓝色的圆。最小化后，再最大化，图形不见了。这是因为用户 Form 窗体用户区内容可能被破坏，例如窗体最小化后，再最大化，菜单被打开再关闭，打开对话框再关闭等，用户区内容被覆盖。Windows 并不保存被破坏的用户区内容，而是由应用程序自己恢复被破坏的用户区的内容。当应用程序窗口用户区内容被破坏后需恢复时，Windows 操作系统向应用程序发送 Paint 事件，应用程序应把在窗口用户区输出数据的语句放在 Paint 事件处理函数中，Windows 发 Paint 事件时，能调用这些在窗口用户区输出数据的语句恢复被破坏的内容。Form 窗体不能自动响应 Paint 事件，程序员必须生成 Paint 事件处理函数。修改上例，增加 Form 窗体的 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;//得到窗体的使用的 Graphics 类对象
    Pen pen1=new Pen(Color.Red);
    SolidBrush brush1=new SolidBrush(Color.Blue);
    g.DrawEllipse(pen1,10,10,100,100);
    g.FillEllipse(brush1,10,10,100,100);
}
```

运行后，出现边界为红色，内部填充蓝色的圆。最小化后，再最大化，图形不消失。

5.3 GDI+中三种坐标系统：

GDI+定义了三种坐标系统，并提供了三种坐标转换的方法 Graphics.TransformPoints()。

- 全局坐标系统。
- 页面(Page)坐标系统：左上角为原点，横向 x 轴向右为正方向，纵向 y 轴向下为正方向。单位为像素。这是默认的坐标系统。

- 设备坐标系统：可以指定特定测量单位的页面(Page)坐标系统。如果单位为像素，和页面(Page)坐标系统相同。

5.4 GDI+中常用的结构

本节介绍 GDI+中常用的结构，包括：Point、PointF、Size、SizeF、Rectangle、RectangleF、Color 等。它们都在名字空间 System.Drawing 中定义的。

5.4.1 结构 Point 和 PointF

点结构有两个成员：X，Y，表示点的 x 轴和 y 轴的坐标。其常用构造函数如下：

```
Point p1=new Point(int X,int Y);//X,Y 为整数
PointF p2=new PointF(float X,floa Y);//X,Y 为浮点数
```

5.4.2 结构 Size 和 SizeF

Size 和 SizeF 用来表示尺寸大小，有两个成员：Width 和 Height。常用构造函数如下：

```
public Size(int width,int height);
public SizeF(float width,float height);
```

5.4.3 结构 Rectangle 和 RectangleF

结构 Rectangle 和 RectangleF 用来表示一个矩形，常用属性如下：

- Top：Rectangle 结构左上角的 y 坐标。
- Left：Rectangle 结构左上角的 x 坐标。
- Bottom：Rectangle 结构右下角的 y 坐标。
- Right：Rectangle 结构右下角的 x 坐标。
- Width：获取或设置此 Rectangle 结构的宽度。
- Height：获取或设置此 Rectangle 结构的高度。
- Size：获取或设置此 Rectangle 的大小。
- X:获取或设置此 Rectangle 结构左上角的 x 坐标。
- Y:获取或设置此 Rectangle 结构左上角的 y 坐标。

其常用构造函数为：

```
//参数为矩形左上角坐标的点结构 location 和代表矩形宽和高的 Size 结构 size
Rectangle(Point location,Size size);//参数也可为 PointF 和 SizeF
//参数为矩形左上角 x 和 y 坐标,宽,高
Rectangle(int X,int Y,int width,int height);//X 和 Y 也可为 float
```

5.4.4 结构 Color

Color 结构表示颜色，结构中包含一个无符号 32 位数代表颜色。任何一种颜色可以用

透明度(al)，蓝色(bb)，绿色(gg)，红色(rr)合成，格式为 0xalrrbbgg，其中 al,bb,gg,rr 为 0 到 255 间的二进制数。常用方法如下：

- public static Color FromArgb(int alpha,int rr,int gg,int bb);
  从四个分量（透明度、红色、绿色和蓝色）值创建 Color 结构。每个分量的值仅限于 8 位（小于 256）。alpha 值表示透明度，=0 为完全透明，=255 为完全不透明
- public static Color FromArgb(int rr,int gg,int bb);
  从指定的 8 位颜色值（红色、绿色和蓝色）创建 Color 结构。透明度值默认为 255（完全不透明）。每个分量的值仅限于 8 位（小于 256）。红色为(255,0,0)，绿色为(0,255,0)，蓝色为(0,0,255)。
- public static Color FromArgb(int alpha,Color color);
  从指定的 Color 结构创建新 Color 结构，使用新指定的透明度值 alpha。alpha 值仅限于 8 位。透明度及颜色的使用方法的例子如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;
    SolidBrush RedBrush=new SolidBrush(Color.FromArgb(128,255,0,0));//半透明
    SolidBrush GreenBrush=new SolidBrush(Color.FromArgb(128,0,255,0));
    SolidBrush BlueBrush=new SolidBrush(Color.FromArgb(128,0,0,255));
    g.FillRectangle(RedBrush,0,0,80,80);
    g.FillRectangle(GreenBrush,40,0,80,80);
    g.FillRectangle(BlueBrush,20,20,80,80);
}
```

效果如右图，可以将透明度 alpha 值设为 255，再运行一次，看看有何不同。C#中还预定义了一些颜色常数，例如黑色为 Color.Black，红色为 Color.Red 等等，可用帮助察看。

5.5 画笔

Pen 类对象指定绘制的图形外轮廓线宽度和颜色。Pen 类有 4 个构造函数，分别是：

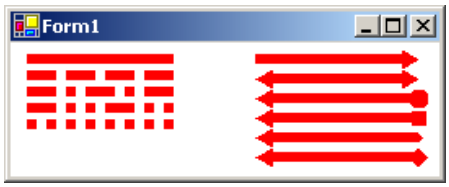
- public Pen(Color color);//建立颜色为 color 的笔，宽度默认为 1
- public Pen(Color color，float width);//建立颜色为 color 的笔，宽度为 width
- public Pen(Brush brush);//使用刷子为笔
- public Pen(Brush,float width);//使用刷子为笔，宽度为 width

Pen 类常用的属性：Color 为笔的颜色，Width 为笔的宽度，DashStyle 为笔的样式，EndCap 和 StartCap 为线段终点和起点的外观。下例显示各种笔的 DashStyle、EndCap 和 StartCap 不同选项的样式(见下图)。主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;
    Pen pen1=new Pen(Color.Red,6);//默认为实线笔
    g.DrawLine(pen1,10,10,100,10);//画实线，图中左边第 1 条线
    pen1.DashStyle=System.Drawing.Drawing2D.DashStyle.Dash;//虚线笔
    g.DrawLine(pen1,10,20,100,20);//画虚线，图中左边第 2 条线
    pen1.DashStyle=System.Drawing.Drawing2D.DashStyle.DashDot;//点，短线风格的线
    g.DrawLine(pen1,10,30,100,30);//图中左边第 3 条线
    //双点，短线风格的线
```

```
        pen1.DashStyle=System.Drawing.Drawing2D.DashStyle.DashDotDot;
        g.DrawLine(pen1,10,40,100,40);//图中左边第 4 条线
        pen1.DashStyle=System.Drawing.Drawing2D.DashStyle.Dot;//由点组成的线
        g.DrawLine(pen1,10,50,100,50);//图中左边第 5 条线
        pen1.DashStyle=System.Drawing.Drawing2D.DashStyle.Solid;//实线笔
        pen1.EndCap=System.Drawing.Drawing2D.LineCap.ArrowAnchor;//后箭头
        g.DrawLine(pen1,150,10,250,10);//图中右边第 1 条线
        pen1.StartCap=System.Drawing.Drawing2D.LineCap.ArrowAnchor;//前箭头
        g.DrawLine(pen1,150,22,250,22);//图中右边第 2 条线
        pen1.EndCap=System.Drawing.Drawing2D.LineCap.RoundAnchor;
        g.DrawLine(pen1,150,34,250,34);//图中右边第 3 条线
        pen1.EndCap=System.Drawing.Drawing2D.LineCap.SquareAnchor;
        g.DrawLine(pen1,150,46,250,46);//图中右边第 4 条线
        pen1.EndCap=System.Drawing.Drawing2D.LineCap.Triangle;
        g.DrawLine(pen1,150,58,250,58);//图中右边第 5 条线
        pen1.EndCap=System.Drawing.Drawing2D.LineCap.DiamondAnchor;
        //图中右边第 6 条线
        g.DrawLine(pen1,150,70,250,70);
}
```

运行效果如右图：



5.6 创建画刷

画刷类对象指定填充封闭图形内部的颜色和样式，封闭图形包括矩形、椭圆、扇形、多边形和任意封闭图形。GDI+系统提供了了几个预定义画刷类，包括：

- SolidBrush：单色画刷。在名字空间 System.Drawing 中定义。
- HatchBrush：阴影画刷。以下画刷在名字空间 System.Drawing.Drawing2D 中定义。
- TextureBrush：纹理(图像)画刷。
- LinearGradientBrush：颜色渐变画刷。
- PathGradientBrush：使用路径及复杂的混合渐变画刷。

5.6.1单色画刷 SolidBrush

前边已使用过单色画刷。其构造函数只有 1 个，定义如下：

```
SolidBrush brush1=new SolidBrush(Color color);//建立指定颜色的画刷
```

在使用中可以修改其属性 Color 来修改其颜色，例如：brush1.Color=Color.Green;

5.6.2阴影画刷 HatchBrush

用指定样式(例如，多条横线、多条竖线、多条斜线等)、指定线条的颜色和指定背景颜

色定义的画刷，阴影画刷有两个构造函数：

```
//指定样式和线条的颜色的构造函数，背景色被初始化为黑色。
HatchBrush brush1=new HatchBrush(HatchStyle h,Color c);
//指定样式、线条的颜色和背景颜色的构造函数。
HatchBrush brush1=new HatchBrush(HatchStyle h,Color c1，Color c2);
```

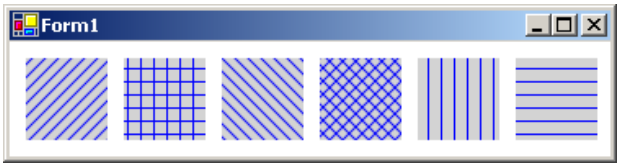
有 3 个属性如下：

- 属性 backgroundColor：画刷背景颜色。
- 属性 foreColor：画刷线条的颜色。
- 属性 HatchStyle：该属性是只读的，不能修改，表示画刷的不同样式。

例子 5.6.2：显示了阴影画刷属性 HatchStyle 为不同值时画刷的不同样式。在 Form1.cs 文件头部增加语句：using System.Drawing.Drawing2D，主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;//得到窗体的使用的 Graphics 类对象
    HatchBrush b1=new
            HatchBrush(HatchStyle.BackwardDiagonal,Color.Blue,Color.LightGray);
    g.FillRectangle(b1,10,10,50,50);//矩形被填充左斜线，第 1 图
    HatchBrush b2=new HatchBrush(HatchStyle.Cross,Color.Blue,Color.LightGray);
    g.FillRectangle(b2,70,10,50,50);//矩形被填充方格，第 2 图
    HatchBrush b3=new
            HatchBrush(HatchStyle.ForwardDiagonal,Color.Blue,Color.LightGray);
    g.FillRectangle(b3,130,10,50,50);//矩形被填充右斜线，第 3 图
    HatchBrush b4=new
            HatchBrush(HatchStyle.DiagonalCross,Color.Blue,Color.LightGray);
    g.FillRectangle(b4,190,10,50,50);//矩形被填充菱形，第 4 图
    HatchBrush b5=new
            HatchBrush(HatchStyle.Vertical,Color.Blue,Color.LightGray);
    g.FillRectangle(b5,250,10,50,50);//矩形被填充竖线，第 5 图
    HatchBrush b6=new
            HatchBrush(HatchStyle.Horizontal,Color.Blue,Color.LightGray);
    g.FillRectangle(b6,310,10,50,50);//矩形被填充横线，第 6 图
}
```

运行效果如右图：



5.6.3纹理(图像)画刷 TextureBrush

纹理(图像)画刷使用图像来填充封闭曲线的内部，有 8 个构造函数，最简单的构造函数如下，其余请用帮助查看。

```
TextureBrush(Image bitmap);//使用位图类对象作为画刷构造函数的参数
```

下边的例子使用文件 n2k.bmp 建立位图类对象作为画刷的图案，在 Form1 文件的头部增加语句：using System.Drawing.Drawing2D，主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;//得到窗体的使用的 Graphics 类对象
    Pen pen1=new Pen(Color.Red);
    //位图类对象作为画刷图案，使用文件 n2k.bmp 建立位图类对象见 5.10 节
    TextureBrush b1=
                new TextureBrush(new Bitmap("C:\\WINNT\\system32\\n2k.bmp"));
    g.FillRectangle(b1,10,10,200,100);
    g.DrawRectangle(pen1,10,10,200,100);
}
```

文件 C:\WINNT\system32\n2k.bmp 定义的图形的显示效果如下：

运行效果如右图。

5.6.4颜色渐变画刷 LinearGradientBrush

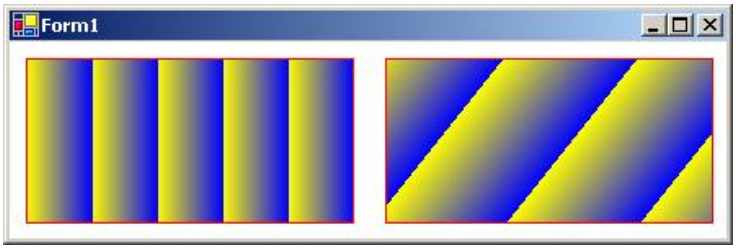
该类封装双色渐变和自定义多色渐变画刷。所有颜色渐变都是沿由矩形的宽度或两个点指定的直线定义的。默认情况下，双色渐变是沿指定直线从起始色到结束色的均匀水平线性变化。有 8 个构造函数，最简单的构造函数如下，其余请用帮助查看。

```
public LinearGradientBrush(
    Point point1,//point1 作为线性渐变直线开始点，也可以为 PointF
    Point point2,//point2 作为线性渐变直线结束点，也可以为 PointF
    Color color1,Color color2);//线性渐变开始颜色和结束颜色
```

下边的例子显示了不同线性渐变直线开始点和结束点，使用颜色渐变画刷从黄渐变到蓝的效果。在 Form1 文件的头部增加语句：using System.Drawing.Drawing2D，主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;//得到窗体的使用的 Graphics 类对象
    Pen pen1=new Pen(Color.Red);
    Point p1=new Point(10,10);//p1 作为渐变直线开始点，也可以为 PointF
    Point p2=new Point(50,10);//p2 作为渐变直线结束点，也可以为 PointF
    LinearGradientBrush brush1=//从黄渐变到蓝，见下图左图
            new LinearGradientBrush(p1,p2,Color.Yellow,Color.Blue);
    g.FillRectangle(brush1,10,10,200,100);
    g.DrawRectangle(pen1,10,10,200,100);
    p1=new Point(220,10);
    p2=new Point(270,50);
    LinearGradientBrush brush2=//从黄渐变到蓝，见下图右图
            new LinearGradientBrush(p1,p2,Color.Yellow,Color.Blue);
    g.FillRectangle(brush2,230,10,200,100);
    g.DrawRectangle(pen1,230,10,200,100);
}
```

运行效果如下图：



5.6.5画刷 PathGradientBrush

画刷 PathGradientBrush 可以实现复杂的渐变颜色。有 5 个构造函数，这里只介绍其中的一个，其参数 GraphicsPath 类使用见例子及 5.7.11 节。其余构造函数请用帮助查看。

```
public PathGradientBrush(GraphicsPath path);//画刷 PathGradientBrush 构造函数
```

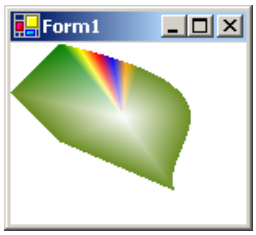
画刷属性 SurroundColors：一个 Color 结构的数组，它表示与 PathGradientBrush 对象填充的路径中的各点相关联的颜色的数组。SurroundColors 数组中的每一 Color 结构都对应于路径中的点。

下边的例子介绍画刷 PathGradientBrush 的使用方法，在 Form1.cs 文件的头部增加语句：

```
using System.Drawing.Drawing2D，
```

主窗体 Paint 事件处理函数如下，运行效果如下图。

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;//得到窗体使用的 Graphics 类对象
    Point[] p1=new Point[6];//点结构数组,有 6 个元素
    p1[0]=new Point(30,0);
    p1[1]=new Point(160,30);
    p1[2]=new Point(90,60);
    p1[3]=new Point(100,90);
    p1[4]=new Point(30,60);
    p1[5]=new Point(0,30);
    GraphicsPath path=new GraphicsPath(//建立 GraphicsPath 类对象
        p1,//由点数组 p1 定义绘制刷子的外轮廓线的路径
        new Byte[]{//定义数组 p1 每个点元素的关联类型，第 1 点是开始点
        //贝塞尔曲线必须由 4 点组成，因此第 1、2、3、4 点组成一条贝塞尔曲线，
        (byte)PathPointType.Start,(byte)PathPointType.Bezier,
        (byte)PathPointType.Bezier,(byte)PathPointType.Bezier,
        //第 5、6 点是直线，从第 4 点到第 5 点和从第 5 点到第 6 点画直线
        (byte)PathPointType.Line,(byte)PathPointType.Line,});
        //为了形成闭合曲线，增加最后一条直线
    PathGradientBrush brush1=new PathGradientBrush(path);//生成画刷
    brush1.SurroundColors=new Color[]{Color.Green,Color.Yellow,Color.Red,
        Color.Blue,Color.Orange,Color.OliveDrab,};//设置属性 SurroundColors 的值
    g.FillPath(brush1,path);
}
```



5.7 基本图形的绘制和填充

Graphics 类提供了一些绘图方法，用来绘制或填充各种图形。本节介绍这些方法。

5.7.1 绘制线段

两个绘制线段的函数和一个绘制多条线段的函数定义如下：

- void DrawLine(Pen pen,int x1,int y1,int x2,int y2);

其中 pen 为画笔，(x1,y1)为画线起点坐标，(x2,y2)为画线终点坐标。

- DrawLine(Pen pen,Point p1,Point p2);

其中 pen 为画笔，点 p1 为画线起点坐标，点 p2 为画线终点坐标。

- public void DrawLines(Pen pen,Point[] points);

此方法绘制多条线段。从 points[0]到 points[1]画第 1 条线，从 points[1]到 points[2]画第 2 条线，依此类推。

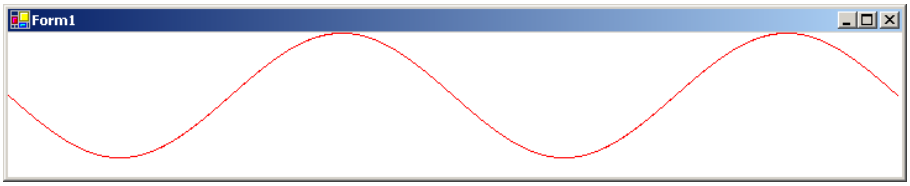
例子 e5_7_1A：使用 DrawLine()的例子，为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;
    Pen pen1=new Pen(Color.Red);
    g.DrawLine(pen1,30,30,100,100);//用笔 pen1 从点(30,30)到(100,100)画直线
    Point p1=new Point(30,40);
    Point p2=new Point(100,110);
    g.DrawLine(pen1,p1,p2);//用笔 pen1 从点(30,40)到(100,110)画直线
}
```

例子 e5_7_1B：使用绘制线段函数画任意曲线(画正弦曲线，注意如何使用数学函数)。

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();//得到窗体的使用的 Graphics 类对象
    Pen pen1=new Pen(Color.Red);
    float y=50,y1,x1,x2;
    for(int x=0;x<720;x++)//画正弦曲线
    {   x1=(float)x;
        x2=(float)(x+1);
        y1=(float)(50+50*Math.Sin((3.14159/180.0)*(x+1)));
        g.DrawLine(pen1,x1,y,x2,y1);
        y=y1;
    }
}
```

运行，在窗体中可以看到一条红色正弦曲线如下图。



例子 e5_7_1C：在画图程序中，可以用鼠标画任意曲线，现实现用拖动鼠标左键在主窗体中画曲线。每条曲线都是由若干很短的线段组成。鼠标左键按下状态下，移动鼠标，每移动很短距离，画出这段线段，所有这些线段组合起来，形成一条曲线。

(1) 新建项目。增加两个私有变量:

```
private bool mark=false;//表示鼠标左键是否按下，如按下鼠标再移动将画曲线
private Point point;//记录画下一很短线段的起始点。
```

(2) 为 Form 窗体的事件 OnMouseDown，OnMouseUp，OnMouseMove 增加事件函数如下：

```
private void Form11_MouseDown(object sender,//鼠标按下事件处理函数
                                    System.Windows.Forms.MouseEventArgs e)
{   if(e.Button==MouseButtons.Left)//如果鼠标左键按下
    {   point.X=e.X;//记录曲线的第一个点的坐标
        point.Y=e.Y;
        mark=true;//表示鼠标左键已按下，鼠标如果再移动，将画曲线
    }
}
private void Form1_MouseMove(object sender,//鼠标移动事件处理函数
                                  System.Windows.Forms.MouseEventArgs e)
{   if(mark)//如果鼠标左键按下
    {   Graphics g=this.CreateGraphics();//得到窗体的使用的 Graphics 类对象
        Pen pen1=new Pen(Color.Black);//黑笔
        g.DrawLine(pen1,point.X,point.Y,e.X,e.Y);//画线
        point.X=e.X;//记录画下一线段的起始点的坐标
        point.Y=e.Y;
    }
}
private void Form1_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)
{   mark=false;}//停止画线
```

(3) 运行，在 Form 窗体拖动鼠标左键可以画线。但最小化后再最大化后，图形消失。修改上例，使其能克服这个缺点。实现的思路是记录每一条曲线的每一条很短线段的坐标。使用 ArrayList 类对象记录曲线以及曲线中的点，请注意 ArrayList 类使用方法。

(4) 为定义主窗体的 Form1 类中增加私有变量:

```
private ArrayList Point_List;//用来记录 1 条曲线的所有点。
private ArrayList Line_List;//用来记录每条曲线，既 Point_List 对象。
```

在 Form1 类构造函数中增加语句：Line_List=new ArrayList();

(5) 修改主窗体事件 OnMouseDown，OnMouseUp，OnMouseMove 事件处理函数如下：

```
private void Form1_MouseDown(object sender,//阴影部分为修改的内容
                                  System.Windows.Forms.MouseEventArgs e)
{   if(e.Button==MouseButtons.Left)
    {   Point_List=new ArrayList();//建立数组，记录 1 条曲线的所有点
        point.X=e.X;
        point.Y=e.Y;
        mark=true;
        Point_List.Add(point);//曲线起点的坐标
    }
```

```
}
private void Form1_MouseMove(object sender,
                                System.Windows.Forms.MouseEventArgs e)
{   if(mark)
    {   Graphics g=this.CreateGraphics();
        Pen pen1=new Pen(Color.Black);
        g.DrawLine(pen1,point.X,point.Y,e.X,e.Y);
        point.X=e.X;
        point.Y=e.Y;
        Point_List.Add(point);//记录曲线中其它点的坐标
    }
}
private void Form1_MouseUp(object sender,System.Windows.Forms.MouseEventArgs e)
{   mark=false;
    Line_List.Add(Point_List);//记录此条线，注意参数是 Point_List
}
```

(6) 增加 Form 窗体的 Paint 事件函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;
    Pen pen1=new Pen(Color.Black);
    Point p1,p2;
    foreach(ArrayList l in Line_List)//取出每条线
    {   for(int k=0;k<(l.Count-1);k++)//重画每条线的点
        {   p1=(Point)l[k];
            p2=(Point)l[k+1];
            g.DrawLine(pen1,p1,p2);
        }
    }
}
```

(7) 运行，在 Form 窗体拖动鼠标可以画线。最小化后再最大化后，图形不消失。

5.7.2 ArrayList 类

ArrayList 类是容量可以动态增加的数组，其元素类型可以是任意类型。和其它数组一样，ArrayList 类可以使用对象名[索引号]引用其元素，索引号也从零开始。前边已多次使用此类，例如：控件 ListBox 和 ComboBox 的属性 Items，以及 5.7.1 节中的例子。其常用的属性及方法如下：

- 属性 Count：ArrayList 中实际包含的元素数。
- 方法 Add：将参数指定的对象添加到 ArrayList 对象的结尾处。
- 方法 Clear：从 ArrayList 中移除所有元素。
- 方法 Contains：bool 类型，确定参数指定的元素是否在 ArrayList 中。
- 方法 IndexOf：int 类型，顺序查找和参数指定对象相同的第一个元素的索引。
- 方法 Insert：插入数据，第 1 个参数为插入的位置(索引号)，第 2 个参数为插入的对象。
- 方法 LastIndexOf：顺序查找和参数指定对象相同的最后一个元素的索引。

- 方法 RemoveAt：移除指定索引处的元素。
- 方法 Sort：对整个 ArrayList 中的元素进行排序。

5.7.3 画椭圆(圆)及键盘消息的使用

两个画椭圆的函数的功能是画指定矩形的内切椭圆，如为正方形则画圆，两个函数如下：

- void DrawEllipse(Pen pen,int x,int y,int width,int height);
  其中 pen 为画笔，画外轮廓线，(x1,y1)为指定矩形的左上角坐标，width 为指定矩形的宽，height 为指定矩形的高。
- void DrawEllipse(Pen pen,Rectangle rect);
  其中 pen 为画笔，画外轮廓线，rect 为指定矩形结构对象。

例子 5_7_3A：画椭圆。为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red);
    g.DrawEllipse(pen1,10,10,200,100);
    Rectangle rect=new Rectangle(20,20,100,100);
    g.DrawEllipse(pen1,rect);
}
```

例子 5_7_3B：用四个箭头键移动窗体中的圆球。移动圆球，实际是先把前边画的圆擦掉，在新的位置重新画圆。如要擦掉圆，可以用窗体背景色作为笔和刷子的颜色，在圆的原先位置重画和填充圆。注意键盘事件处理函数的使用。具体实现步骤如下：

(1) 新建项目。在 Form1 类中增加变量：int x,y，记录定义圆位置的矩形左上角的坐标。

(2) 在 Form1 类中增加一个方法，该方法按照参数指定颜色画圆，方法定义如下：

```
void DrawCir(Color color)//参数是画圆的笔和刷子的颜色
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(color);
    SolidBrush brush1=new SolidBrush(color);
    g.DrawEllipse(pen1,x,y,100,100);
    g.FillEllipse(brush1,x,y,100,100);
}
```

(3) 为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   DrawCir(Color.Red);}
```

(4) 为主窗体 KeyDown 事件增加事件函数如下：(注意不要使用 KeyPress 事件，其事件处理函数的第 2 个参数 e 的 e.KeyChar 是按下键的 ASCII 值，但很多键无 ASCII 值。)

```
private void Form1_KeyDown(object sender,System.Windows.Forms.KeyEventArgs e)
{   switch (e.KeyCode)//e.KeyCode 是键盘每个键的编号
    {   case Keys.Left://左箭头键编号
            DrawCir(this.BackColor);//用 Form 窗体的背静色画圆，即擦除圆
            x=x-10;//圆左移
            DrawCir(Color.Red);//在新的位置用红色画圆，效果是圆左移
            break;
        case Keys.Right://圆右移
```

```
            DrawCir(this.BackColor);
            x+=10;
            DrawCir(Color.Red);
            break;
        case Keys.Down://圆下移
            DrawCir(this.BackColor);
            y+=10;
            DrawCir(Color.Red);
            break;
        case Keys.Up://圆上移
            DrawCir(this.BackColor);
            y=y-10;
            DrawCir(Color.Red);
            break;
    }
}
```

(5) 运行，可以用 4 个箭头键移动红色圆。

使用 KeyDown 事件，事件处理函数的第 2 个参数 e 的 e.KeyCode 是键盘每个键的编号，其它常用键的编号如下：数字键 0-9 编号为 Keys.D0-Keys.D9；字母键 A-Z 为 Keys.A-Keys.Z；F0-F12 键表示为 Keys.F0-Keys.F12 等。

如使用 KeyPress 事件，事件处理函数的第 2 个参数 e 的 e.KeyChar 表示按键的 ACSII 值，例如可用如下语句 if(e.KeyChar==(char)13)判断是否按了回车键。

5.7.4 画矩形

两个绘制 1 个矩形(正方形)的函数和一个绘制多个矩形(正方形)的函数定义如下：

- void DrawRectangle(Pen pen,int x,int y,int width,int height);
  其中 pen 为画笔，画外轮廓线，(x1,y1)为矩形的左上角坐标，width 为指定矩形的宽，height 为指定矩形的高。
- void DrawRectangle(Pen pen,Rectangle rect)；
  其中 pen 为画笔，画外轮廓线，rect 为矩形结构对象。
- public void DrawRectangles(Pen pen,Rectangle[] rects);
  绘制一系列由 Rectangle 结构指定的矩形。

例子 5_7_3：画矩形。为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red);
    g.DrawRectangle(pen1,10,10,200,100);
    Rectangle rect=new Rectangle(20,20,100,100);
    g.DrawRectangle(pen1,rect);
}
```

5.7.5 绘制圆弧

DrawArc 方法绘制指定矩形的内切椭圆(圆)中的一段圆弧，方法定义如下：

```
void DrawArc(Pen pen,int x,int y,int width,int height,int StartAngle,int EndAngle);
```

其中 pen 为画笔，画外轮廓线，(x1,y1)为矩形的左上角坐标，width 为指定矩形的宽，height 为指定矩形的高。StartAngle 为圆弧的起始角度,EndAngle 为圆弧的结束角度，单位为度。指定矩形的中心点做矩形宽和高的的垂线作为 x，y 轴，中心点为圆点。圆点右侧 x 轴为 0 度，顺时针旋转为正角度，逆时针旋转为负角度。

例子 5_7_4：画圆弧。为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red);
    g.DrawArc(pen1,10,10,200,100,0,30);
}
```

5.7.6 DrawPie 方法

DrawPie 方法方法绘制指定矩形的内切椭圆(圆)中的一段圆弧，并且用指定矩形的中心点连接开始点和结束点，这个图形叫做饼图，方法定义如下：

```
void DrawPie(Pen pen,int x,int y,int width,int height,int StartAngle,int EndAngle);
```

方法参数和 DrawArc 方法参数相同。

例子 5_7_5：画饼图。为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red);
    g.DrawPie(pen1,10,10,200,100,0,30);
}
```

5.7.7 Bezier 曲线

可以使用 DrawBezier 方法画一条 Bezier 曲线。它的两个画线函数定义如下：

- Void DrawBezier(Pen pen,float x1,float y1,
  float x2,float y2,float x3,float y3,float x4,float y4);
  其中 pen 是画笔对象,画轮廓线,(x1,y1)是起始点，(x2,y2)是第一控制点，(x3,y3)是第二控制点，(x4,y4)是结束点。
- Void DrawBezier(Pen pen,Point p1,Point p2,Point p3,Point P4);
  其中 pen 是画笔对象,画轮廓线,p1 是起始点，p2 是第一控制点，p3 是第二控制点，p4 是结束点。

例子 5_7_6：画 Bezier 曲线。为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red);
```

```
    g.DrawBezier(pen1,10,10,200,100,50,60,100,200);
}
```

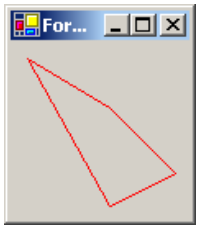
5.7.8 DrawPolygon 方法

该方法画一个多边形，使用点结构数组定义多边形的顶点。两个画线函数定义如下：

- void DrawPolygon(Pen pen,Point[] points);
- void DrawPolygon(Pen pen,PointF[] points);//点坐标可以是小数

例子 5_7_7：画一个多边形如下图，为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red);
    Point[] p1=new Point[]
    {   new Point(10,10),
        new Point(60,40),
        new Point(100,80),
        new Point(60,100)
    };
    g.DrawPolygon(pen1,p1);
}
```



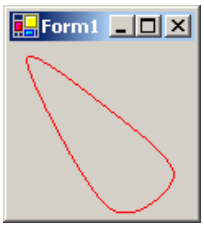
5.7.9 DrawClosedCurve 方法

DrawClosedCurve 方法用来绘制经过 Point 结构数组中每个点的闭合基数样条。基数样条是一连串单独的曲线，这些曲线连接起来形成一条较大的曲线。样条由点的数组指定，并通过该数组中的每一个点。基数样条平滑地通过数组中的每一个点，请比较一下本节的图形和上节图形的区别。如果最后一个点不匹配第一个点，则在最后一个点和第一个点之间添加一条附加曲线段以使该图闭合，点 Point 结构数组必须至少包含四个元素，此方法使用默认张力 0.5。有 4 个画线函数，常用的 2 个画线函数定义如下：

- void DrawClosedCurve(Pen pen,Point[] points);
- void DrawClosedCurve(Pen pen,PointF[] points);//点坐标可以是小数

例子 e5_7_9：使用 DrawClosedCurve 方法，绘制有 4 个元素的 Point 结构数组定义的闭合基数样条闭合曲线如下图，为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red);
    Point[] p1=new Point[]
    {   new Point(10,10),
        new Point(60,40),
        new Point(100,80),
        new Point(60,100)
    };
    g.DrawClosedCurve(pen1,p1);
```



```
}
```

5.7.10 DrawCurve 方法

用 DrawCurve 方法和 DrawClosedCurve 方法一样，用来绘制经过 Point 结构数组中每个点的闭合基数样条，但最后两个点之间不连线。常用的两个画线函数定义如下：

- void DrawPolygon(Pen pen,Point[] points);
- void DrawPolygon(Pen pen,PointF[] points);

例子 5_7_9：使用 DrawCurve 方法，绘制有 4 个元素的 Point 结构数组定义的闭合基数样条闭合曲线如下图，为主窗体 Paint 事件增加事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    Pen pen1=new Pen(Color.Red,3);
    Point[] p1=new Point[]
    {   new Point(10,10),
        new Point(60,40),
        new Point(100,80),
        new Point(60,100)
    };
    g.DrawCurve(pen1,p1);
}
```

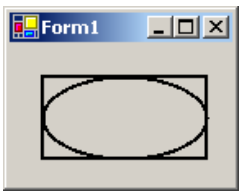
5.7.11 DrawPath 方法和 GraphicsPath 类

用 DrawPath 方法可以绘制多个曲线，方法参数 GraphicsPath 类对象 path 定义每个曲线类型。DrawPath 方法定义如下：

```
void DrawPath(Pen pen,GraphicsPath path);
```

例子 5_7_10A：用 DrawPath 方法画一个矩形和其内切椭圆如下图，在 Form1.cs 文件的头部增加语句：using System.Drawing.Drawing2D，主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Rectangle myEllipse=new Rectangle(20,20,100,50);
    GraphicsPath myPath=new GraphicsPath();//建立 GraphicsPath()类对象
    myPath.AddEllipse(myEllipse);//追加一椭圆
    myPath.Add Rectangle(myEllipse);//追加一矩形
    Pen myPen=new Pen(Color.Black,2);
    //画这条曲线，即椭圆和矩形。
    e.Graphics.DrawPath(myPen,myPath);
}
```



GraphicsPath 类是表示一系列相互连接的直线和曲线的路径，应用程序使用此路径来绘制曲线的轮廓、填充形状内部和创建剪辑区域。由属性 PathPoints(点数组)定义绘制直线和曲线的路径。路径可由任意数目的图形（子路径）组成，每一图形都是由一系列相互连接的直线和曲线或几何形状基元构成的。由属性 PathTypes（字节数组）定义属性 PathPoints(点数组)中每个点元素的关联图形或曲线类型，图形的起始点是相互连接的

一系列直线和曲线中的第一点。终结点是该序列中的最后一点。GraphicsPath 在 System.Drawing.Drawing2D 名字空间。常用属性、事件和方法定义如下：

- 构造函数 GraphicsPath()；//建立空对象
- 构造函数 public GraphicsPath(Point[] pts,byte[] types);
  参数 pts 为 Point 结构数组，数组元素表示构造路径所使用的点。参数 types 指定路径中相应点的关联图形或曲线类型数组。参见 5.6.5 节。
- 属性 PointCount：获取 PathPoints 或 PathTypes 数组中的元素数。
- 方法 IsVisible：指定点是否包含在此 GraphicsPath 对象内。
- 方法：void AddArc(Rectangle rect,float startAngle,float sweepAngle);
  在代表要描绘图形的 GraphicsPath 类对象中追加一段椭圆弧。
- 方法：void AddEllipse(Rectangle rect)，追加一椭圆(圆)
- 方法：void AddLine(Point pt1,Point pt2)，追加一线段
- 方法：void AddRectangle(Rectangle rect)，追加一矩形
  还有其它增加曲线的方法，例如：AddBezier 方法、AddBeziers 方法、AddClosedCurve 方法、AddCurve 方法、AddLines 方法、AddPath 方法、AddPie 方法、AddPolygon 方法、AddRectangles 方法、AddString 方法等，请用帮助查看。

5.7.12 DrawString 方法

DrawString 方法在指定位置并且用指定的 Brush 和 Font 对象绘制指定的文本字符串。有 6 个重载方法，常用的一个是：

```
public void DrawString(string s,//s 是为要显示的字符串
    Font font,//显示的字符串使用的字体
    Brush brush,//用刷子写字符串
    PointF point);//显示的字符串左上角的坐标
```

最后一个参数也可以是 RectangleF 对象，仍表示显示的字符串位置。还可以再增加一个参数，即第 5 个参数，StringFormat 对象，它指定应用于所绘制文本的格式化属性（如行距和对齐方式）。在打印和打印预览一节已使用了这个方法。

例子 5_7_12：用 DrawString 方法显示字符串，主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   String drawString="Sample Text";//要显示的字符串
    Font drawFont=new Font("Arial",16);//显示的字符串使用的字体
    SolidBrush drawBrush=new SolidBrush(Color.Black);//写字符串用的刷子
    PointF drawPoint=new PointF(20.0F,20.0F);//显示的字符串左上角的坐标
    e.Graphics.DrawString(drawString,drawFont,drawBrush,drawPoint);
}
```

5.7.13 DrawImage 和 DrawIcon 方法

用来在指定的位置绘制指定的 Image 对象和图标。Graphics 类中有多个 DrawImage 重载方法，最简单的是以下方法：

- public void DrawImage(Image image,Point point);
  在指定的位置使用原始物理大小绘制指定的 Image 对象。参数 1 为要绘制的 Image 对象，

参数 2 表示所绘制图像的左上角在窗体中的位置。

- public void DrawImage(Image image,Point[] destPoints);

在指定位置并且按指定形状和大小绘制指定的 Image 对象。参数 1 为要绘制的 Image 对象，参数 2 表示有 3 个元素的 Point 结构数组，三个点定义一个平行四边形。缩放和剪切 image 参数表示的图像，以在此平行四边形内显示。参数 2 也可以是一个矩形结构。

- public void DrawImage(Image image,//要绘制的 Image 对象
  Rectangle destRect,//指定所绘制图像的位置和大小，图像进行缩放以适合该矩形
  Rectangle srcRect,//指定 image 对象中要绘制的部分
  GraphicsUnit srcUnit);//枚举的成员，指定 srcRect 参数所用的度量单位

例子 5_7_1A：在指定位置绘制 Image 对象指定部分。

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Image newImage=Image.FromFile("d:\\CSARP\\1.jpg");//建立要绘制的 Image 图像
    Rectangle destRect=new Rectangle(10,10,150,150);//显示图像的位置
    Rectangle srcRect=new Rectangle(50,50,150,150);//显示图像那一部分
    GraphicsUnit units=GraphicsUnit.Pixel;//源矩形的度量单位设置为像素
    e.Graphics.DrawImage(newImage,destRect,srcRect,units);//显示
}//如果把显示图像的位置变宽，看一下效果，为什么？其它重载方法可用帮助查看。
```

- public void DrawIcon(Icon icon,Rectangle targetRect);

在 Rectangle 结构指定的区域内绘制指定的 Icon 对象表示的图标。

例子 5_7_1B：在指定位置画图标。

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Icon newIcon=new Icon("d:\\CSARP\\TASKS.ICO");
    Rectangle rect=new Rectangle(100,100,200,200);
    e.Graphics.DrawIcon(newIcon,rect);
}
```

5.7.14 FillEllipse 方法

该方法用指定画刷来填充指定矩形的内切椭圆(圆)。两个填充函数的定义如下：

- void FillEllipse(Brush brush,int x,int y,int width,int height);

其中 brush 为指定画刷，(x1,y1)为指定矩形的左上角坐标，width 为指定矩形的宽，height 为指定矩形的高。

- void DrawEllipse(Pen pen,Rectangle rect);

其中 brush 为指定画刷，rect 为指定矩形结构对象。

例子 5_7_13:用指定画刷来填充指定矩形的内切椭圆。

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    SolidBrush brush=new SolidBrush(Color.Blue);
    g.FillEllipse(brush,10,10,200,100);
    Rectangle rect=new Rectangle(120,120,100,100);
    g.FillEllipse(brush,rect);
}
```

5.7.15 FillRectangle 方法

FillRectangle 方法用指定画刷来填充指定矩形。两个填充函数定义如下：

- void FillRectangle(Brush brush,int x,int y,int width,int height);

  其中 brush 为指定画刷，(x1,y1)为矩形的左上角坐标，width 为指定矩形的宽，height 为指定矩形的高。
- void FillRectangle(Brush brush,Rectangle rect);

  其中 brush 为指定画刷，rect 为矩形结构对象。

例子 5_7_14：用指定画刷来填充指定矩形。

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    SolidBrush brush=new SolidBrush(Color.Blue);
    g.FillRectangle(brush,10,10,200,100);
    Rectangle rect=new Rectangle(120,120,100,100);
    g.FillRectangle(brush,rect);
}
```

5.7.16 FillPie 方法

FillPie 方法用指定画刷来填充指定饼图。函数定义如下：

```
void FillPie(Brush brush,int x,int y,int width,
                                    int height,int StartAngle,int EndAngle);
```

其中 brush 为指定画刷，方法其它参数和 DrawArc 方法参数相同。

例子 5_7_15：用指定画刷来填充指定饼图。

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   Graphics g=this.CreateGraphics();
    SolidBrush brush=new SolidBrush(Color.Blue);
    g.FillPie(brush,10,10,200,100,0,30);
}
```

5.7.17 FillRegion 方法和 Region 类

FillRegion 方法用刷子填充区域 Region 类对象内部。Region 类对象由矩形和路径构成。如果区域不闭合，则在最后一个点和第一个点之间添加一条额外的线段来将其闭合。方法定义如下：

```
public void FillRegion(Brush brush,Region region);
```

第 1 个参数是填充使用的刷子，第 2 个参数是指定的区域。

例子 5.7.16A：用纯蓝色刷子，使用 FillRegion 方法填充一个矩形区域。

```
private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{   SolidBrush blueBrush=new SolidBrush(Color.Blue);
    Rectangle fillRect=new Rectangle(10,10,100,100);
    Region fillRegion=new Region(fillRect);
```

```
    e.Graphics.FillRegion(blueBrush,fillRegion);
}
```

区域是输出设备显示区域的一部分。区域可以是简单的（单个矩形）或复杂的（多边形和闭合曲线的组合）。下图中的左数第 1 图显示了两个区域：一个利用矩形构造，另一个利用路径构造。可以通过合并现有的区域来创建复杂区域。Region 类提供了以下合并区域的方法：Intersect、Union、Xor、Exclude 和 Complement。两个区域的交集是同时属于两个区域的所有点的集合，方法 Intersect 可以得到两个 Region 类对象的交集。并集是多个区域的所有点的集合，方法 Union 可以得到两个 Region 类对象的并集。方法 Xor 可以得到两个 Region 类对象的并集减去这两者的交集，即下图中的左数第 4 图显示的蓝色区域。方法 Exclude 和 Complement 可以得到 1 个 Region 类对象和参数指定的 Region 类对象的不相交的部分，即下图中的左数第 5 图显示区域。

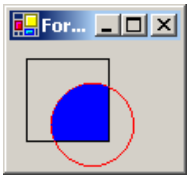
Region 类常用的方法如下：

- 构造函数 Region：可以没有参数，即创建一个空区域。也可以有一个参数，可以是 GraphicsPath、Rectangle、RectangleF 和 RegionData 类型，由此生成一个区域。
- 方法 Exclude 和 Complement：得到 1 个 Region 类对象和参数指定的 Region 类对象的不相交的部分。参数可以是 GraphicsPath、Rectangle、RectangleF 和 Region 类型。
- 方法 Equals：比较 2 个区域是否相等。参数 1 是要比较的区域，参数 2 是要绘制表面的 Graphics 对象。
- 方法 Intersect：可以得到两个 Region 类对象的交集。参数可以是 GraphicsPath、Rectangle、RectangleF 和 Region 类型。
- 方法 IsEmpty：测试是否为空区域。
- 方法 IsVisible：测试参数指定的点或矩形是否在区域中。
- 方法 Union：可以得到两个 Region 类对象的并集。
- 方法 Xor：可以得到两个 Region 类对象的并集减去这两者的交集。

例子 5_7_16B：建立 2 个矩形，有部分相交，将相交部分填充为蓝色。在 Form1.cs 文件的头部增加语句：using System.Drawing.Drawing2D，主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   Graphics g=e.Graphics;
    Rectangle regionRect=new Rectangle(10,10,50,50);
    Pen pen1=new Pen(Color.Black);
    g.DrawRectangle(pen1,regionRect);//绘制第1个矩形
    RectangleF unionRect=new RectangleF(25,25,50,50);//第2个矩形
    pen1.Color=Color.Red;
    g.DrawEllipse(pen1,unionRect);//画椭圆
    GraphicsPath myPath=new GraphicsPath();//建立GraphicsPath()类对象
    myPath.AddEllipse(unionRect);//追加一椭圆
    Region myRegion=new Region(regionRect);//建立表示第1个矩形的区域
    myRegion.Intersect(myPath);//得到两个区域的交集
    SolidBrush myBrush=new SolidBrush(Color.Blue);
```

```
    e.Graphics.FillRegion(myBrush,myRegion);//填充区域
}
```



运行效果如右图。除了以上介绍的填充方法，还有如下方法：FillClosedCurve 方法、FillPath 方法、FillPolygon 方法等，请用帮助查看。

5.8 Matrix 类和图形的平移、变形、旋转

本节介绍使用 Matrix 类实现图形的平移、变形、旋转。

5.8.1 Matrix 类

Matrix 类封装了表示几何变形的 3 行 3 列仿射矩阵，可以记录图形的平移、变形、旋转等操作。主要包括如下方法：

- 构造函数 Matrix()：创建一个空 Matrix 类对象。
- 方法 Rotate：在 Matrix 类对象中增加相对于原点顺时针旋转指定角度的操作。参数指定旋转角度。
- 方法 RotateAt：在 Matrix 类对象中增加相对于指定点顺时针旋转指定角度的操作。参数 1 指定旋转角度。参数 2 指定相应的点。
- 方法 Scale：在 X 轴或 Y 轴方向对图形放大或缩小。参数 1 指定在 X 轴方向缩放的值，参数 2 指定在 Y 轴方向缩放的值。
- 方法 Translate：使图形在 X 轴或 Y 轴方向移动。参数 1 指定在 X 轴方向移动的值，参数 2 指定在 Y 轴方向移动的值。

例子 5_8_1：下面的示例创建了复合变形（先旋转 30 度，再在 y 方向上缩放 2 倍，然后在 x 方向平移 5 个单位）的 Matrix 类对象。注意变形得顺序非常重要。一般说来，先旋转、再缩放、然后平移，与先缩放、再旋转、然后平移是不同的。

```
Matrix myMatrix=new Matrix();
myMatrix.Rotate(30);
myMatrix.Scale(1,2,MatrixOrder.Append);
myMatrix.Translate(5,0,MatrixOrder.Append);
```

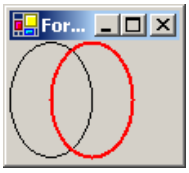
5.8.2 图形的平移、变形、旋转

GraphicsPath 类的 Transform 方法可以缩放、转换、旋转或扭曲 GraphicsPath 对象，参数 Matrix 对象表示需要的变形。

例子 5_8_2A：下面的示例代码执行下列操作：创建一个路径并向该路径添加一个椭圆。将路径绘制到主窗体上。创建一个 Matrix 类对象，在对象中增加在 X 轴方向上将路径移动 100 个单位操作。将该已变形的路径绘制到屏幕。观察一下变换前和变换后的不同，注意，初始椭圆是以黑色绘制的，而变形后的椭圆是以红色绘制的。在 Form1.cs 文件的头部增加语句：using System.Drawing.Drawing2D，主窗体 Paint 事件处理函数如下：

```
private void Form1_Paint(object sender,System.Windows.Forms.PaintEventArgs e)
{   GraphicsPath myPath=new GraphicsPath();//创建一个路径
```

```
    myPath.AddEllipse(0,0,50,70);//向路径添加一个椭圆
    e.Graphics.DrawPath(Pens.Black,myPath);//用黑笔画出这个椭圆
    Matrix translateMatrix=new Matrix();//创建一个 Matrix 类对象
    translateMatrix.Translate(25,0);//在 X 轴方向上移动 25 个单位
    //根据 Matrix 类对象修改路径 myPath
    myPath.Transform(translateMatrix);
    //用红笔按新路径画这个椭圆
    e.Graphics.DrawPath(new Pen(Color.Red,2),myPath);
}
```



运行效果如右图。请读者实现变形、旋转。

5.8.3 仿射矩阵

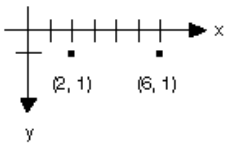
m×n 矩阵是以 m 行和 n 列排列的一组数字，例如一个 3×3 矩阵记为如下图形式，也可简记为：$[a_{33}]$。

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix}$$

两个行、列分别相同的矩阵可以相加，例如：$[a_{33}]+[b_{33}]=[c_{33}]$，矩阵相加运算的规则是：$c_{ij}=a_{ij}+b_{ij}$，i 和 j 为常量，即相对应位置的项相加。如果有矩阵$[a_{mn}]$和$[b_{nk}]$，$[a_{mn}]$矩阵的列数等于$[b_{nk}]$矩阵的行数，两个矩阵可以相乘，记为：$[a_{mn}]*[b_{nk}]=[c_{mk}]$，矩阵相乘的运算的规则是：$c_{ij}=\Sigma(a_{it}+b_{tj})$，其中，i 和 j 为常量，t 为变量，初始值为 1，最大值为 n。

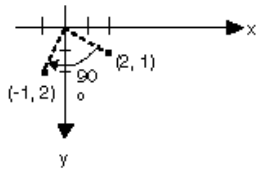
如果将平面中的点视为 1×2 矩阵，则可通过将该点乘以 2×2 变换矩阵来变形该点。下图是点(2,1)在 X 轴按比例 3 放大，Y 轴不变。

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 1 \end{bmatrix}$$



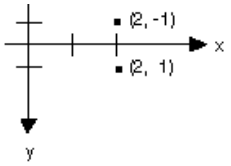
下图表示点(2,1)旋转了 90 度。

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & 2 \end{bmatrix}$$

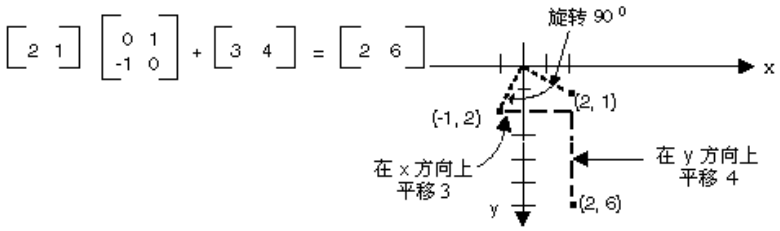


下图表示点(2,1)以 x 轴为对称轴的新点。

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 2 & -1 \end{bmatrix}$$



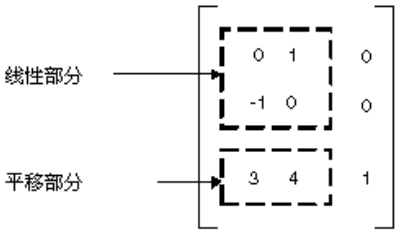
假定要从点(2,1)开始，将其旋转 90 度，在 x 方向将其平移 3 个单位，在 y 方向将其平移 4 个单位。可通过先使用矩阵乘法再使用矩阵加法来完成此操作。



如果用矩阵[2 1 1]代表点(2,1)，使用一个 3×3 变换矩阵，可以用一个矩阵乘法代替以上的两个矩阵运算，见下图：

$$\begin{bmatrix}2 & 1 & 1\end{bmatrix}\begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 3 & 4 & 1\end{bmatrix}=\begin{bmatrix}2 & 6 & 1\end{bmatrix}$$

注意运运结果的矩阵[2 6 1]代表点(2,6)，即点(2,1)映射到了点(2,6)。这个 3×3 矩阵叫作仿射矩阵，Matrix 类中用这个仿射矩阵记录增加的各种变换操作。它和前边的两个 2×2 矩阵的关系如下图，其中第三列固定为 0、0、1。



Matrix 类增加了一些方法处理这个仿射矩阵，主要包括：逆转方法 Invert、相乘方法 Multiply、重置为单位矩阵方法 Reset 等。

5.9 图形文件格式

在磁盘中存储图形和图像的文件格式有多种。GDI+支持以下图形文件格式。

- 位图文件(.bmp)：
  位图文件是 Windows 使用的一种标准格式，用于存储设备无关和应用程序无关的图像。BMP 文件通常不压缩，因此不太适合 Internet 传输。
- 可交换图像文件格式(.gif)：
  GIF 是一种用于在 Web 页中显示图像的通用格式。GIF 文件是压缩的，但是在压缩过程中没有信息丢失，解压缩的图像与原始图像完全一样。GIF 文件中的一种颜色可以被指定为透明，这样，图像将具有显示它的任何 Web 页的背景色。在单个文件中存储一系列 GIF 图像可以形成一个动画 GIF。GIF 文件每个像素颜色最多用 8 位表示，所以它们只限于使用 256 种颜色。
- JPG 文件(.jpg)：
  JPEG 是联合摄影专家组提出的一种适应于自然景观(如扫描的照片）的压缩方案。一些信息会在压缩过程中丢失，但是这些丢失人眼是察觉不到的。JPEG 文件每像素颜色用 24 位表示，因此能够显示超过 16,000,000 种颜色。JPEG 文件不支持透明或动画。JPEG 图像中的压缩级别是可以控制的，但是较高的压缩级别（较小的文件）会导致丢失更多的信息。对于一幅以 20:1 压缩比生成的图像，人眼难以把它和原始图像区别开来。JPEG 是一种压缩方案，不是一种文件格式。“JPEG 文件交换格式(JFIF)”是一种文件格式，

常用于存储和传输根据 JPEG 方案压缩的图像。Web 浏览器显示的 JFIF 文件使用.jpg 扩展名。

- 可移植网络图形(.PNG)
  PNG 格式不但保留了许多 GIF 格式的优点，还提供了超出 GIF 的功能。像 GIF 文件一样，PNG 文件在压缩时也不损失信息。PNG 文件能以每像素 8、24 或 48 位来存储颜色，并以每像素 1、2、4、8 或 16 位来存储灰度。相比之下，GIF 文件只能使用每像素 1、2、4 或 8 位。PNG 文件还可为每个像素存储一个透明度 alpha 值，该值指定了该像素颜色与背景颜色混合的程度。PNG 优于 GIF 之处在于它能够逐渐显示一幅图像，也就是说，当图像通过网络连接到达时显示将越来越近似。PNG 文件可包含伽玛校正和颜色校正信息，以便图像可在各种各样的显示设备上精确地呈现。
- 图元文件(.emf)：
  GDI+提供 Metafile 类，以便能够记录和显示图元文件。图元文件，也称为矢量图像，是一种存储为一系列绘图命令和设置的图像。Metafile 对象记录的命令和设置可以存储在内存中或保存到文件或流。下面示例在主窗体显示了一个图元文件的图形。
  ```
  Graphics g=this.CreateGraphics();
  Metafile myMetafile=new Metafile("SampleMetafile.emf");
  myGraphics.DrawImage(myMetafile,10,10);//图形左上角的位置是(10,10)
  ```
- 支持的文件格式还有：图标文件(.ico)、.EXIF、.TIFF、.ICON、.WMF 等

5.10 图形框 PictureBox 控件

PictureBox 控件常用于图形设计和图像处理程序，又称为图形框，该控件可显示和处理的图像文件格式有：位图文件(.bmp)、图标文件(.ico)、GIF 文件(.gif)和 JPG 文件(.jpg)。其常用的属性、事件和方法如下：

- 属性 Image：指定要显示的图像，一般为 Bitmap 类对象。
- 属性 SizeMode：指定如何显示图像，枚举类型，默认为 Normal，图形框和要显示的图像左上角重合，只显示图形框相同大小部分，其余不显示；为 CentreImage，将图像放在图形框中间，四周多余部分不显示；为 StretchImage，调整图像大小使之适合图片框。
- 方法 CreateGraphics()：建立 Graphics 对象。
- 方法 Invalidate()：要求控件对参数指定区域重画，如无参数，为整个区域。
- 方法 Update()：方法 Invalidate()并不能使控件立即重画指定区域，只有使用 Update()方法才能立即重画指定区域。使用见 5.10.4 节中的鼠标移动事件处理函数。

例子 e5_10：使用 PictureBox 控件显示图像

(1)新建项目。放 PictureBox 控件到窗体。属性 Name=pictureBox1。
(2)放 Button 控件到窗体。属性 Name=button1。
(3)放 OpenFileDialog 控件到窗体。属性 Name=openFileDialog1。
(4)可以在设计阶段修改属性 Image 为指定图形文件，设定初始显示的图像。
(5)button1 控件事件处理函数如下:

```
private void button1_Click(object sender, System.EventArgs e)
{   if(openFileDialog1.ShowDialog()==DialogResult.OK)
    {   Bitmap p1=new Bitmap(openFileDialog1.FileName);//Bitmap类见下节
        pictureBox1.Image=p1;
    }
}
```

5.11 Bitmap 类

System.Drawing 命名空间有一个类 Image，用来处理图像。Image 类的派生类 Bitmap 类封装了 GDI+中的位图，可以处理由像素数据定义的图像。Image 类的派生类 metafile 处理元文件，此类文件用记录绘图命令的方法存储图像。

5.11.1 Bitmap 类支持的图像类型

使用 Bitmap 类可以显示和处理多种图像文件，可处理的文件类型及文件扩展名如下：扩展名为.bmp 的位图文件、扩展名为.ico 的图标文件、扩展名为.gif 的 GIF 文件、扩展名为.jpg 的 JPG 文件。当使用构造函数 Bitmap(string FileName)建立 Bitmap 类对象时，如果文件是以上类型，将自动转换为位图格式存到 Bitmap 类对象中。可使用 Bitmap 类方法 Save(string FileName,ImageFormat imageFormat)把 Bitmap 类对象中的位图存到文件中，其中第 1 个参数是选定的文件名，第 2 个参数是指定文件存为那种类型，可以是如下类型：System.Drawing.Imaging.ImageFormat.bmp(或.ico、.gif、.jpg)。

5.11.2 Bitmap 类的方法

- 方法 SetPixel()：画点方法，前 2 个参数是指定点的位置，第 3 个参数是颜色值。
- 方法 GetPixle()：得到指定点的颜色，2 个参数是指定点的位置，返回颜色值。
- 有多个构造函数例如：new Bitmap(”图像文件名”)，new Bitmap(宽,高)等。
- 方法 Save()：第 1 个参数是文件名，第 2 个参数是指定文件存为那种类型，可以是如下类型：System.Drawing.Imaging.ImageFormat.bmp(或.ico、.gif、.jpg)。
- 方法 Dispose()：释放位图对象

5.11.3 画点

例子 e5_11_3：用 SetPixel 画点，GetPixle 得到指定点的颜色。放 Button 和 PictureBox 控件到主窗体。增加 Button 控件单击事件函数如下：

```
private void button1_Click(object sender,System.EventArgs e)
{   pictureBox1.Width=720;//设定 pictureBox1 的宽和高
    pictureBox1.Height=110;
    Bitmap bits=new Bitmap(720,110);//建立位图对象，宽=720，高=110
    int x,y;
    for(x=0;x<720;x++)//画正弦曲线
    {   y=(int)(50+50*Math.Sin((3.14159/180.0)*x));
        bits.SetPixel(x,y,Color.Red);
    }
    pictureBox1.Image=bits;//位图对象在 pictureBox1 中显示
    Color c1=bits.GetPixel(20,20);
    string s="R="+c1.R+",G="+c1.B+",G+"+c1.G;
```

```
    MessageBox.Show(s);
}
```

5.11.4 在 PictureBox 中画任意曲线

例子 e5_11_4：例子 5_7_1C 实现了画任意曲线程序，用 ArrayList 类对象记录绘制的曲线。在该程序中增加橡皮功能、图像的拷贝、图像的剪贴、图像的粘贴比较困难，也不能和画图程序交换文件。为了实现这些功能，用图形框(PictureBox 控件)显示绘制图形。绘制图形必须存在图形框属性 Image 引用的位图对象中，图形框显示的图像被破坏，图形框响应 Paint 事件，将用其属性 Image 引用的位图对象恢复所绘制的图形。仅将图形绘制在图形框表面，图形框响应 Paint 事件，绘制的图形将不能被恢复，也就是说，绘制在图形框表面的图形丢失了。具体实现步骤如下：

(1) 新建项目。增加 4 个私有变量:private bool mark=false; private Point point;
    private Bitmap bits; private Graphics bitG;

(2) 放 PictureBox 控件到窗体，修改属性 Dock=Fill。

(3) 在构造函数中增加语句：

```
   //建立位图对象，宽和高为指定值
   bits=new Bitmap(pictureBox1.Width,pictureBox1.Height);
   bitG=Graphics.FromImage(bits);//得到位图对象的 Graphics 类的对象
   bitG.Clear(Color.White);//用白色清除位图对象中的图像
   pictureBox1.Image=bits;//位图对象在 pictureBox1 中显示
```

(4) 为控件 PictureBox 事件 OnMouseDown，OnMouseUp，OnMouseMove 增加事件处理函数：

```
private void pictureBox1_MouseDown(object sender,//鼠标按下事件处理函数
                                   System.Windows.Forms.MouseEventArgs e)
{   if(e.Button==MouseButtons.Left)//是否是鼠标左键按下
    {   point.X=e.X;
        point.Y=e.Y;//画线段开始点
        mark=true;//鼠标左键按下标识
    }
}
private void pictureBox1_MouseMove(object sender,//鼠标移动事件处理函数
                                   System.Windows.Forms.MouseEventArgs e)
{   if(mark)//如果鼠标左键按下
    {   Graphics g=pictureBox1.CreateGraphics();
        Pen pen1=new Pen(Color.Black);
        g.DrawLine(pen1,point.X,point.Y,e.X,e.Y);//图形画在PictureBox表面
        bitG.DrawLine(pen1,point.X,point.Y,e.X,e.Y);//图形画在位图对象bits中
        EndPoint.X=e.X;
        EndPoint.Y=e.Y;//下次绘制画线段开始点
    }
}
private void pictureBox1_MouseUp(object sender,
                                   System.Windows.Forms.MouseEventArgs e)
{   mark=false;
```

```
    pictureBox1.Image=bits;//保存了所画的图形
}
```
(5) 运行，在 PictureBox 控件拖动鼠标可以画线。最小化后再最大化后，图形不消失。

5.11.5 存取位图文件

例子 e5_11_5：为上例增加存取位图文件功能。

(6) 把 Mainmenu 控件放到主窗体中。增加菜单：文件，属性 Name=menuItemFile。为文件菜单增加菜单项：新建、打开、另存为、退出，属性 Name 分别为 menuItemNew、menuItemOpen、menuItemSaveAs、menuItemExit。

(7) 为主窗体菜单项新文件增加单击事件函数如下：

```
private void menuItemNew_Click(object sender, System.EventArgs e)
{   bitG.Clear(Color.White);//用白色清空位图对象bitG
    pictureBox1.Image=bits;//pictureBox1显示用白色清空位图对象bitG
}
```
(8) 把 OpenFileDialog 控件放到窗体中。为主窗体菜单项打开文件增加单击事件函数如下：

```
private void menuItemOpen_Click(object sender,System.EventArgs e)
{   if(openFileDialog1.ShowDialog(this)==DialogResult.OK)
    {   bits.Dispose();//撤销 bitG 所引用的对象
        bits=new Bitmap(openFileDialog1.FileName);//建立指定文件的新位图对象
        bitG=Graphics.FromImage(bits);//得到位图对象使用的 Graphics 类对象
        pictureBox1.Image=bits;
    }
}
```
(9) 把 SaveFileDialog 控件放到子窗体中。为主窗体菜单项另存为增加单击事件函数如下：

```
private void menuItemSaveAs_Click(object sender,System.EventArgs e)
{   if(saveFileDialog1.ShowDialog(this)==DialogResult.OK)
    {   string s=saveFileDialog1.FileName+".bmp";
        bits.Save(s,System.Drawing.Imaging.ImageFormat.Bmp);
    }
}//也可以存为其它格式，例如：Jpg，Gif 等。请读者试一下。
```
(10)为主窗体菜单项退出增加单击事件函数如下：

```
private void menuItemExit_Click(object sender, System.EventArgs e)
{   Close();}
```
(11)运行，在 PictureBox 控件拖动鼠标可以画线。存所画的图形到文件，再重新读出该文件，看是否正常运行。

5.11.6 用拖动鼠标方法画椭圆或圆

例 5_11_6：画笔程序中，拖动鼠标方法画椭圆或圆，拖动鼠标时显示椭圆或圆的轮廓，鼠标抬起时，按指定条件画椭圆或圆。如果图形仅画在图形框(PictureBox 控件)上，而不保存到其属性 Image 引用的位图对象中，当调用图形框的 Invalidate()方法，图形框响应 Paint 事件，用图形框属性 Image 引用的位图对象恢复图像，将擦除仅画在图形框上的图形。

拖动鼠标方法画椭圆或圆时，仅将椭圆或圆画在 PictureBox 上，在鼠标拖动到下一个位置，用图形框的 Invalidate()方法将前一位置所画的图形擦除。实现步骤如下：

(1) 新建项目。增加 5 个私有变量:private bool mark=false;private Point StartPoint; private Bitmap bits;private Graphics bitG;private Point EndPoint;

(2) 放 PictureBox 控件到子窗体。修改属性 Dock=Fill。

(3) 在构造函数中增加语句：

```
//bits 用来保存 pictureBox1 中位图图像
bits=new Bitmap(pictureBox1.Width,pictureBox1.Height);
bitG=Graphics.FromImage(bits);
bitG.Clear(Color.White);
pictureBox1.Image=bits;
```

(4) 在 Form1 类中增加 MakeRectangle 方法返回由参数指定的两个点定义的矩形。方法如下：

```
private Rectangle MakeRectangle(Point p1,Point p2)
{   int top,left,bottom,right;
    top=p1.Y<=p2.Y? p1.Y:p2.Y;//计算矩形左上角点的 y 坐标
    left=p1.X<=p2.X? p1.X:p2.X;//计算矩形左上角点的 x 坐标
    bottom=p1.Y>p2.Y? p1.Y:p2.Y;//计算矩形右下角点的 y 坐标
    right=p1.X>p2.X? p1.X:p2.X;//计算矩形右下角点的 x 坐标
    return(new Rectangle(left,top,right-left,bottom-top));//返回矩形
}
```

(5) 为 PictureBox 事件 OnMouseDown、OnMouseUp、OnMouseMove 增加事件处理函数如下：

```
private void pictureBox1_MouseDown(object sender,//鼠标按下事件处理函数
                                   System.Windows.Forms.MouseEventArgs e)
{   if(e.Button==MouseButtons.Left)
    {   StartPoint.X=e.X;
        StartPoint.Y=e.Y;
        EndPoint.X=e.X;
        EndPoint.Y=e.Y;
        mark=true;
    }
}
private void pictureBox1_MouseMove(object sender,//鼠标移动事件处理函数
                                   System.Windows.Forms.MouseEventArgs e)
{   if(mark)
    {   Rectangle r1=MakeRectangle(StartPoint,EndPoint);//计算重画区域
        r1.Height+=2;
        r1.Width+=2;//区域增大些
        pictureBox1.Invalidate(r1);//擦除上次鼠标移动时画的图形，r1为擦除区域
        pictureBox1.Update();//立即重画，即擦除
        Graphics g=pictureBox1.CreateGraphics();
        Pen pen1=new Pen(Color.Black);
        EndPoint.X=e.X;
        EndPoint.Y=e.Y;
        r1=MakeRectangle(StartPoint,EndPoint);//计算椭圆新位置
```

```
        g.DrawEllipse(pen1,r1);//在新位置画椭圆
    }
}
private void pictureBox1_MouseUp(object sender,//鼠标抬起事件处理函数
                                 System.Windows.Forms.MouseEventArgs e)
{   Pen pen1=new Pen(Color.Black);
    EndPoint.X=e.X;
    EndPoint.Y=e.Y;
    Rectangle r1=MakeRectangle(StartPoint,EndPoint);
    bitG.DrawEllipse(pen1,r1);
    mark=false;
    pictureBox1.Image=bits;
}
```

(6) 运行，在 PictureBox 控件中拖动鼠标可以画圆或椭圆。

5.12 图像剪贴板功能

Windows 中的许多程序都支持剪贴板功能。通过剪贴板可以完成显示数据的剪贴（Cut），复制（Copy），粘贴（Paste）等功能。剪贴板可以理解为一块存储数据的公共区域，用户可以用菜单项复制（Copy）或剪贴（Cut）把数据放入到剪贴板中，当本任务或其它任务要用剪贴板中的数据时，可以用菜单项粘贴（Paste）从剪贴板中把数据取出。存入剪贴板中的数据，可以是字符，位图，或者其它格式数据。在图形模式下使用剪贴板包括如下动作：选定剪贴区域、剪贴（Cut）、复制（Copy）、粘贴（Paste）等。使过画图程序的读者都知道，在使用剪贴和复制前，必须首先选定剪贴或复制区域，首先按一个按钮，通知程序要选定剪贴或复制区域，然后在要选定区域的左上角按下鼠标左键，拖动鼠标画出一个矩形，抬起鼠标后显示一个矩形既为要选定剪贴或复制区域。剪贴或复制后，矩形自动消失。下面详细介绍实现以上功能的方法。

5.12.1 剪贴区域选定

剪贴区域选定的方法和前边章节中拖动鼠标方法绘制椭圆或圆的方法基本一样，只是在这里绘制的是矩形，而且在鼠标抬起时，不把矩形存入 PictureBox 控件属性 Image 引用的位图对象中，仅仅记录矩形的位置。请读者自己实现此功能。

5.12.2 剪贴板复制功能的实现

假定已选定剪贴区域，例如为区域 Rectangle(10,10,50,50)，把此区域的图形或图像放到剪贴板中。具体实现步骤如下：

(1) 新建项目。放 PictureBox 控件到窗体，修改属性 Dock=Fill。属性 Name=pictureBox1，修改属性 Image，使其显示一幅图。

(2) 把 Mainmenu 控件放到主窗体中。增加顶级菜单项：编辑，属性 Name=menuItemEdit。为编辑弹出菜单增加菜单项：复制、剪贴、粘贴。属性 Name 分别为 menuItemCopy、

menuItemCut、menuItemPaste。

(3) 为窗体菜单项复制增加单击事件函数如下：

```
private void menuItemCopy_Click(object sender, System.EventArgs e)
{   Bitmap myBitmap=new Bitmap(pictureBox1.Image);
    Rectangle cloneRect=new Rectangle(10,10,50,50);
    System.Drawing.Imaging.PixelFormat format=myBitmap.PixelFormat;
    Bitmap cloneBitmap=myBitmap.Clone(cloneRect,format);
    Clipboard.SetDataObject(cloneBitmap);
}
```

(4) 运行，选中复制菜单项，复制图形到剪贴板。打开画图程序，选中画图程序粘贴菜单项，可以看到被复制的图形能正确粘贴到画图程序中。

5.12.3 剪贴板剪贴功能的实现

(5) 剪贴是先复制，再把选中区域图形清除，菜单项剪贴单击事件处理函数如下：

```
private void menuItemCut_Click(object sender,System.EventArgs e)
{   menuItemCopy_Click(sender,e);//调用复制菜单项单击事件处理函数
    Bitmap bits=new Bitmap(50,50);//建立位图对象，宽和高为选中区域大小
    Graphics g=Graphics.FromImage(bits);//得到位图对象的 Graphics 类的对象
    g.Clear(Color.White);//用白色清除位图对象中的图像
    Bitmap myBitmap=new Bitmap(pictureBox1.Image);
    g=Graphics.FromImage(myBitmap);
    g.DrawImage(bits,10,10,50,50);
    pictureBox1.Image=myBitmap;//位图对象在 pictureBox1 中显示，即清除
}
```

(6) 运行，选中剪贴菜单项，拷贝图形到剪贴板，原位置图形被清空为白色，最小化后再最大化，图形不变。打开画图程序，选中画图程序粘贴菜单项，可以看到被拷贝的图形能正确粘贴到画图程序中。

5.12.4 剪贴板粘贴功能的实现

(7) 为窗体菜单项粘贴增加单击事件函数如下：

```
private void menuItemPaste_Click(object sender, System.EventArgs e)
{   IDataObject iData=Clipboard.GetDataObject();//得到剪贴板对象
    if(iData.GetDataPresent(DataFormats.Bitmap))//判断剪贴板有无位图对象
    {   Bitmap bits=(Bitmap)iData.GetData(DataFormats.Bitmap);//得到剪贴板位图
        Bitmap myBitmap=new Bitmap(pictureBox1.Image);
        Graphics g=Graphics.FromImage(myBitmap);
        g.DrawImage(bits,30,30);
        pictureBox1.Image=myBitmap;//位图对象在pictureBox1中显示
    }
}
```

(8) 运行画图程序，选中拷贝菜单项，拷贝图形到剪贴板。运行自己编制的程序，选中粘贴

菜单项，可以看到画图程序中被拷贝的图形能正确粘贴到自己编制的程序中。

(9) 画图程序粘贴后，能用鼠标移动粘贴的图形，现实现此功能。放 PictureBox 控件到窗体，属性 Name=pictureBox2，属性 Visable=false。这里把粘贴后的图形放到 PictureBox2 中，使其可以移动。为 Form1 类增加变量：bool mark=false;int x=0,y=0;

为 pictureBox2 控件的事件 OnMouseDown，OnMouseUp，OnMouseMove 增加事件函数如下：

```
private void pictureBox2_MouseDown(object sender,
                                   System.Windows.Forms.MouseEventArgs e)
{   mark=true;
    x=e.X;
    y=e.Y;
}
private void pictureBox2_MouseMove(object sender,
                                   System.Windows.Forms.MouseEventArgs e)
{   if(mark)
    {   int x1,y1;
        x1=e.X-x;
        y1=e.Y-y;
        pictureBox1.Invalidate();//擦除上次鼠标移动时画的图形
        pictureBox1.Update();//立即重画，即擦除
        pictureBox2.Left+=x1;
        pictureBox2.Top+=y1;
        x=e.X;//原来没有此 2 句
        y=e.Y;
    }
}
private void pictureBox2_MouseUp(object sender,
                                   System.Windows.Forms.MouseEventArgs e)
{   mark=false;}
```

(10)修改窗体菜单项粘贴单击事件函数如下：

```
private void menuItemPaste_Click(object sender,System.EventArgs e)
{   IDataObject iData=Clipboard.GetDataObject();
    if(iData.GetDataPresent(DataFormats.Bitmap))
    {   Bitmap bit=(Bitmap)iData.GetData(DataFormats.Bitmap);
        pictureBox2.Width=bit.Width;//阴影为修改部分
        pictureBox2.Height=bit.Height;
        pictureBox2.Image=bit;
        pictureBox2.Top=pictureBox1.Top;
        pictureBox2.Left=pictureBox1.Left;
        pictureBox2.Parent=pictureBox1;
        pictureBox2.Visible=true;
    }
}
```

(11)在 pictureBox1 控件任意位置单击鼠标，表示已将粘贴图像拖到指定位置，需将粘贴图像粘贴到 pictureBox1 控件。为 pictureBox1 控件的事件 OnMouseDown 增加事件函数如下：

```
private void pictureBox1_MouseDown(object sender,
                                   System.Windows.Forms.MouseEventArgs e)
{   if(pictureBox2.Image!=null&&pictureBox2.Visible)
    {   Bitmap bits=new Bitmap(pictureBox2.Image);
        Bitmap myBitmap = new Bitmap(pictureBox1.Image);
        Graphics g=Graphics.FromImage(myBitmap);
        g.DrawImage(bits,pictureBox2.Left,pictureBox2.Top);
        pictureBox1.Image=myBitmap;//位图对象在 pictureBox1 中显示
        pictureBox2.Visible=false;
    }
}
```

(12)运行画图程序，选中拷贝菜单项，拷贝图形到剪贴板。运行自己编制的程序，选中粘贴菜单项，可以看到画图程序中被拷贝的图形能正确粘贴到自己编制的程序中。拖动被拷贝的图形，使其运动到指定位置，在 pictureBox2 外，单击鼠标右键，图形固定到指定位置。

5.13 图像的处理

本节介绍图像的处理的最基础知识，要想深入了解这方面的知识，还要读这方面的专著。

5.13.1 图像的分辨力

例子 e5.13.1：将原图形的分辨率降低 16 倍，其方法是将原图形分成 4*4 的图形块，这 16 个点的颜色都置成这 16 个点中某点的颜色，例如 4*4 的图形块左上角的颜色。

(1) 新建项目。放两个 PictureBox 控件到窗体，属性 Name 分别为 pictureBox1，pictureBox2，修改 pictureBox1 属性 Image，使其显示一幅图。

(2) 放 Button 控件到窗体，为其增加事件处理函数如下：

```
private void button1_Click(object sender,System.EventArgs e)
{   Color c;
    int i,j,size,k1,k2,xres,yres;
    xres=pictureBox1.Image.Width;//pictureBox1 显示的图像的宽
    yres=pictureBox1.Image.Height;//pictureBox1 显示的图像的高
    size=4;
    pictureBox2.Width=xres;//令 pictureBox2 和 pictureBox1 同宽，同高。
    pictureBox2.Height=yres;
    Bitmap box1=new Bitmap(pictureBox1.Image);
    Bitmap box2=new Bitmap(xres,yres);
    for(i=0;i<=xres-1;i+=size)
    {   for(j=0;j<=yres-1;j+=size)
        {   c=box1.GetPixel(i,j);
            for(k1=0;k1<=size-1;k1++)
            {   for(k2=0;k2<=size-1;k2++)
                box2.SetPixel(i+k1,j+k2,c);
```

```
            }
        }
    }
    pictureBox2.Image=box2;
}
```

(3) 运行，单击按钮，在 PictureBox2 中可以看到分别率低的图形。

5.13.2 彩色图像变换为灰度图像

例子 e5.13.2：本例把彩色图像变换为灰度图像。其方法是将原彩色图形每一个点的颜色取出，求出红色、绿色、蓝色分量的平均值，即(红色+绿色+蓝色)/3，作为这个点的红色、绿色、蓝色分量，这样就把彩色图像变成了灰度图像。具体步骤如下：

(1) 新建项目。放两个 PictureBox 控件到窗体，属性 Name 分别为 pictureBox1，pictureBox2，修改 pictureBox1 属性 Image，使其显示一幅图。

(2) 放 Button 控件到窗体，为其增加事件函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   Color c;
    int i,j,xres,yres,r,g,b;
    xres=pictureBox1.Image.Width;
    yres=pictureBox1.Image.Height;
    pictureBox2.Width=xres;
    pictureBox2.Height=yres;
    Bitmap box1=new Bitmap(pictureBox1.Image);
    Bitmap box2=new Bitmap(xres,yres);
    for(i=0;i<xres;i++)
    {   for(j=0;j<yres;j++)
        {   c=box1.GetPixel(i,j);
            r=c.R;
            g=c.G;
            b=c.B;
            r=(r+g+b)/3;
            c=Color.FromArgb(r,r,r);
            box2.SetPixel(i,j,c);
        }
    }
    pictureBox2.Image=box2;
}
```

(3) 运行，单击按钮，在 PictureBox2 中可以看到黑白图形。

5.13.3 灰度图像处理

例子 e5.13.3：将一幅灰度图像变换为另一幅灰度图像，两幅灰度图形的灰度满足如下关系：设图 1 和图 2 的灰度分别为 d1 和 d2，如 d1<85，d2=0；如 85<=d1<=170，d2=(d1-85)*3；

如 d1>170，d2=255。变换的效果是增强了对比度。具体步骤如下：

(1) 新建项目。放两个 PictureBox 控件到窗体，属性 Name 分别为 pictureBox1，pictureBox2，修改 pictureBox1 属性 Image，使其显示一黑白幅图。

(2) 放 Button 控件到窗体，为其增加事件函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   Color c;
    int i,j,xres,yres,m;
    xres=pictureBox1.Image.Width;
    yres=pictureBox1.Image.Height;
    pictureBox2.Width=xres;
    pictureBox2.Height=yres;
    Bitmap box1=new Bitmap(pictureBox1.Image);
    Bitmap box2=new Bitmap(xres,yres);
    int[] lut=new int[256];
    for(i=0;i<85;i++)
        lut[i]=0;
    for(i=85;i<=170;i++)
        lut[i]=(i-85)*3;
    for(i=171;i<256;i++)
        lut[i]=255;
    for(i=0;i<xres;i++)
    {   for(j=0;j<yres;j++)
        {   c=box1.GetPixel(i,j);
            m=lut[c.R];
            c=Color.FromArgb(m,m,m);
            box2.SetPixel(i,j,c);
        }
    }
    pictureBox2.Image=box2;
}
```

(3) 运行，单击按钮，在 PictureBox2 中可以看到对比度增强的黑白图形。

5.13.4 动画

学习了以上知识，制作一些简单动画是比较容易的。例如，如果一段动画，要求一个动画小人从窗体左侧走到右侧，如何实现呢？首先，为了看到人在走动，应该有 3 个动作：右脚在前，左脚在后；两脚并排；左脚在前，右脚在后。因此应制作 3 幅图画，表示这三个动作。每当转换一幅图画，图画应在 X 轴方向右移一步的距离。将 3 幅图画放到 3 个 PictureBox 控件中，用定时器产生中断，中断处理程序负责使其中一幅图画显示，其余两幅不显示，同时，修改 PictureBox 控件属性 Left，使其在正确的位置上。这样就可以看到人的走动了。请读者自己完成这段动画。

习题

(1)使用 PictureBox 控件显示图像，修改属性 SizeMode 为不同值，例如 pictureBox1.SizeMode=PictureBoxSizeMode.StretchImage 看一下效果。
(2) 实现画图程序的橡皮功能。
(3) 有时为了很快找到一幅图像，把很多图像都压缩后在窗体中并排显示，如希望更仔细的查看某幅图像，单击这幅压缩图像，放大这幅图像。请实现此功能。
(4) 实现设定剪贴板剪贴区域为矩形的功能。
(5) 实现设定剪贴板剪贴区域为任意封闭曲线的功能。(提示：使用 GraphicsPath 类)
(6) 如何将 PictureBox 控件显示图像存为其它格式文件，例如：Jpg，Gif 等。
(7) 完成 5.11.4 所要求的动画，并能走到窗体右边界后，从右向左走回来，到左边界后，再向右走回去。如果有背景，如何处理。
(8) 制作自己的画图程序，看一看能完成那些功能。
(9) 有些时候，为突出图形的分界，例如医院的 X 片时黑白的，为了使医生能更清楚的看到肿瘤，将黑白图形变为彩色的图形，在分界两侧用不同颜色表示，这种方法叫伪彩色。实现的原理，就是在黑白图像灰度变化很大处，认为是边界。试一下，能否实现黑白图像的伪彩色。

第六章 文件和流

编程语言在如何处理输入/输出问题方面已经经过了很多变革。早期语言，例如 Basic 语言，使用 I/O 语句。后来的语言，例如 C 语言，使用标准的 I/O 库(stdio.h)。在 C++和 Java 语言中，引入了抽象的概念：流。流的概念不仅可用于文件系统，也可用于网络。但在 C++和 Java 语言中流的概念比较复杂。C#语言也采用了流的概念，但是使用起来要简单的多。本章介绍 C#语言中，如何处理目录和文件夹，如何处理文件，如何使用流的概念读写文件。

6.1 用流读写文件

C#把每个文件都看成是顺序的字节流，用抽象类 Stream 代表一个流，可以从 Stream 类派生出许多派生类，例如 FileStream 类，负责字节的读写，BinaryRead 类和 BinaryWrite 类负责读写基本数据类型，如 bool、String、int16、int 等等，TextReader 类和 TextWriter 类负责文本的读写。本节介绍这些类的用法。

6.1.1 用 FileStream 类读写字节

写字节代码段如下：

```
byte[] data=new byte[10];
For(int i=0;i<10;i++)
data[i]=(byte)i;
System.IO.FileStream fs=new System.IO.FileStream("g1",FileMode.OpenOrCreate);
fs.Write(data,0,10);
```

读字节代码段如下：

```
byte[] data=new byte[10];
System.IO.FileStream fs=new System.IO.FileStream("g1",FileMode.OpenOrCreate);
fs.Seek(-5,SeekOrigin.End);
int n=fs.Read(data,0,10);//n 为所读文件字节数
```

6.1.2 用 BinaryReader 和 BinaryWriter 类读写基本数据类型

C#中除了字节类型以外，还有许多其它基本数据类型，例如，int、bool、float 等等，读写这些基本数据类型需要使用 BinaryReader 和 BinaryWriter 类。写 int 类型数据代码段如下：

```
System.IO.FileStream fs=new System.IO.FileStream("g1",FileMode.OpenOrCreate);
System.IO.BinaryWrite w=new System.IO. BinaryWrite(fs);
```

```
For(int i=0;i<10;i++)
    w.Write(i);
w.Close();
```

读 int 类型数据代码段如下：

```
int[] data=new int[10];
System.IO.FileStream fs=new System.IO.FileStream("g1", FileMode. OpenOrCreate);
System.IO.BinaryReader r=new System.IO. BinaryReader(fs);
For(int i=0;i<10;i++)
    data[i]=r.ReadInt();
r.Close();
```

6.1.3 用 StreamReader 和 StreamWriter 类读写字符串

读写字符串可以用 StreamReader 和 StreamWriter 类。写字符串类型数据代码段如下：

```
System.IO.FileStream fs=new System.IO.FileStream("g1", FileMode. OpenOrCreate);
System.IO.StreamWrite w=new System.IO.StreamWrite(fs);
w.Write(100);
w.Write("100 个");
w.Write("End of file");
w.Close();
```

读字符串代码段如下：

```
String[] data=new String[3];
System.IO.FileStream fs=new System.IO.FileStream("g1", FileMode. OpenOrCreate);
System.IO.StreamReader r=new System.IO.StreamReader(fs);
For(int i=0;i<3;i++)
    data[i]=r.ReadLine();
r.Close();
```

6.2 File 类和 FileInfo 类

C#语言中通过 File 和 FileInfo 类来创建、复制、删除、移动和打开文件。在 File 类中提供了一些静态方法，使用这些方法可以完成以上功能，但 File 类不能建立对象。FileInfo 类使用方法和 File 类基本相同，但 FileInfo 类能建立对象。在使用这两个类时需要引用 System.IO 命名空间。这里重点介绍 File 类的使用方法。

6.2.1 File 类常用的方法

- AppendText：返回 StreamWrite，向指定文件添加数据；如文件不存在，就创建该文件。
- Copy：复制指定文件到新文件夹。
- Create：按指定路径建立新文件
- Delete：删除指定文件。
- Exists：检查指定路径的文件是否存在，存在，返回 true。

- GetAttributes：获取指定文件的属性。
- GetCreationTime：返回指定文件或文件夹的创建日期和时间。
- GetLastAccessTime：返回上次访问指定文件或文件夹的创建日期和时间。
- GetLastWriteTime：返回上次写入指定文件或文件夹的创建日期和时间。
- Move：移动指定文件到新文件夹。
- Open：返回指定文件相关的 FileStream，并提供指定的读/写许可。
- OpenRead：返回指定文件相关的只读 FileStream。
- OpenWrite：返回指定文件相关的读/写 FileStream。
- SetAttributes：设置指定文件的属性。
- SetCretionTime：设置指定文件的创建日期和时间。
- SetLastAccessTime：设置上次访问指定文件的日期和时间。
- SetLastWriteTime：设置上次写入指定文件的日期和时间。

下面通过程序实例来介绍其主要方法：

6.2.2 文件打开方法：File.Open

该方法的声明如下：public static FileStream Open(string path, FileMode mode)。下面的代码打开存放在 c:\Example 目录下名称为 e1.txt 文件，并在该文件中写入 hello。

```
FileStream TextFile=File.Open(@"c:\ Example\e1.txt",FileMode.Append);
byte [] Info={(byte)'h',(byte)'e',(byte)'l',(byte)'l',(byte)'o'};
TextFile.Write(Info,0,Info.Length);
TextFile.Close();
```

6.2.3 文件创建方法：File.Create

该方法的声明如下：public static FileStream Create(string path)。下面的代码演示如何在 c:\Example 下创建名为 e1.txt 的文件。

```
FileStream NewText=File.Create(@"c:\Example\e1.txt");
NewText.Close();
```

6.2.4 文件删除方法：File.Delete

该方法声明如下：public static void Delete(string path)。下面的代码演示如何删除 c:\Example 目录下的 e1.txt 文件。

```
File.Delete(@"c:\Example\e1.txt");
```

6.2.5 文件复制方法：File.Copy

该方法声明如下：

```
public static void Copy(string sourceFileName,string destFileName,bool overwrite);
```

下面的代码将 c:\Example\e1.txt 复制到 c:\Example\e2.txt。由于 Cope 方法的 OverWrite 参数设为 true，所以如果 e2.txt 文件已存在的话，将会被复制过去的文件所覆盖。

File.Copy(@"c:\Example\e1.txt",@"c:\Example\e2.txt",true);

6.2.6 文件移动方法：File.Move

该方法声明如下：public static void Move(string sourceFileName,string destFileName);

下面的代码可以将 c:\Example 下的 e1.txt 文件移动到 c 盘根目录下。**注意：只能在同一个逻辑盘下进行文件转移。如果试图将 c 盘下的文件转移到 d 盘，将发生错误。**

File.Move(@"c:\Example\BackUp.txt",@"c:\BackUp.txt");

6.2.7 设置文件属性方法：File.SetAttributes

方法声明如下：public static void SetAttributes(string path,FileAttributes fileAttributes);
下面的代码可以设置文件 c:\Example\e1.txt 的属性为只读、隐藏。

```
File.SetAttributes(@"c:\Example\e1.txt",
                   FileAttributes.ReadOnly|FileAttributes.Hidden);
```

文件除了常用的只读和隐藏属性外，还有 Archive(文件存档状态)，System(系统文件)，Temporary(临时文件)等。关于文件属性的详细情况请参看 MSDN 中 FileAttributes 的描述。

6.2.8 判断文件是否存在的方法：File.Exist

该方法声明如下：public static bool Exists(string path);下面的代码判断是否存在 c:\Example\e1.txt 文件。

```
if(File.Exists(@"c:\Example\e1.txt"))//判断文件是否存在
{…}//处理代码
```

6.2.9 得到文件的属性

用下面的代码可以得到文件的属性，例如文件创建时间、最近访问时间、最近修改时间等等。

```
FileInfo fileInfo=new FileInfo(“file1.txt”);
string s=fileInfo.FullName+” 文件长度=”+fileInfo.Length+”, 建立时间=”+fileInfo.CreationTime+”;
```

也可用如下代码：

```
string s=” 建立时间=”+File.File.GetCreationTime(“file1.txt”)+” 最后修改时间=”+File.GetLastWriteTime(“file1.txt”)+”访问时间=”+File.GetLastAccessTime(“file1.txt”);
```

6.3 Directory 类和 DirectoryInfo 类

C#语言中通过 Directory 类来创建、复制、删除、移动文件夹。在 Directory 类中提供了

一些静态方法，使用这些方法可以完成以上功能。但 Directory 类不能建立对象。DirectoryInfo 类使用方法和 Directory 类基本相同，但 DirectoryInfo 类能建立对象。在使用这两个类时需要引用 System.IO 命名空间。这里重点介绍 Directory 类的使用方法。

6.3.1 Directory 类常用的方法如下：

- CreateDirectory：按指定路径创建所有文件夹和子文件夹。
- Delete：删除指定文件夹。
- Exists：检查指定路径的文件夹是否存在，存在，返回 true。
- GetCreationTime：返回指定文件或文件夹的创建日期和时间。
- GetCurrentDirectory：获取应用程序的当前工作文件夹。
- GetDirectories：获取指定文件夹中子文件夹的名称。
- GetDirectoryRoot：返回指定路径的卷信息、根信息或两者同时返回。
- GetFiles：返回指定文件夹中子文件的名称。
- GetFileSystemEntries：返回指定文件夹中所有文件和子文件的名称。
- GetLastAccessTime：返回上次访问指定文件或文件夹的创建日期和时间。
- GetLastWriteTime：返回上次写入指定文件或文件夹的创建日期和时间。
- GetLogicalDrives：检索计算机中的所有驱动器，例如 A:、C：等等。
- GetParent：获取指定路径的父文件夹，包括绝对路径和相对路径。
- Move：将指定文件或文件夹及其内容移动到新位置。
- SetCreationTime：设置指定文件或文件夹的创建日期和时间。
- SetCurrentDirectory：将应用程序的当前工作文件夹设置指定文件夹。
- SetLastAccessTime：设置上次访问指定文件或文件夹的日期和时间。
- SetLastWriteTime：设置上次写入指定文件夹的日期和时间。

6.3.2 目录创建方法：Directory.CreateDirectory

方法声明如下：public static DirectoryInfo CreateDirectory(string path);下面的代码演示在 c:\Dir1 文件夹下创建名为 Dir2 子文件夹。

```
Directory.CreateDirectory(@"c:\Dir1\Dir2");
```

6.3.3 目录属性设置方法：DirectoryInfo.Atttributes

下面的代码设置 c:\Dir1\Dir2 目录为只读、隐藏。与文件属性相同，目录属性也是使用 FileAttributes 来进行设置的。

```
DirectoryInfo DirInfo=new DirectoryInfo(@"c:\Dir1\Dir2");
DirInfo.Atttributes=FileAttributes.ReadOnly|FileAttributes.Hidden;
```

6.3.4 目录删除方法：Directory.Delete

该方法声明如下：public static void Delete(string path,bool recursive);下面的代码可以将 c:\Dir1\Dir2 目录删除。Delete 方法的第二个参数为 bool 类型，它可以决定是否删除非空目

录。如果该参数值为 true，将删除整个目录，即使该目录下有文件或子目录；若为 false，则仅当目录为空时才可删除。

Directory.Delete(@"c:\Dir1\Dir2",true);

6.3.5 目录移动方法：Directory.Move

该方法声明如下：public static void Move(string sourceDirName,string destDirName);下面的代码将目录 c:\Dir1\Dir2 移动到 c:\Dir3\Dir4。

File.Move(@"c:\Dir1\Dir2",@"c:\Dir3\Dir4");}

6.3.6 获取当前目录下所有子目录：Directory.GetDirectories

该方法声明如下：public static string[] GetDirectories(string path;);下面的代码读出 c:\Dir1\目录下的所有子目录，并将其存储到字符串数组中。

```
string [] Directorys;
Directorys = Directory. GetDirectories (@"c:\Dir1");
获得所有逻辑盘符：
string[] AllDrivers=Directory.GetLogicalDrives();
```

6.3.7 获取当前目录下的所有文件方法：Directory.GetFiles

该方法声明如下：public static string[] GetFiles(string path;);下面的代码读出 c:\Dir1\目录下的所有文件，并将其存储到字符串数组中。

```
string [] Files;
Files = Directory. GetFiles (@"c:\Dir1",);
```

6.3.8 判断目录是否存在方法：Directory.Exist

该方法声明如下：public static bool Exists(string path;);下面的代码判断是否存在 c:\Dir1\Dir2 目录。

```
if(File.Exists(@"c:\Dir1\Dir2"))//判断目录是否存在
  {…}//处理语句
```

注意：路径有 3 种方式，当前目录下的相对路径、当前工作盘的相对路径、绝对路径。以 C:\dir1\dir2 为例(假定当前工作目录为 C:\Tmp)。“dir2”，“\dir1\dir2”，“C:\dir1\dir2”都表示 C: \dir1\dir2。另外，在 C#中 “\”是特殊字符，要表示它的话需要使用“\\”。由于这种写法不方便，C#语言提供了@对其简化。只要在字符串前加上@即可直接使用“\”。所以上面的路径在 C#中应该表示为”dir2”，@”\dir1\dir2”，@”C:\dir1\dir2”。

6.4 例子：查找文件

6.4.1 Panel 和 ListView 控件

6.4.2 在指定文件夹中查找文件

Windows 操作系统提供了一个查找文件的程序，可以查找指定文件夹中的指定文件，本例也实现了同样的功能。具体实现步骤如下：

(1) 新建项目。

(2) 放 Panel 控件到窗体，属性 Dock=Left。Panel 控件可以把窗体分割为多个部分，这里将窗体分割为左右两部分。

(3) 在 Panel 控件中增加两个 Label 控件，属性 Text 分别为”要搜索的文件或文件夹”和”搜索范围”。

(4) 在 Panel 控件中增加一个 TextBox 控件，属性 Name=textBox1，属性 Text 为空，用来输入要搜索的文件或文件夹。

(5) 在 Panel 控件中增加一个 TextBox 控件，属性 Name=textBox2，属性 Text 为空，用来输入搜索范围。在其后增加一个 Button 控件，属性 Name=Broswer，属性 Text=”浏览”。

(6) 为”浏览”按钮增加事件函数如下：

```
private void Broswer_Click(object sender, System.EventArgs e)
{
    OpenFileDialog dlg=new OpenFileDialog();
    if(dlg.ShowDialog()==DialogResult.OK)
    {
      textBox2.Text=dlg.FileName;
    }
}
```

(7) 在 Panel 控件中增加一个 Button 控件，属性 Name 分别为 Start 和 Stop，属性 Text 分别为”开始搜索”和”停止搜索”。

(8) 放分割器控件 Splitter 到窗体，属性 Dock=Left。

(9) 在分割器控件右侧放置视图控件 ListView，属性 Dock=Right,属性 SmallImgeList =”imageList”，属性 View=”Detail”。点击属性 Column 右侧标题为…的按钮，在弹出的 ColumnHeader 编辑对话框中添加 4 个列头，属性 Name 分别为：FileName、FileDirectory、FileSize 和 LastWriteTime，属性 Text 分别为：名称、所在文件夹、大小和修改时间。

(10)为窗体增加一个方法：FindFiles(DirectoryInfo dir,string FileName)，该方法是在第一个参数指定的文件夹中查找第二个参数指定的所有文件。在一个文夹中可能还有子文件夹，子文件夹中可能还有子文件夹，因此要在第一个参数指定的文件夹中和其子文件夹中查找第二个参数指定的所有文件。为了实现能够查找所有文件夹中的同名文件，采用递归调用方法，如果在一个文件夹中存在子文件夹，在一次调用函数自己，查找子文件夹中的文件。具体实现代码如下：

```
void FindFiles(DirectoryInfo dir,string FileName)
```

```
    {
     FileInfo[] files=dir.GetFiles(FileName);//查找所有文件并在 ListView 中显示
     If(files.Length!=0)
     {
        foreach(FileInfo aFile in files)
        {
            ListViewItem lvi;
            lvi=new    ListViewItem(aFile.Name,aFile.Directory.FullName,aFile.Length.ToString,
    aFile.LastWriteTime.ToShortDateString());
            lvi.ImageIndex=0;
            listView1.Items.Add(lvi);
        }
     }
        DirectoryInfo[] dirs=dir.GetDirectories();//查找子文件夹中的匹配文件
        If(dirs.Length!=0)
        {
            foreach(DirectoryInfo aDir in dirs)
            {
                FindFiles(aDir,FileName);
            }
        }
    }
```

(11)为”开始搜索”按钮增加事件函数如下：

```
private void Start_Click(object sender, System.EventArgs e)
{
 DirectoryInfo aDir=CreateDirectorie(comboBox1.Text);
 FindFiles(aDir,textBox1.Text);
}
```

(12)为”停止搜索”按钮增加事件函数如下：

```
private void Stop_Click(object sender, System.EventArgs e)
{
}
```

(13)编译、运行，

6.5 例子：拆分和合并文件

在将一个文件作为电子邮件的附件传送时，由于附件的大小有限制，可以将较大的文件分割为较小的多个文件，传送后再合并为一个文件，下边两个方法实现文件的拆分和合并。首先是拆分方法，参数 1 时要拆分的文件名，参数 2 是拆分后的文件名，文件名后边由拆分方法自动增加序号，参数 3 是被拆分后的文件大小。拆分方法定义如下：

```
void SplitFile(string f1,string f2,int f2Size)
{
    FileStream inFile=new FileStream(f1,FileMode.OpenOrCreate,FileAccess.Read);
    bool mark=true;
```

```
    int i=0;
    int n=0;
    byte[] buffer=new byte[f2Size];
    while(mark)
    {
        FileStream OutFile=new FileStream(f2+i.ToString+".fsm",
                                    FileMode.OpenOrCreate,FileAccess.Read);
        if((n=inFile.Read(buffer,0,f2Size))>0)
        {
            OutFile.Write(buffer,0,n);
            i++;
            OutFile.Close();
        }
        else
        {
            mark=false;
        }
    }
    inFile.Close();
}
```

合并文件方法，参数 1 时要合并的文件名，参数 2 是被拆分的文件名，文件名后边有序号，要将这些文件合并到一起，参数 3 是要合并的文件数。合并方法定义如下：

```
void MergeFile(string f1,string f2,int f2Num)
{
  FileStream OutFile=new FileStream(f1,FileMode.OpenOrCreate,FileAccess.Write);
  int n,l;
  for(int i=0;i<f2Num;i++)
  {
    FileStream InFile=new
      FileStream(f2+i.ToString+".fsm",FileMode.OpenOrCreate,FileAccess.Read);
      l=InFile.Length;
      byte[] buffer=new byte[l];
      n=inFile.Read(buffer,0,l);
      OutFile.Write(buffer,0,n);
      InFile.Close();
  }
  OutFile.Close();
}
```

习题：

(1) 实现拆分和合并文件的完整程序。

第七章 多线程程序设计

如果在一个程序中，有多个工作要同时做，可以采用多线程。在 Windows 操作系统中可以运行多个程序，把一个运行的程序叫做一个进程。一个进程又可以有多个线程，每个线程轮流占用 CPU 的运算时间，Windows 操作系统将时间分为许多个时间片，一个线程使用一个时间片后，操作系统将此线程挂起，将另一个线程唤醒，使其使用下一个时间片，操作系统不断的把线程挂起，唤醒，再挂起，再唤醒，如此反复，由于现在 CPU 的速度比较快，给人的感觉象是多个线程同时执行。Windows 操作系统中有很多这样的例子，例如复制文件时，一方面在进行磁盘的读写操作，同时一张纸不停的从一个文件夹飘到另一个文件夹，这个飘的动作实际上是一段动画，两个动作是在不同线程中完成的，也就是说两个动作是同时完成的。又如 Word 程序中的拼写检查也是在另一个线程中完成的。每个进程最少有一个线程，叫主线程，是进程自动创建的，每进程可以创建多个线程。

不同语言和操作系统对线程提供了不同支持，编写多线程应用程序的方法也不尽相同。例如，VB6 没有提供对线程的支持，程序员不能处理自己的线程。VC++6.0 开发人员必须充分理解 Windows 线程和处理模型的复杂性，同时拥有这种线程模型的强大功能。C++程序员可以创建出多线程程序，但必须学习掌握很多复杂的技巧，以确保线程在自己的控制之下。

.NET Framework 提供了一个完整而功能强大的线程模型，该模型允许编程人员精确控制线程中运行的内容，线程何时退出，以及它将访问多少数据等。所以，在.NET 中，既提供了 C++的强大功能，又具有 VB6 的简单性。

7.1 线程类(Thread)的属性和方法

线程类在命名空间 System..Threading 中定义的，因此如果要创建多线程，必须引入命名空间 System..Threading。Thread 类的常用方法如下：

- 属性 ThreadPriority：设置线程优先级，有 5 种优先级类别：(AboveNormal)稍高、(BelowNormal)稍低、Normal(中等，默认值)、Highest(最高)和 Lowest(最低)。
- 构造函数：new Thread(new ThreadStart(线程首先执行的方法名))，构造方法参数中指定的方法需要程序员自己定义，在这个方法中完成线程要完成的任务，退出该方法，线程结束。该方法必须为公有 void 类型的方法，不能有参数。
- 方法 Start()：建立线程类对象后，线程并不能自动运行，用这个方法启动线程。
- 方法 IsAlive()：判断线程对象是否存在，=true，存在。
- 方法 Abort()：撤销线程对象。不能撤销一个已不存在的线程对象，因此在撤销一个线程对象前，必须用方法 IsAlive()判断线程对象是否存在。
- 方法 Sleep()：参数为毫秒，线程暂时停止参数指定的时间，允许其它线程运行。
- 方法 Suspend()：线程挂起。如只是暂时停止线程的运行，可用此函数将线程挂起。必须用 Resume()方法唤醒线程。
- 方法 Resume()：恢复挂起线程。如希望继续运行挂起线程，可用此方法唤醒线程。需要注意的是，如果线程多次被挂起，调用一次 Resume()方法就可以把线程唤醒。

7.2 线程的创建

例子:多线程程序设计，该程序包括一个子线程，在标签控件中显示子线程运行的时间。增加 4 个按钮，分别单击按钮，可以建立、挂起、恢复和停止线程。

(1) 新建项目。在窗体中放置 4 个按钮和一个标签控件，属性 Name 分别为 button1、button2、button3、button4 和 label1，按钮属性 Text 分别为新线程、挂起、恢复和撤销。button1 属性 Enabled=true，其余按钮的属性 Enabled=false。

(2) 在 Form1.cs 头部增加语句：using System.Threading。

(3) 为 Form1 类定义一个线程类变量：private Thread thread;

(4) 为新线程按钮(button1)增加单击事件函数如下：

```
thread= new Thread(new ThreadStart(fun);//生成线程类对象
label1.Text="0";
thread.Start();
button1.Enabled=false;
button2.Enabled=true;
button3.Enabled=false;
button4.Enabled=true;
```

(5) 为挂起按钮(button2)增加单击事件函数如下：

```
thread. Suspend();
button1.Enabled=false;
button2.Enabled=false;
button3.Enabled=true;
button4.Enabled=false;
```

(6) 为恢复按钮(button3)增加单击事件函数如下：

```
thread. Resume();
button1.Enabled=false;
button2.Enabled=true;
button3.Enabled=false;
button4.Enabled=true;
```

(7) 为撤销按钮(button4)增加单击事件函数如下：

```
if(thread.IsAlive())
{
    thread.Abort();//撤销线程对象
    button1.Enabled=true;
    button2.Enabled=false;
    button3.Enabled=false;
    button4.Enabled=false;
}
```

(8) C#线程模型由于允许将任何一个原型为 void 类型的公有类成员方法（静态或非静态）作为线程方法，因此它实际上允许在任何一个类（不要求这个类是某个类的子类）中实现线程方法，而且同一个类中可以实现多个线程方法。为 Form1 类定义一个方法如下：

```
public void fun()//在线程中执行的方法，必须为公有 void 类型方法，不能有参数。
{
```

```
    while(true)//死循环，线程将一直运行
    {
    int x=Convert.ToInt(label1.Text);
    x++;
    label1.Text=Convert.ToString(x);
    thread.Sleep(1000);//休眠 1 秒钟，休眠一次，线程运行了 1 秒钟
    }
}
```

(9) 编译，运行，按新线程(Button1)按钮，新线程开始，计数器从 0 开始计数。按挂起(Button2)按钮，线程暂停，计数器也暂停。按恢复(Button3)按钮，线程重新启动，计数器也重新计数。按撤销(Button4)按钮，线程对象被撤销，线程对象不存在，计数器停止计数。

7.3 建立线程类

有时需要建立多个线程，每个线程要实现的功能基本相同，但有个别参数不同，例如，每个线程完成同样的任务，但控制的对象不同。线程构造函数参数指定的方法需要自己定义，在这个方法中完成一些任务，但该方法不能有参数，因此不能通过方法的参数传递不同设置。为解决这个问题，可以定义一个线程类。具体实现方法件下例。

例子：建立两个线程，分别控制进度条(ProgressBar)控件，每个进度条的速度不一样。首先介绍进度条(ProgressBar)控件。

7.3.1 进度条(ProgressBar)控件

进度条(ProgressBar)控件经常用来显示一个任务的进度。有时，要在后台完成一个长时间的任务，例如一个软件的安装，如果没有任何提示，使用者可能分不清任务是在进行中，还是死机了，为了让用户知道安装正在进行，可以使用进度条控件显示一个安装进度。进度条控件常用的属性如下：

- 属性 Maximum：进度条所代表的整数最大值，等于此值，任务完成。默认值 100。
- 属性 Minimum：进度条所代表的整数最小值，等于此值，任务开始。默认值 0。
- 属性 Step：变化的步长，默认值为 10。
- 属性 Value：进度条当前位置代表的值。修改该值，达到一个 Step，进度增加一格。

7.3.2 用线程控制进度条

例子实现的具体实现步骤如下：

(1) 新建项目。在 Form1.cs 头部增加语句：using System.Threading。

(2) 在窗体中放置 2 个进度条(ProgressBar)控件和一个标签控件，属性 Name 分别为 progressBar1、progressBar2 和 label1。Label1 的属性 Text=""。

(3) 在文件 Form1.cs 的最后建立线程类如下：

```
public class myThread
{
    private int SleepTime;
```

```
    private ProgressBar progressBar;
    private Thread thread1;
    public myThread(int Time,ProgressBar p1)
    {
        SleepTime=Time;
        progressBar=p1;
        thread1=new Thread(new ThreadStart(Fun));
        Thread1.Start();
    }
    public void fun()
    {
        while(progressBar.Value!=100)
        {
            progressBar.Value+=1;
            thread1.Sleep(SleepTime);
        }
        if(label.Text=="")
            label1.Text="第一个线程结束";
        else
            label.Text+=",第二个线程结束";
    }
}
```

(4) 为 Form1 类增加变量：myThread　myThread1,myThread2。

(5) 为 Form1 类构造函数增加语句如下：

```
myThread1=new myThread(100,progressBar1);
myThread2=new myThread(200,progressBar2);
```

(6) 编译，运行，可以看到两个进度条以不同的速度前进，当进度条被添满，线程停止。

7.4 线程的优先级

当一个程序被调入内存准备运行时，操作系统自动创建一个进程和一个主线程，并为进程指定基本优先级，进程基本优先级分为以下四种：

- IDLE_PROCESS_CLASS　　系统空闲时才执行
- NORMAL_PROCESS_CLASS　　系统默认进程优先级
- BELOW_NORMAL_PROCESS_CLASS　比系统默认进程优先级低一级
- ABOVE_NORMAL_PROCESS_CLASS　比系统默认进程优先级高一级
- HIGH_PROCESS_CLASS　　高进程优先级
- REALTIME_PROCESS_CLASS　　进程最高(实时)优先级

操作系统一般分配用户应用程序进程为NORMAL_PROCESS_CLASS 系统默认进程优先级。对于进程中的各个子线程，可以修改属性 ThreadPriority 来调整其的优先级，属性 ThreadPriority 可以取如下值：

- THREAD_PRIORITY_IDLE　　系统空闲时才执行
- THREAD_PRIORITY_LOWEST　　比 NORMAL 低 2 级
- THREAD_PRIORITY_LOWER　　比 NORMAL 低 1 级

- THREAD_PRIORITY_NORMAL　　系统默认线程优先级
- THREAD_PRIORITY_HIGHT　　比 NORMAL 高 1 级
- THREAD_PRIORITY_HIGHTEST　　比 NORMAL 高 2 级
- THREAD_PRIORITY_TIME_CRITICAL　线程最高(实时)优先级

一个线程的优先权并不是越高越好，应考虑到整个进程中所有线程以及其他进程的情况做出最优选择。优先级相同的线程按照时间片轮流运行。优先级高的线程先运行，只有优先级高的线程停止、休眠或暂停时，低优先级线程才能运行。

7.5 多个线程互斥

多个线程同时修改同一个共享数据可能发生错误，例如，两个线程记录不同入口进入的人数，用一个变量实时显示总人数。每个线程都要对这个总人数变量执行加 1 操作，这个加 1 操作是一个高级语言语句，可能包含若干机器语言语句，例如，可能先从内存取数，加 1，再存回内存。假如，当前人数为 100，第一个线程运行，从内存取出总人数 100，时间片时间到，第 2 个线程启动，执行加 1 操作，总人数变为 101，第 2 个线程退出运行。第一个线程恢复运行，执行加 1 操作，存回内存，总数本应为 102，实际为 101，少计算了一个。为了防止此类错误，在一个线程操作这个总人数变量时，不允许其它线程对它进行操作，这叫线程的互斥。

7.5.1 多个线程同时修改共享数据可能发生错误

例子 e7_5_1：下边的例子模拟多个线程同时修改同一个共享数据发生的错误。
(1) 新建项目。在 Form1.cs 头部增加语句：using System.Threading。
(2) 在窗体中放置一个标签控件，属性 Name=label1。
(3) 为 Form1 类定义 2 个线程类变量：Thread thread1,thread2。定义 1 个整形变量：int num=0。
(4) 为 Form1 类构造函数增加语句如下：

```
thread1= new Thread(new ThreadStart(Fun1);
thread2= new Thread(new ThreadStart(Fun2);
```

(5) 为 Form1 类定义 Fun1()和 Fun2()方法如下：

```
public void Fun1()
{
    int k,n;
    for(k=0;k<4;k++)
    {
        n=num;
        n++;
        thread1.Sleep(100);
        num=n;
    }
    label1.Text=Convert.ToString(num);
}
public void Fun2()
{
```

```
    int k,n;
    for(k=0;k<4;k++)
    {
        n=num;
        n++;
        thread2.Sleep(200);
        num=n;
    }
    label1.Text=Convert.ToString(num);
}
```

(6) 编译，运行，标签控件应显示 8，实际运行多次，显示的数要小于 8。

7.5.2 用 LOCK 语句实现互斥

7.5.3 用 Mutex 类实现互斥

7.5.4 用 Monitor 类实现互斥

7.6 Monitor 类

7.7 线程的同步：生产者和消费者关系

在生产者和消费者关系中，生产者线程产生数据，并把数据存到公共数据区，消费者线程使用数据，从公共数据区取出数据，并进行分析。很显然，如果公共数据区只能存一个数据，那么在消费者线程取出数据前，生产者线程不能放新数据到公共数据区，否则消费者线程将丢失数据。同样，只有在生产者线程把数据已经放到公共数据区，消费者线程才能取出数据，如果新数据未放到公共数据区，消费者线程不能取数据。这些就是所谓的生产者和消费者关系，这要求生产者线程和消费者线程同步。

7.7.1 生产者线程和消费者线程不同步可能发生错误

例子 e7_7_1：下边的例子模拟生产者线程和消费者线程不同步可能发生错误。有一个

公共变量，要求生产者线程顺序放 1 到 4 到这个公共变量中，每放一个变量，消费者线程取出这个数求和，最后把和显示出来，显然和应为 10。如不采取同步措施，和的结果不正确。

(1) 新建项目。在 Form1.cs 头部增加语句：using System.Threading。

(2) 在窗体中放置一个标签控件，属性 Name=label1。

(3) 为 Form1 类定义 2 个线程类变量：Thread thread1,thread2。

(4) 为 Form1 类定义 2 个整形变量：int sum=0,x=-1。

(5) 为 Form1 类构造函数增加语句如下：

```
thread1= new Thread(new ThreadStart(Fun1);
thread2= new Thread(new ThreadStart(Fun2);
```

(6) 为 Form1 类定义 Fun1()和 Fun2()方法如下：

```
public void Fun1()//生产数据
{
    int k,n;
    for(k=1;k<5;k++)
    {
        x=i;
        thread1.Sleep(200);
    }
}
public void Fun2()//消费数据
{
    int k,n;
    for(k=0;k<4;k++)
    {
        sum+=x;
        thread2.Sleep(100);
    }
    label1.Text=Convert.ToString(sum);
}
```

(7) 编译，运行，标签控件应显示 10，实际运行多次，显示的数要小于 10。

7.7.2 生产者线程和消费者线程同步的实现

修改上例，为 Form1 类定义 1 个布尔变量：bool mark=false。其值为 false，表示数据还未放到公共数据区(即 x)中，生产者线程可以放数据到公共数据区中，由于没有数据，消费线程不能取数据，必须等待。mark=true，表示数据已未放到公共数据区(即 x)中，消费线程还未取数据，生产者线程不能再放数据到公共数据区中，必须等待。由于有了数据，消费线程可以取数据。修改 Fun1()如下：

```
public void Fun1()//生产数据
{
    int k,n;
    for(k=1;k<5;k++)
    {
        Monitor.Enter(this);
```

```
        If(mark)
            Monitor.Wait(this);//如果消费者数据未取走，生产者等待
        !mark;
        x=i;
        Monitor.Pulse(this);//激活消费者线程
        Monitor.Exit(this);
    }
}
```

修改 Fun2()如下：

```
public void Fun2()//消费数据
{
    int k,n;
    for(k=0;k<4;k++)
    {
        Monitor.Enter(this);
        If(!mark)
            Monitor.Wait(this);//如果生产者未放数据，消费者等待
        !mark;
        sum+=x;
        Monitor.Pulse(this);
        Monitor.Exit(this);
    }
    label1.Text=Convert.ToString(sum);
}
```

编译，运行，标签控件应显示 10。

习题:

(1) 修改例子 e7_5，用线程类实现。

第八章 ADO.NET 与数据操作

8.1 数据库基本概念

数据库系统提供了一种将信息集合在一起的方法。数据库主要由三部分组成：数据库管理系统(DBMS)，是针对所有应用的，例如 ACCESS。数据库本身，按一定的结构组织在一起的相关数据。数据库应用程序，它是针对某一具体数据库应用编制的程序，用来获取，显示和更新数据库存储的数据，方便用户使用。这里讲的就是如何编写数据库应用程序。

常见的数据库系统有：FoxPro，Access，Oracle，SQLserver，Sybase 等。数据库管理系统主要有四种类型：文件管理，层次数据库，网状数据库和关系数据库。目前最流行，应用最广泛的是关系数据库，以上所列举的数据库系统都是关系数据库。关系数据库以行和列的形式来组织信息，一个关系数据库由若干表组成，一个表就是一组相关的数据按行排列，例如一个通讯录就是这样一个表，表中的每一列叫做一个字段，例如通讯录中的姓名，地址，电话都是字段。字段包括字段名及具体的数据，每个字段都有相应的描述信息，例如数据类型，数据宽度等。表中每一行称为一条记录。

数据库可分为本地数据库和远程数据库，本地数据库一般不能通过网络访问，本地数据库往往和数据库应用程序在同一系统中，本地数据库也称为单层数据库。远程数据库通常位于远程计算机上，用户通过网络来访问远程数据库中的数据。远程数据库可以采用两层，三层和四层结构，两层结构一般采用 C/S 模式，即客户端和服务器模式。三层模式一般采用 B/S 模式，用户用浏览器访问 WEB 服务器，WEB 服务器用 CGI，ASP，PHP，JSP 等技术访问数据库服务器，生成动态网页返回给用户。四层模式是在 WEB 服务器和数据库服务器中增加一个应用服务器。利用 ADO.NET 可以开发数据库应用程序。

由于 ADO.Net 的使用，设计单层数据库或多层数据库应用程序使用的方法基本一致，极大地方便了程序设计，因此，这里讨论的内容也适用于后边的 Web 应用程序设计。

8.2 设计连接和不连接数据库应用程序的基本步骤：

设计一个数据库应用程序可以采用连接和不连接方式。所谓连接方式，是数据库应用程序运行期间，一直保持和数据库连接，数据库应用程序通过 SQL 语句直接对数据库操作，例如，查找记录、删除记录、修改记录。所谓不连接方式，是数据库应用程序把数据库中感兴趣的数据读入建立一个副本，数据库应用程序对副本进行操作，必要时将修改的副本存回数据库。设计一个不连接方式数据库应用程序一般包括以下基本步骤：

(1) 建立数据库，包括若干个表，在表中添入数据。
(2) 建立和数据库的连接。
(3) 从数据库中取出感兴趣的数据存入数据集 DataSet 对象，包括指定表和表中满足条件的记录，DataSet 对象被建立在内存中，可以包含若干表，可以认为是数据库在内存中的一个子集。然后断开和数据库的联接。
(4) 用数据绑定的方法显示这个子集的数据，供用户浏览、查询、修改。
(5) 把修改的数据存回源数据库。

设计一个连接方式数据库应用程序一般包括以下基本步骤：

(1) 建立数据库，包括若干个表，在表中添入数据。

(2) 建立和数据库的连接。

(3) 使用查询、修改、删除、更新等 Command 对象直接对数据库操作。

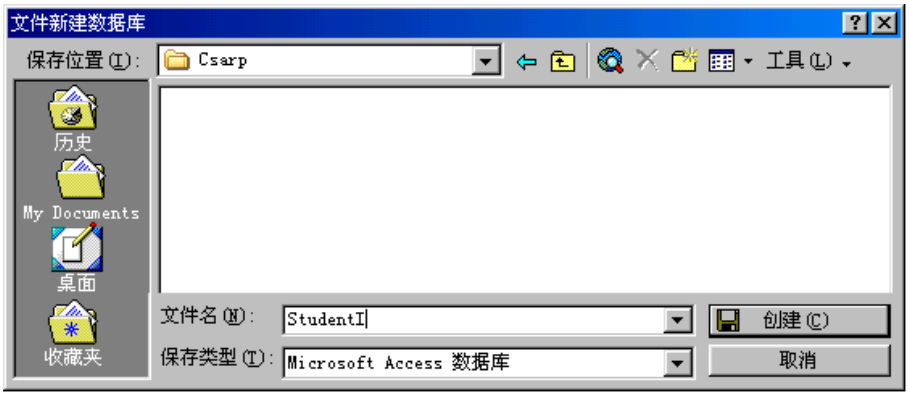
以下章节将按以上步骤说明数据库应用程序的具体设计方法。

8.3 用 ACCESS 创建数据库

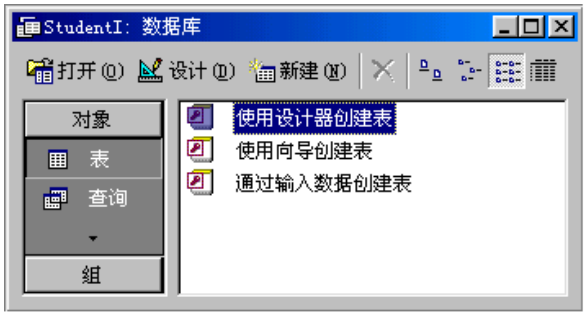
本例创建一个学生信息管理系统，包括两个表，第一个表记录学生的基本情况，包括以下字段：学号、姓名、性别等。第二个表记录学生的学习成绩，包括以下字段：记录编号、课程名称、分数、拥有该课程成绩学生的学号。注意，把学习成绩表字段定义为：学号、语文成绩、数学成绩、物理成绩等字段是不合适的，这样做，增加一门课程，就要增加一个字段，字段要动态增加，这显然不合理。用 Access2000 程序创建数据库具体步骤如下：

(1)运行 microsoft Access2000 程序，出现《microsoft Access》对话框如下图，选择空 Access 数据库，单击确定按钮，打开标题为《文件新建数据库》对话框。

(2)在标题为《文件新建数据库》对话框中，添入数据库文件名：StudentI，选择保存位置和保存类型如下图。单击创建按钮，出现《StudentI:数据库》对话框。



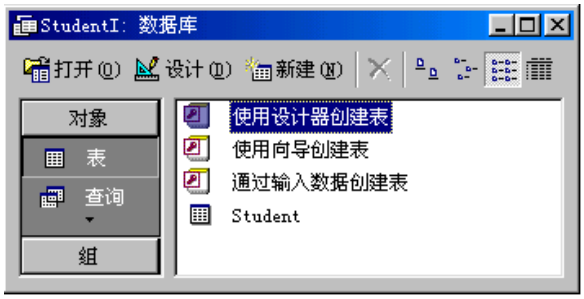
(3)在《StudentI:数据库》对话框中，双击《使用设计器创建数据表》，出现《表 1：表》对话框。在表中可以创建数据库表的字段。



(4)在《表 1:表》对话框中，创建字段 StudentNum，数字类型的整形，必填字段，默认值为 0，标题为学生编号。字段 StudentName，文本，字段大小 8，必填字段，默认值为空，标题为学生姓名。字段 StudentSex，文本，字段大小 2，标题为性别。右击字段 StudentNum，在弹出菜单中选择主键菜单项，设置字段 StudentNum 为主关键字。如下图。

(5)选择《文件》弹出菜单的《保存》菜单项，出现《另存为》对话框，在对话框中的《表名称(N)》编辑框中输入表名：Student，单击确定按钮。关闭《表 1：表》对话框。

(6)在《StudentI:数据库》对话框左侧，选择对象为：表，在右测出现 Student 表，双击 Student 表，出现《Student:表》对话框。



(7)在《Student:表》对话框中为各个字段输入数据：例如：1，张三，男；2，李四，女；3，王五，男；4，鲁六，女。然后存盘。如下图。

(8)同样方法建立表 Score,记录所有学生学习成绩。包括字段 ScoreID(记录编号)，自动编号；ClassName(课程名称),文本，字段大小 26，必填字段，默认值为空；Score(分数)，字节类型，必填字段，默认值为 0；StudentNum(拥有该课程成绩学生的学号),数字类型的整形，必填字段，默认值为空。设置字段 ScoreID 为主关键字。增加若干数据。

(9) 退出 microsoft Access2000 程序。

8.4 结构化查询语言 SQL

用户通过 SQL(Structed Query Language,结构化查询语言)来访问数据库中的数据，使用 SQL 语句可以对数据库中的数据进行查询、增加、删除记录，修改记录中的数据。几乎所有的数据库都支持 SQL 语言，编写数据库应用程序必须学习 SQL 语言。

8.4.1 Select 语句

Select 语句是最常用的语句，可以从数据库的表中获得满足一些条件的数据集。常见的 Select 语句如下：

- select * from student
  从表 Student 表中选择所有字段的所有记录
- select StudentNum,StudentName from student
  从表 Student 表中选择字段 StudentNum 和字段 StudentName 的所有记录
- select * from score where StudentNum=1
  从表 score 表中查找学号 StudentNum=1 同学的所有课程的成绩。

8.4.2 Insert 语句

用于向数据库表中插入一个新记录。例如,向表 student 中插入一个新纪录的语句如下：
Insert student (StudentNum,StudentName,StudentSex)Value(5,"田七","男")

8.4.3 Delete 语句

用于删除数据库表中的一个记录。例如，删除 student 表中学号为 1 的学生，语句如下：
Delete From student where StudentNum=1

8.4.4 Update 语句

更新数据库的 Student 表中学号为 1 的学生名字为"陈七"：

```
Update Student Set StudentName="陈七" Where StudentNum=1。
```

8.5 用 Connection 对象连接数据库。

Connection 类对象用来连接数据库。ADO.NET 中有两类 Connection 对象，一类用于微软的 Sql Server 数据库，该对象连接微软 SQL 数据库时效率较高，另一类用于其它支持 ODBC 的数据库。连接 Sql Server 数据库序应引用如下命名空间：

```
Using System.Data;
using System.Data.SqlClient;
```

连接其它数据库序应引用如下命名空间：

```
Using System.Data;
using System.Data.OleDb;
```

使用 SqlConnection 的例子：

```
string txtConn="DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
SqlConnection conn=new SqlConnection(txtConn);//建立连接
```

其中，DATABASE 为数据库名称，这里为 Northwind，是微软 Sql Server 数据库自带的数据库例子，必须安装此数据库才能使用。UID 为用户名，PWD 为密码，Northwind 数据库安装后的用户名为 sa，密码为空。SERVER 为所使用的数据库服务器，这里数据库服务器和数据库应用程序在同一台计算机中，因此为 localhost，中文意义是本地主机。

使用 OleDbConnection 的例子：

```
string txtConn=
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\\VC#\\studentI.mdb";
OleDbConnection conn = new OleDbConnection(txtConn);//建立连接
```

Provider 为所使用的数据库驱动程序。DataSource 为数据库的位置，有时还增加参数 ConnectionTimeOut，为链接超时时间，默认为 15 秒。

也可以使用 Visual Stutdio.Net 建立连接，例子见 8.10B。

8.6 Command 对象

建立连接后，ADO.Net 通过 Command 对象用 SQL 语句来访问数据库中的数据，对数据库中的数据进行查询，增加、删除记录，修改记录中的数据。具体用法如下：

```
string txtCommand="SELECT * FROM student";
```

使用 OleDbCommand

```
OleDbCommand Command1=new OleDbCommand(txtCommand,conn);
```

使用 SQLCommand

```
SQLCommand Command1=new SQLCommand(txtCommand,conn);
```

例子 8.6：连接方式数据库应用程序。有时，数据库应用程序使用连接方式可能更方便一些，例如，用户的注册信息应该立即存到数据库中。下边的例子模拟用户注册，首先请用户输入个人信息，单击注册按钮，用 SQL 语句把数据存到 StudentI 数据库的 Student 表中。

具体步骤如下：
(1)新建项目。放三个 Label 控件到窗体,修改属性 Text 分别为:学号、姓名、性别。
(2)放三个 TextBox 控件到窗体,修改属性 Text 都为空。TextBox1 输入学号，TextBox2 输入学生姓名，TextBox3 输入学生性别。
(3)引入命名空间 using System.Data.OleDb;
(4)增加变量：OleDbConnection conn;　　OleDbCommand da;
(5)增加一个按钮，属性 Text="增加记录"。界面如下图。为其增加单击事件函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   if(textBox1.Text==""&&textBox2.Text==""&&textBox3.Text=="")
    {   MessageBox.Show("所有项都必须填写！");
        return;
    }//注意，下边语句必须保证数据库文件studentI.mdb在文件夹D:\\vc#中
    string txt1=
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\\vc#\\studentI.mdb";
    string txt2=
        "Insert Into Student(StudentNum,StudentName,StudentSex) Values('";
    txt2+=textBox1.Text + "' , '";
    txt2+=textBox2.Text + "' , '";
    txt2+=textBox3.Text+"')";//最后结果要保证在TextBox中输入的内容被单引号括住
    conn = new OleDbConnection(txt1);
    conn.Open();
    da=new OleDbCommand();
    da.CommandText=txt2;
    da.Connection=conn;
    da.ExecuteNonQuery();
    textBox1.Text="";
    textBox2.Text="";
    textBox3.Text="";
    conn.Close();
}
```

(6)运行，输入学号，姓名，性别，单击《增加记录》按钮，察看数据库，可以看到增加了一个记录。请读者修改为删除一个记录，修改一个记录。

8.7 DataAdapter 对象

DataAdapter 对象包含 4 个 Command 对象:SelectCommand 对象，InsertCommand 对象，UpdateCommand 对象，DeleteCommand 对象，完成对数据库中的数据的选择，插入，更新，删除等功能。DataAdapter 对象隐藏了 Connection 对象和 Command 对象沟通的细节，方便使用，在许多数据库应用中，都使用 DataAdapter 对象。使用 DataAdapter 对象例子如下：

```
string txtConn=
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\\VC#\\studentI.mdb";
OleDbConnection conn=new OleDbConnection(txtConn);//建立连接
string txtCommand="SELECT * FROM student";
OleDbDataAdapter da=new OleDbDataAdapter(txtCommand,conn);
```

8.8 DataSet 对象

DataSet 对象从数据库中取出感兴趣的数据，包括指定表和表中满足条件的记录，DataSet 对象被建立在内存中，可以包含若干表，可以认为是数据库在内存中的一个子集。DataSet 对象只在获取或更新数据时保持和数据库连接，其它时间都是断开的。

8.8.1 使用 DataSet 的优点

在传统的数据库应用程序中，必须建立与数据库的连接，并在数据库应用程序运行过程中保持连接状态。出于各种原因，该方法在许多数据库应用程序中是不实用的。

- 一般情况下，数据库只可以维持少量的并发连接。维持过多并发连接将降低数据库的总体性能，增加数据库应用程序的访问时间。保持四个用户连接的执行也许还可以接受，但连接 100 个用户时可能就不行了。同时访问 Web 数据库的访问者可能非常多，例如大型网上商店，所有访问都保持连接是不现实的。
- 在 Web 应用程序中，浏览器从服务器请求网页，服务器发送该页后，服务器就不再与浏览器有任何连接，直到下一次请求为止。在这种情况下，维持打开的数据库连接是不可行的，因为没有办法知道数据使用者（客户端）是否还将对数据库访问。
- 如果数据库应用程序的多个控件需对数据库数据操作，则多个控件都必须和数据库建立连接，或者为这些控件设计一种方式以相互传递数据。这使问题变得复杂。

出于以上这些原因，ADO.NET 数据库访问被设计为以不连接的数据模型为基础，应用程序只在获取或更新数据时保持连接，其它时间都是断开的。由于数据库并未被大部分时间空闲的连接占用，所以它可以为更多用户服务。

8.8.2 数据集 DataSet 概念

在不连接的数据模型中，每次数据库应用程序需要处理下一条记录时都连接回数据库是不可行的，这样做会大大消除使用不连接数据的优越性。解决方案是临时存储从数据库检索的记录，然后使用该临时集。这便是数据集的概念。数据集 DataSet 是从数据库检索的记录的缓存。数据集 DataSet 中包含一个或多个表（这些表基于源数据库中的表），并且还可以包含有关这些表之间的关系，以及对表包含数据的约束信息。数据集 DataSet 的数据通常是源数据库内容的子集，可以用与操作实际数据库十分类似的方式操作数据集 DataSet，但操作时，将保持与源数据库的不连接状态，使数据库可以自由执行其他任务。

因为数据集 DataSet 是数据库数据的私有子集，所以它不一定反映源数据库的当前状态，因此，需要经常更新数据集 DataSet 中的数据。可以修改数据集 DataSet 中的数据，然后把这些修改写回到源数据库。为了从源数据库获取数据和将修改写回源数据库，请使用数据适配器 DataAdapter 对象。数据适配器 DataAdapter 对象包含更新数据集 DataSet 和将修改写回源数据库的方法。DataAdapter.Fill()方法执行更新数据集 DataSet 操作。DataAdapter.Update()方法执行将修改写回源数据库操作。

尽管数据集是作为从数据库获取的数据的缓存，但数据集与数据库之间没有任何实际关系。数据集是容器，它用数据适配器的 SQL 命令或存储过程填充。

8.8.3 使用 DataSet 对象

使用 DataSet 对象例子如下：

```
string txtConn=
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\\VC#\\studentI.mdb";
OleDbConnection conn=new OleDbConnection(txtConn);//建立连接
string txtCommand="SELECT * FROM student";//将表 student 所有记录从数据库取出
OleDbDataAdapter da=new OleDbDataAdapter(txtCommand,conn);
DataSet MyDataSet=new DataSet();//建立 DataSet 对象 MyDataSet
da.Fill(MyDataSet,"MyTable");//将数据填充到数据集 MyDataSet 中，新表名为：MyTable
```

下例说明了 DataSet 的使用方法：

- 添加记录

```
DataRow dr=MyDataSet.Tables["Student"].NewRow();
dr["StudentNum"]=4;
dr["StudentName"]="鲁豫";
dr["StudentSex"]="女"
```

- 修改 Student 表中第 0 个记录的"StudentName"字段的值为"田歌"

```
MyDataSet.Tables["Student"].Rows.[0]["StudentName"]="田歌";
```

- 删除 Student 表中第 0 个记录

```
MyDataSet.Tables["Student"].Rows.[0].Delete();
```

- 恢复数据

```
if(MyDataSet.HasErros)
      MyDataSet.RejectChanges();
```

- 检查 DataSet 是否有改动

```
if(MyDataSet.HasChanges())
      da.Update(MyDataSet);//更新数据库
```

8.8.4 为 DataSet 对象中的表指定主键、建立关系

为 DataSet 对象中的表指定主键、建立关系，可以保证数据的完整性，例如，主键取值不能重复,不能删除主表中的数据(例如某个学生),而不删除另一个表中和其有关的数据(例如另一个表中的学生成绩)等等。

- 设置表的主键：

```
DataColumn[] pKey=new DataColumn[1];
pKey[0]=MyDataSet.Tables["Student"].Columns["StudentNum"];
MyDataSet.Tables["Student"].PrimaryKey=pKey;
```

- 建立两个表的联系：

```
MyDataSet.Relations.Add("StudentNum",
      MyDataSet.Tables["Student"].Columns["StudentNum"],
            MyDataSet.Tables["Score"].Columns["StudentNum"]);
```

8.9 用 DataGraid 控件显示数据和数据绑定

DataGraid 控件用来按行和列格式显示数据表中的数据。DataGraid 控件属性 DataSource 用来指定数据表所在的数据集 DataSet 对象。DataGraid 控件属性 DataMember 用来指定在数据集 DataSet 对象中要显示的数据表的名字。当这两个属性被正确设定，DataGraid 控件将以网格形式正确显示指定数据表。DataGraid 控件中的数据被修改后，DataSet 对象中相应数据表中的数据也被修改。这叫做数据绑定。数据绑定有两个要点：第一，数据绑定控件能按绑定的数据源正确显示数据源的数据，第二，在数据绑定控件中被修改的数据能被正确写回数据源，这里的数据源一般是数据集 DataSet 对象中的一个表或者是表中的一个字段。许多控件都可以数据绑定。

8.10 不连接数据库应用程序的完整的例子

例 8.10A：

(1) 新建项目。增加语句 using System.Data.OleDb;

(2) 添加控件 DataGraid，属性 Name=dataGrid1

(3) 为 Form1 类增加变量：OleDbConnection conn; OleDbDataAdapter da; DataSet ds;

(4) 为 Form1 类增加一个方法：

```
private void OnLoadData()
{   string txtConn=
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\\vc#\\studentI.mdb";
    conn=new OleDbConnection(txtConn);
    string txtCommand="SELECT StudentName, StudentNum, StudentSex FROM Student";
    da=new OleDbDataAdapter(txtCommand,conn);
    ds=new DataSet("ds");
    da.Fill(ds,"Student");
    dataGrid1.DataSource=ds;
    dataGrid1.DataMember="Student";
}
```

(5) 在构造函数中增加语句

OnLoadData();

(6) 运行，可以看到表 Student 中的信息。运行效果如上图。

例 8.10B：使用 Visual Stutdio.Net 连接数据库 StudentI 并显示 Student 表，具体步骤如下：

(1) 新建项目。在窗体中放置控件 OleDbConnection，其属性 Name=OleDbConnection1。单击控件 oleDbConnection 属性 ConnectionString 的下拉列表的箭头，在列表中选择新建连接，打开《数据链接属性》对话框，选择提供程序选项卡页，选择 OLE DB 提供程序为 Microsoft.Jet.OLEDB.4.0 后，单击下一步按钮，在 1.选择或输入数据库名称下的编辑框中，单击其后按钮，选择数据库 StudentI。在 2.输入登录服务器信息下，选中使用指定的用户名称和密码单选按钮，在用户名称中输入 Admin，选中空白密码多选按钮。单击测试连接按钮，应出现测试连接成功对话框。按确定按钮退出。

(2) 在窗体中放置控件 DataGrid，其属性 Name=dataGrid1。

(3) 选择菜单项视图/服务器资源管理器，打开服务器资源管理器窗口，可以看到新建立的

连接，单击前边的加号，展开此连接树后，再展开表(Table)树，可以看到数据库中的所有表名。拖表 Student 到窗体控件 DataGrid 中，将自动增加控件 oleDbDataAdapter1 到窗体。

(4) 单击 oleDbDataAdapter1 选中它，单击菜单项数据/生成数据集…，打开生成数据集对话框，他选择默认值。按确定按钮退出。自动增加数据集 DataSet 对象 dataSet11。

(5) 修改 DataGrid1 属性 DataSource= dataSet11，属性 Datamember=Student，Student 为表名。此时在 DataGrid1 中可看到表的表头。

(6) 在构造函数中增加语句如下：

```
oleDbDataAdapter1.Fill(dataSet11);
```

(7) 运行，可以在控件 DataGrid1 中看到表 Student 的内容。

(8) 以下将标题改为中文。选中控件 DataGrid1，单击其属性 TableStyles 后的标题为…的按钮，打开 DataGridTableStyle 集合编辑器对话框，单击添加按钮，增加一个 dataGridTableStyle 对象。从属性 MapingName 的下拉列表中选择 Student 表。单击其属性 GridColumnStyles 后的标题为…的按钮，打开 DataGridTextBoxColumn 集合编辑器对话框，单击添加按钮，增加一个 dataGridTextBoxColumn 对象。修改 HeadText 属性为：学生姓名，从属性 MapingName 的下拉列表中选择 Student 表的 StudentName 字段。按此办法，再增加两个 dataGridTextBoxColumn 对象，修改 HeadText 属性分别为：学生学号、学生性别，属性 MapingName 分别为：StudentNum、StudentSex。

(9) 运行，可以在控件 DataGrid1 中看到表头为中文。

8.11 修改数据并保存修改的数据到源数据库

在控件 DataGrid1 中可以修改数据，可以增加记录。现增加一个按钮，标题为：删除，单击此按钮后，将删除控件 DataGrid1 中选定的纪录。为其增加单击事件函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   int x;
    x=dataGrid1.CurrentRowIndex;
    dataSet11.Tables["Student"].Rows[x].Delete();
}
```

运行，选中一个控件 DataGrid 中某行(一条记录)，单击按钮可以删除此行。由于排序等原因，控件 DataGrid 中数据记录的顺序可能和 dataSet11 中表的记录的顺序不一致，因此此方法删除数据可能出出错。修改按钮单击事件函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   int x;
    object z;
    x=dataGrid1.CurrentRowIndex;
    z=dataGrid1[x,1];
    DataRow foundRow;
    object[] findTheseVals = new object[1];
    findTheseVals[0] = z;
    foundRow = dataSet11.Tables["Student"].Rows.Find(findTheseVals);
    if(foundRow != null)
        foundRow.Delete();
}
```

为了保存修改的数据到源数据库，现增加一个按钮，标题为：保存数据，单击此按钮后，将保存修改的数据到源数据库。为其增加单击事件函数如下：

```
private void button2_Click(object sender, System.EventArgs e)
{   if(dataSet11.HasChanges())//在退出程序时，也应用此函数检查，提醒用户是否存盘
        oleDbDataAdapter1.Update(dataSet11);
}
```

运行，增加一个记录，单击第二个按钮，关闭程序，再打开，可以看到新增的记录被显示，说明新增记录已被存到源数据库。

8.12 其它数据绑定控件

本例用 TextBox 控件显示 Student 表，具体步骤如下：

(1) 新建项目。

(2) 从工具箱中，将 3 个 Label 控件放到窗体上，属性 Text 分别为：学号、姓名、性别。

(3) 从工具箱中，将 3 个 TextBox 控件放到窗体上，属性 Text 都为空。

(4) 从“工具箱”的“数据”选项卡中，将 OleDbDataAdapter 对象拖到窗体上。“数据适配器配置向导”启动，它将帮助您创建连接和适配器。

(5) 在该向导中，执行下列操作：

- 在第二个窗格中，创建或选择一个指向数据库 StudentI 的连接。
- 在第三个窗格中，指定您要使用 SQL 语句来访问数据库。
- 在第四个窗格中创建以下 SQL 语句：SELECT * FROM Student
- 单击“完成”完成该向导。

(6) 从“数据”菜单中选择“生成数据集”。 如果看不到“数据”菜单，请在窗体中单击；该窗体必须具有焦点，该菜单才会出现。

(7) 选择“新建”选项，将该数据集命名为 DataSet1。在“选择要添加到数据集中的表”下面的列表中，确保选择了“Student”。 选中“将此数据集添加到设计器”，然后单击“确定”。 从“文件”菜单中选择“全部保存”，存所有文件。

(8) 选中 textBox1 控件的属性 DataBindings，单击前边+号展开，选中 Text 属性，从下拉列表中选择 dataSet11 - Student.StudentNum。同样办法 textBox2 为 dataSet11 - Student.StudentName，textBox3 为 dataSet11 - Student.StudentSex。

(9) 增加主窗体 Form1 的 Load 事件函数如下：

```
private void Form1_Load(object sender, System.EventArgs e)
{   dataSet11.Clear();
    oleDbDataAdapter1.Fill(dataSet11);
}
```

(10)运行，可以看到第一个学生情况。现增加移动记录功能，以显示不同学生情况。

(11)增加变量：BindingManagerBase Navigator;

(12)修改主窗体 Form1 的 Load 事件函数如下：

```
private void Form1_Load(object sender, System.EventArgs e)
{   dataSet11.Clear();
    oleDbDataAdapter1.Fill(dataSet11);
    Navigator=this.BindingContext[dataSet11,"Student"];
}
```

(13)从工具箱中，将 4 个 Button 控件放到窗体上，属性 Text 分别为：第一记录、下一记录、

前一记录、最后记录。各个按钮的单击事件函数如下：

```
private void button1_Click(object sender, System.EventArgs e)
{   Navigator.Position=0;//第一记录
}
private void button2_Click(object sender, System.EventArgs e)
{   if(Navigator.Position!=Navigator.Count-1)
        Navigator.Position+=1;//下一记录
}
private void button3_Click(object sender, System.EventArgs e)
{   if(Navigator.Position!=0)
        Navigator.Position-=1;//前一记录
}
private void button4_Click(object sender, System.EventArgs e)
{   Navigator.Position=Navigator.Count-1;//最后记录
}
```

(14)运行，可以看到第一个学生情况。单击 4 个按钮可以移动记录。

8.13 建立主从关系表

在数据库 StudentI 中，在显示表 Student 和 Score 时，希望选中某个学生时，表 Score 只显示此学生的成绩，两个表的这种关系叫做主从关系。现在实现这两个表的主从关系。具体步骤如下：

(1) 新建项目。

(2) 从“工具箱”的“数据”选项卡中，将 **OleDbDataAdapter** 对象拖到窗体上。“数据适配器配置向导”启动，它将帮助您创建连接和适配器。

(3) 在该向导中，执行下列操作：

- 在第二个窗格中，创建或选择一个指向数据库 StudentI 的连接。
- 在第三个窗格中，指定您要使用 SQL 语句来访问数据库。
- 在第四个窗格中创建以下 SQL 语句：SELECT * FROM Student
- 单击“完成”完成该向导。

(4) 将第二个 OleDbDataAdapter 对象拖到窗体上。“数据适配器配置向导”再次启动。

(5) 重复第(3)步，其中有以下差异：

- 在第二个窗格中，选择上次所使用或创建的同一连接。
- 创建下列 SQL 语句来访问 Score 表：SELECT Score.* FROM Score

(6) 从“数据”菜单中选择“生成数据集”。 如果看不到“数据”菜单，请在窗体中单击；该窗体必须具有焦点，该菜单才会出现。

(7) 选择“新建”选项，将该数据集命名为 DataSet1。在“选择要添加到数据集中的表”下面的列表中，确保选择了“Student”和“Score”。 选中“将此数据集添加到设计器”，然后单击“确定”。从“文件”菜单中选择“全部保存”，存所有文件。

(8) Visual Studio 生成某类型化数据集类（**DataSet1**）和定义该数据集的架构。将在“解决方案资源管理器”中看到新架构（**DataSet1.xsd**）。

(9) 在“解决方案资源管理器”中，双击刚创建的数据集的架构（名为 **DataSet1.xsd**）。“XML 设计器”在“架构”视图中打开，显示数据集内的两个表。

(10)从“工具箱”的“XML 架构”选项卡中，将 **Relation** 对象拖到 Score 表（子表）上。

“编辑关系”对话框打开，其中带有从这两个表中派生的默认值。父元素为 Student 表，子元素为 Score 表，键字段和外键字段都为 StudentNum。其它不修改选默认值。单击“确定”按钮，关闭“编辑关系”对话框。在“XML 设计器”中，这两个表之间显示一个关系图标。如果需要更改关系设置，则可以右击相应的关系，选择“编辑关系”。

(11)保存该架构并关闭 XML 设计器。此刻，为执行从数据库获取信息并转移到数据集的操作所需的全部设置均已完成。可以向窗体添加显示数据的控件了。

(12)返回到创建该项目时已打开的默认窗体 (Form1)。从“工具箱”的“Windows 窗体”选项卡中，将 **DataGrid** 控件拖到窗体上，Name=dataGrid1。RowHeadersVisable=false。在“属性”窗口中，将 **DataSource** 属性设置为 **dataSet11**，将 **DataMember** 属性设置为 Student。

(13)从“工具箱”的“Windows 窗体”选项卡中，将 **DataGrid** 控件拖到窗体上，Name=dataGrid2。在“属性”窗口中，将 **DataSource** 属性设置为 **dataSet11**，将 **DataMember** 属性设置为 Student.StudentScore，设置这两个属性将网格绑定到关系对象，以便网格只包含 dataGrid1 表中选择的学生成绩。

(14)增加主窗体 Form1 的 Load 事件函数如下：

```
private void Form1_Load(object sender, System.EventArgs e)
{   dataSet11.Clear();
    oleDbDataAdapter1.Fill(dataSet11);
    oleDbDataAdapter2.Fill(dataSet11);
}
```

(15)运行，可以看到两个 **DataGrid** 控件，dataGrid1 显示学生情况表，dataGrid2 显示 dataGrid1 表中选中的学生的成绩。在 dataGrid1 表中选择不同学生，dataGrid2 显示相应学生的成绩。

第九章 ASP.Net 编程基础知识

本章首先介绍用 ASP.Net 技术编制服务器端动态网页所需的网络和 HTML 标记语言方面有关的知识。然后介绍 ASP.Net 技术基础。

9.1 网络基础

用 ASP.Net 技术编制服务器端动态网页，必然要和网络打交道，具备一些网络方面的知识是必要的。这里假设读者已经学过计算机基础课程，在此基础上，进一步介绍用 ASP.Net 技术编制服务器端动态网页所需的必备网络基础知识。

9.2.1 局域网、广域网和 INTERNET

把分布在不同地理区域的计算机以及专门的外部设备利用通信线路互连，使各个计算机之间能够相互通讯，实现信息和资源共享，就组成了计算机网络。在一个较小区域，例如在单位内部组成的计算机网络，称为局域网。一个较大区域的计算机网络，称为广域网。为了使各个局域网之间互相通讯，可以把各个局域网连起来，组成广域网。如将全世界范围的计算机网络采用 TCP/IP 网络传输协议联到一起，则组成 INTERNET。

INTERNET 提供了许多服务，例如：远程登录服务 Telnet、文件传送服务 FTP、电子邮件服务 E-mail、电子公告板系统 BBS、万维网 WWW(Web)、电子商务、IP 电话等等。本课程主要介绍万维网 WWW 中服务器端动态网页的设计方法。

9.2.2 网络传输协议

网络的目的是为了通讯，共享资源。通讯即传输数据，为传输数据应遵守一定规则，这个规则叫网络传输协议。不同的网络操作系统采用不同的网络传输协议。而在 INTERNET 中，为了传输数据，大家都必须采用相同的传输协议，即 TCP/IP 协议。

9.2.3 IP 地址

INTERNET 中有成千上万台计算机，它允许任何两台计算机之间进行通讯，为了区分不同的计算机，必须给每一台计算机一个唯一的编号，这个编号叫计算机的 IP 地址，它是一个 32 位二进制数，用四个十进制数表示，中间用点隔开，每个十进制数允许值为 0-255(一个字节)，例如，202.112.10.105，这种记录方法叫点数记法。一个 IP 地址一般由两部分组成，网络标志号及此网络中的计算机号，例如，202.112.10.105 如果是一个 C 类地址，其网络标志号为 202.112.10.0，网络中的计算机号为 105。一个局域网络中，所有计算机中都应有相同的网络标志号，每个计算机有不同的计算机号，两个不同局域网络，其网络标志号必须不同，但不同网络中，主机号可以相同。每个计算机要和 INTERNET 联接，必须有自己

的 IP 地址。

32 位 IP 地址被分为 5 类，即 A、B、C、D 和 E 类地址。A 类地址的第一字节为网络标志号，其余 3 字节为计算机号。B 类地址的前两个字节为网络标志号，其余 2 字节为计算机号。C 类地址的前三个字节为网络标志号，最后一字节为计算机号。D 为特殊地址，E 为私有地址。设置 TCP/IP 时,有一项叫子网掩码，用子网掩码表示 IP 地址是哪类地址，A 类地址的子网掩码是：255.0.0.0，B 类地址的子网掩码是：255.255.0.0,C 类地址的子网掩码是：255.255.255.0。我国大部分单位的 IP 地址为 C 类地址。

9.2.4 域名

用点数法表示的 IP 地址，虽然简单，但很难记忆，为了解决此问题，可以为 INTERNET 网中的每台计算机起一个名字，在 INTERNET 中叫域名，并使此计算机的名字和 IP 地址对应起来，使我们可以使用名字访问计算机，就象我们使用 IP 地址一样。例如微软的域名 www.microsoft.com，清华大学的域名 www.tsinghua.deu.cn。

在 INTERNET 中访问其它计算机必须使用 IP 地址，因此域名必须转换为 IP 地址。实现域名（和 WINDOWS/2000/98 中计算机名有区别）和 IP 地址转换的软件叫 DNS（域名服务器）。在网内有一台计算机运行 DNS 服务器软件，这台计算机叫 DNS 服务器。它负责一定区域内的计算机域名和 IP 地址的转换，这个区域一般是一个网的内部的所有计算机。当网内的计算机用域名和其它计算机通讯时，则首先呼叫 DNS 服务器，DNS 服务器送出此域名对应的 IP 地址，网内的计算机收到 IP 地址后，再利用 IP 地址和其它计算机通讯。如果，本 DNS 不能转换相应的域名，则向上一级 DNS 申请转换。计算机要和 INTERNET 联接，设置 TCP/IP 时,必须设置 DNS 服务器 IP 地址。

9.2.5 URL

我们在用浏览器浏览网页时，实际上是用 URL 来定位一个网页的。URL 是 Uniform Resource Location(统一资源定位器)的简称。他的一般格式是：传输协议://域名:端口号/全路径文件名。http:为 www 专用超文本传输协议。域名，即上边讲到，例如 www.microsoft.com 微软域名，此处可以是 IP 地址，其格式为 http://IP 地址:端口号/全路径文件名。全路径文件名：它指示访问文件的全路径，只写出宿主目录以下的全路径文件名。如果在访问主页时不指定此项，则访问其默认主页，例如当我们在浏览器的 URL（地址）处键入 http://www.sun.com/时，将访问 sun 公司默认主页。当光标移到某关键词，光标变为手型，单击此关键词则显示和此关键词相联系的 URL 所指定的网页。此网页可能在 Internet 中某网站的计算机中。在 www 系统中，每一个网页都有自己的 URL，由它决定此网页在 www 网中的具体位置。它很象计算机文件系统中的文件全路径名。

9.2.6 端口号

一台计算机上可能运行多个服务器软件，如 www 服务器软件（可能不只一个）、ftp 服务器软件等，它们的 IP 地址是相同的。为了区分不同的服务器，为每个服务器编一个号，叫端口号。此项不是必须的，如果一台计算机仅运行一个 www 服务器软件，则一般使用默认端口号 80，运行一个 ftp 服务器软件，其端号为采用默认值 20 或 21 等，如采用默认端口

号，端口号可以不写。但如有多个相同服务器运行，则应指定不同端口，其中不是采用默认端口号值的服务器软件在使用 URL 定位时，则应指明使用的端口号。

9.2.7 HTML，HTTP 和网页

网页使用 HTML 标记语言写成。HTML 标记是用特殊的 ASCII 字符来定义网页中的格式、字体、颜色等内容。由于各种系统中，例如 Windows、Linux、Unix 和苹果系统，都支持 ASCII 字符标准，不同系统中的浏览器都可以解释这些 ASCII 标记，将其所表示的网页在屏幕中显示。这样，不同的系统都可以使用统一的 ASCII 标记，访问其它系统中的网页。网页是 WWW 系统最小传输单位，它是用 HTML 语言写的具有一定格式的数据集合，可供浏览器显示。HTTP 是超文本传输协议，用在 WWW 服务器和浏览器之间传输网页。本质上是 TCP/IP 协议，所有的 WWW 服务器和浏览器都应遵守 HTML 和 HTTP 协议，才能使同一网页在任何计算机中，使用任何浏览器都能显示同一画面，但实际上各公司浏览器软件是有差异的，最明显的是 IE 和 Netscape 之间在显示同一网页时，是有差别的。做好网页后，应用不同浏览器测试一下，看是都能通过

9.2.8 Web 服务器和浏览器工作方式

Web 是基于客户机/服务器模式，运行 Web 服务器软件的计算机叫 Web 服务器，运行浏览器的计算机叫客户机。服务器 24 小时开机，在指定的文件夹(宿主目录)上存贮大量的网页，这些网页用 URL 定位，Web 服务器软件总是在监视是否有浏览器访问自己。客户机用浏览器访问 Web 服务器，在浏览器上键入要访问的网页的 URL，例如：http：//www.sun.com/，用 DNS（域名系统）转换 www.sun.com 域名为 IP 地址，通过 IP 地址和 sun 公司的 Web 服务器通讯，sun 公司的 Web 服务器接到信息后，由于未指定文件名，将默认主页送出。浏览器接到主页数据，将其显示。主页中列出各项主题，每当鼠标经过这些主题时，鼠标变为手形，双击此主题，将和主题有关的网页调入并显示。这种方法叫超链接。超链接的本质是：单击该题目，转换为所联系网页的 URL，在此 URL 中，在文件路径名处，指出了网页在 Web 服务器中路径及文件名，当把这些信息送给 Web 服务器后，Web 服务器就送出指定的网页。

9.2.9 宿主目录、默认主页及网站

默认文档（主页）就是当用户用不带文件名的 URL 访问 www 服务器时送出去的网页。它所在的目录，叫宿主目录。宿主目录下的文件对访问者都是可见的。宿主目录和默认文档名在不同的 Web 服务器中是不同的。Windows2000 的 IIS 服务器宿主目录为…\InetPub\wwwroot,默认文件名（主页名）为 Default.htm。大部分 www 服务器允许修改宿主目录。当用户使用不带文件名的 URL 访问 www 服务器时，并且宿主目录中不存在默认文档，而且 www 服务器允许列出宿主目录下的所有目录列表，将返回宿主目录下的所有目录列表。

由此，可以看出，创建一个网站，就是创建一个文件夹，在文件夹中存入预先设计好的网页。然后，把此文件夹设定为宿主目录，例如 Windows2000 的 IIS 服务器默认的宿主目录为…\InetPub\wwwroot。然后运行 WEB 服务器软件，例如微软的 IIS 服务器软件。网络上的其它计算机就可以通过 URL 访问该计算机中宿主目录中的网页。

9.2.10 静态网页

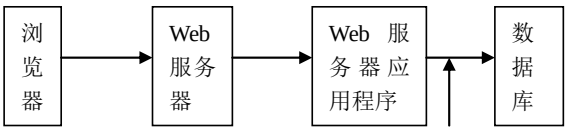
创建一个网站，必须编制若干网页，网站才算建成。静态网页是预先做好的网页，在被访问时不被修改。相对讲静态网页制作比较容易，即使不懂超文本语言，使用 FrontPage，Dreamerware 就可以完成，而动态网页制作则比较困难。

9.2.11 客户端动态网页

例如，网页根据上下午，晚上问客人早上好,下午好和晚上好，又如，网页对于用户输入的内容进行检查，根据用户的选择完成不同的功能。有时，希望在浏览器内实现动画，放映影片等，这些都是客户端动态网页的例子。这可以在 HTML 语言加入 Javascript 或 VBscript 脚本语言来实现。也可以在 HTML 语言嵌入 Java 小程序（Applet）来实现。因此应比较系统的学习一下 HTML 语言，还应该学习 Javascript 或 VBscript 脚本语言，进一步还可以学习 Java 语言。另外使用微软的 AxtiveX 控件也是实现客户端动态网页的一种方法。

9.2.12 服务器端动态网页

举一个客户登记的例子，当客人在客户机端用浏览器填好表格后，将表格数据返回 Web 服务器，应把这些信息存入运行 Web 服务器的计算机内的数据库。Web 服务器并不能完成此工作,它调用运行 Web 服务器的计算机上的其它程序完成,这个程序叫 Web 服务器应用程序。又如，建立一个中学校园网，可以建立一个学生情况数据库，可供校长、教导处、老师查询，可能要根据一些条件进行查询，如某班的学生，获得三好生的学生等，浏览器把所查找的条件送给 Web 服务器后，Web 服务器必须调用 Web 服务器应用程序完成此项工作，此程序查到有关数据后，要变成用 HTML 语言的网页，送给 Web 服务器，再送给请求此项内容的客户机。编制服务器端的 Web 服务器应用程序的方法主要有如下几种：CGI，ISAPI，NSAPI，ASP，ASP.NET，JSP，PHP 等。ASP.NET 技术是微软最新提出的编制动态网页技术。现在 Web 数据库比较流行的方法是三层数据库，如下图：



访问数据库的方法

这里的 3 层是指浏览器、Web 服务器和数据库。一些文献中提到 4 层数据库系统是在 Web 服务器和数据库之间增加一个应用服务器层。本章既是讲述设计 Web 服务器应用程序的方法。

9.2.13 修改宿主目录及建立虚拟目录

创建一个网站，必须有如下条件：第一，有固定的 IP 地址。第二，安装并运行 Web 服务器软件，例如 Windows 操作系统下的 IIS，Unix 或 Liunx 下的 Aparc。第三，在宿主目录下有可供游览的网页。Windows2000 的 IIS 服务器的宿主目录为\InetPub\wwwroot。可以修

改宿主目录为其它任意目录，修改 Windows2000 的 IIS 服务器的宿主目录方法如下：

(1) 在 D 盘建文件夹：ASP

(2) 打开控制面板，双击管理工具图标。

(3) 双击 Internet 服务管理器图标。

(4) 右击第一个+号后的计算机名，出现弹出菜单，选择菜单项新建/WEB 站点，按向导步骤选择 D:/asp 为新站点。单击第一个+，打开文件树，右击默认 WEB 站点，将其停止。右击管理 WEB 站点，将其停止。

请读者作如下试验：首先用记事本创建如下网页：

```
<html>
  <body>
   这是我的第一个网页
  </body>
</html>
```

以文件名 Test.htm 存到 d:/asp 文件夹中，查找本机 IP 地址，假如为：202.204.206.98。在另一台计算机中打开浏览器，输入地址：http://202.204.206.98/Test.htm，在浏览器中应能看到文字：这是我的第一个网页。由此可知，必须把创建的网页拷贝到宿主目录下。从本机访问宿主目录下的网页时，可以在浏览器的 URL（地址）处键入：http://localhost/网页以宿主目录为根目录的全路经。其他人访问时，可以在浏览器的 URL（地址）处键入：http://IP 地址或域名/网页以宿主目录为根目录的全路经。

也可以用建虚拟目录的方法，存放网页，具体步骤如下：

(1) 打开控制面板，双击管理工具图标。

(2) 双击 Internet 服务管理器图标。

(3) 单击计算机名前的+号。

(4) 右击默认 Web 站点，出现弹出菜单，选择菜单项新建/虚拟目录，按向导步骤选择 D:/asp 为新虚拟目录。此时，在默认 Web 站点下将会出现设定的虚拟目录，此目录允许其他人访问。

请读者想一想，如把文件 Test.htm 拷贝到新虚拟目录，在另一台计算机中如何访问此网页。

9.2 HTML 标记语言

网页使用 HTML 标记语言写成。HTML 标记是用特殊的 ASCII 字符来定义网页中的格式，字体等等特点。由于各种系统中，都支持 ASCII 字符标准，不同系统中的浏览器都可以解释这些 ASCII 标记，将其所表示的网页在屏幕中显示。这样，不同的系统都可以使用统一 ASCII 标记，访问其它系统中的网页。

如果有 WEB 服务器时，用 IE 浏览器显示网页前，必须把网页拷贝到宿主目录下，例如 Windows2000 的 IIS 服务器为\InetPub\wwwroot。访问此网页时，在浏览器的 URL（地址）处键入此网页的 URL，回车即可。

为了在没有 WEB 服务器时，能用 IE 浏览器显示静态网页，首先将 IE 的默认网页设置为 about:blank。然后运行 IE，在地址栏中输入网页文件的路径，回车即可。修改 IE 的默认网页为 about:blank 及 IE 的默认编辑器为写字板的具体办法是：单击 IE 菜单工具/Internet 选项，打开 Internet 选项对话框。在常规页中的主页选择使用空白页，程序页中，HTML 编辑器选择 Windows Notepad。

9.2.1 HTML 标记

HTML 标记是用特殊的 ASCII 字符来定义网页中的格式、字体、字符对齐方式等特点。其格式为：<标记名称>被控制的文字</标记名称>。其中，<标记名称>为开始标记，</标记名称>为结束标记，一般用来定义被控制的文字的格式或字体等。例如，下列标记使被控制的文字 ASP.NET 中间对齐： ASP.NET

9.2.2 HTML 文件结构

一个网页文件的最基本 HTML 标记结构如下：

```
<html>
  <head>
   <title>
    显示在浏览器标题栏中的文字
   </title>
  </head>
  <body>
   这里是网页的内容
  </body>
</html>
```

<html>表示网页文件的开始，</html>表示网页文件的的结束，网页的所有内容都应在这两个标记之间。<head>…</head>之间可以设定网页的一些信息，<title> </title>之间的文字显示在 IE 浏览器的标题栏中。<body>…</body>之间时网页的主题内容。由此可以看出，在 HTML 语法中，HTML 标记可以嵌套，一个 HTML 标记可以包含另一个 HTML 标记，但标记的嵌套不能是交错的。下边是一个实际例子：

```
<html>
  <head>
    <title>
     这是我的第一个网页
    </title>
  </head>
  <body>
     <center>
这是我的第一个网页。<BR>
其中&ltBR&gt 标记表示换行，
注意，仅键入回车时不能在网页中换行的。<BR>
注意空格的用法，       键入四个空格，网页中只有一个空格。<BR>
增加 4 个空格，     是 4 个空格<BR>
网页中不区分大小写，不同浏览器显示的效果可能不同<BR>
<!--这是注释，浏览时不显示-->
注意，特殊符号的显示：&lt，&gt，&quot，&amp，&copy，&reg
     </center>
  </body>
```

```
</html>
```
网页中可以增加一些标记，例如：
```
<html>
  <head>
    <title>,<base>,<link>,<isindex>,<meta>
  </head>
  <body>
HTML 文件的正文写在这里…
 </body>
</html>
```
这些标记的用法见以下各节。

9.2.3 语言字符集的信息

语言字符集的信息用法见下边的网页。主要是选用网页使用的字符集，gb2312 是中文字符集。
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>meta 的使用</title>
</head>
<body>
meta 的使用。
</body>
</html>
```

9.2.4 背景色彩和文字色彩

设置背景和文字色的语法如下：<body bgcolor=#,text=#,link=#,alink=#,vlink=#>其中，bgcolor,text,link,alink,vlink 的意义见书，#表示颜色，可用 rrggbb 表示，rr,gg,bb 分别是表示红色，绿色，蓝色的 16 进制数，例如，红色为：ff0000。下例设置背景色为红色。
```
<html>
<head>
<title>设置背景色为红色</title>
</head>
<body bgcolor=ff0000>
设置背景色为红色。
</body>
</html>
```

9.2.5 页面空白

可以设置页面的上下左右边距，单位为像素。例如设置左边距为 20 个像素格式如下：
<body leftnargin=20>

9.2.6 显示一幅图

下例在网页中显示一幅图。注意<IMG src="file:///D:/耿肇英/ASP.NET 教案/baobao048.jpg" width="320" height="240">的写法。其中，file://是文件协议，用来选择本地文件。一般网页中用 http://，即超文本协议，此时要求运行 WEB 服务器。

```
<html>
  <head>
   <title>
    增加一幅图形!
   </title>
  </head>
  <body>
    <IMG src="file:///D:/耿肇英/ASP.NET 教案/baobao048.jpg" width="320" height="240">
  </body>
</html>
```

9.2.7 超级链接

浏览网页时，当鼠标变为手形时，单击，可以打开另一个网页，下边的例子在当前窗口打开另一个网页。

```
<html>
<head>
 <title>
  链接的例子
 </title>
</head>
<body>
  这是一个
  <a href="C214.htm">链接的例子</a>
  点一下带下划线的文字!
</body>
</html>
```

9.2.8 超级链接在新窗口打开另一网页

如下例子：

```
<html>
```

```
<head>
 <title>
    开一个新窗口例子
 </title>
</head>
<body>
   <a href="window.htm"target="window_name">
    开一个新窗口！
   </a>
</body>
</html>
```

将光标移到”开一个新窗口”，光标变为手型，单击”开一个新窗口”，将打开另一个网页窗口，即 window.htm 网页。window.htm 网页如下：

```
<html>
<head>
 <title>
    New Window Form HyperLink Dir
 </title>
</head>
<body>
 <H1aligh=center>这是新开的窗口！</H1>
 <center><p>返回原先窗口<a href="C226.htm#window">开新一个(浏览器)窗口</a>"一处
 </p></center>
</body>
</html>
```

9.2.9 标尺线

标尺线标记格式为：<hr align=left color="red" noshade size=10 width=50>。其中，align 为标尺线对齐方式，可以取 left，right，center。Color 为标尺线颜色。Noshade 设定标尺线为没有三维效果的实心黑线。Size 为标尺线的厚度。width 为标尺线的宽度，可以是像素值，也可以相对于窗口的百分比。实际使用见下例：

```
<html>
<head>
 <title>
    标尺线
 </title>
</head>
<body>
   <hr size=10>
   <hr width=#>
   <hr width=50>
   <hr width=50%>
   <hr align=left>
```

```
    <hr align=right>
    <hr width=50% align=left>
    <hr width=50% align=right>
    <hr noshade>
    <hr color="red">
</body>
</html>
```

9.2.10 网页中标题的字体

网页中一共有 6 种标题字体，见下例：

```
<html>
<head>
  <title>
    标题字体
  </title>
</head>
<body>
    <h1>这是 1 级标题！</h1>
    <h2>这是 2 级标题！</h2>
    <h3>这是 3 级标题！</h3>
    <h4>这是 4 级标题！</h4>
    <h5>这是 5 级标题！</h5>
    <h6>这是 6 级标题！</h6>
</body>
</html>
```

其中，标题字体是黑体，每个标题自动插入一个空行。

9.2.11 网页中正文字体

网页中正文字体只有 7 种大小，可以用如下方法设置正文字体：

```
<html>
<head>
  <title>
    正文字体
  </title>
</head>
<body>
    <font size=7>字号为 7，最大字体</font>
    <font size=6>字号为 6</font>
    <font size=5>字号为 5</font>
    <font size=4>字号为 4</font>
    <font size=3>字号为 4</font>
```

```
   <font size=2>字号为 2</font>
   <font size=1>字号为 1，最小字体</font>
</body>
</html>
```
注意字号只能是 1 到 7。

9.2.12　斜体、粗体字符及为字体增加下划线，删除线

可以使字体为斜体、粗体，为字体增加下划线，删除线。见下例：

```
<html>
<head>
</head>
<body>
   <b>标记内的字为黑体</b> <br>
   <i>标记内的字为斜体</i> <br>
   <u>标记内的字有下划线</u> <br>
   <tt>标记内的字等宽，例如 w 和 i 等宽</tt> <br>
   H<sub>2</sub> ,2 为下标<br>
   A<sup>2</sup> ,2 为上标<br>
   <s>标记内的字加删除线</s> <br>
   <strike>标记内的字加删除线</strike> <br>
</body>
</html>
```

9.2.13　字体标记的组合使用

见下例：

```
<html>
<head>
  <title>
   字体标记的组合使用
  </title>
</head>
<body>
   <i><font size=5>
        <b>今天</b>  天气<font size=6>真好！ </font>
   </font></i>
</body>
</html>
```

9.2.14 字体的颜色

见下例：

```
<html>
<head>
</head>
<body>
   <font size=5 color=000000>Black</font>&
   <font size=5 color=red>Red</font>
</body>
</html>
```

9.2.15 客户端字体

每个操作系统都提供若干字体，网页可以使用操作系统提供的字体，下边的网页显示如何使用这些字体。

```
<html>
<head>
</head>
<body>
   <h1>客户端字体事例</h1>
   <font face="Arial">Arial...</font><br>
   <font face="Comic Sans MS">Comic Sans MS...</font><br>
   <font face="Courier">Courier...</font><br>
   <font face="Courier New">Courier New...</font><br>
   <font face="Modern">Modern...</font><br>
   <font face="MS Serif">MS Serif...</font><br>
   <font face="MS-DOS CP 932">MS-DOS CP 932...</font><br>
   <font face="Roman">Roman...</font><br>
   <font face="Script">Script...</font><br>
   <font face="Small Fonts">Small Fonts...</font><br>
   <font face="Symbol">Symbol...</font><br>
   <font face="Times Roman">Times Roman...</font><br>
   <font face="Times New Roman">Times New Roman...</font><br>
   <font face="WingDings">WingDings...</font><br>
</body>
</html>
```

9.2.16 网页中控件的概念

WEB 服务器为了和用户进行交互，必须解决如下问题：首先，用户应能输入一些数据，例如，要查询的条件，用户登录的信息等等。第二，这些数据用什么方法传到 WEB 服务器。

第三，WEB 服务器用那个程序响应用户。为了实现以上功能，必须使用窗体控件，也叫表单控件 form，Visual Studio.net 中叫 WebForm。同时还需要一些其它控件，例如，编辑框控件，列表框控件，下拉列表框控件和按钮等。可以用 HTML 标记语言定义控件。IE 浏览器看到这些标记，就把它显示为相应的控件。控件有许多属性，也可以用 HTML 标记语言表示，每个属性用空格分开，用属性名称=属性值格式定义。

9.2.17 窗体控件和其它控件的使用

窗体控件是其它控件的容器，所有其它控件都要放到窗体控件中。Form 控件的基本语法如下：

```
<form action=URL method="POST">
   <!--在此处增加交互控件，例如编辑框、列表框-->
  <input type="submit" value="提交" name="B1">
  <input type="reset" value="全部重写" name="B2">
</form>
```

其中，<form action=URL method="POST">...</form>定义Form控件，action是WEB服务器用响应用户程序的URL，method="POST"是数据用POST方法传到WEB服务器，也可以是get方法。用户用交互控件输入信息。<input type="submit" value="提交" name="B1">是一个按钮控件，其类型为submit，按钮标题为：提交，代表此控件的name为B1。当用户单击此按钮，form控件将把控件内的所有交互控件中的数据用POST方法，传递给action指定WEB服务器的程序处理。<input type="reset" value="全部重写" name="B2">也是一个按钮，用户单击此按钮，将清空form控件内的所有交互控件。

form控件内可以增加交互控件，交互控件的使用<input>标记，其语法如下：<input align=top|middle|bottom [check] maxlength=? Name=? Size=? Src=? Type=? Value=?>各个属性意义如下：

align为对齐方式，可以取top、middle和bottom。

Type为控件类型，可以是：type="submit"为提交按钮。type="reset"为全部重写按钮。type="text"为编辑框。type="password"为口令编辑框。type="checkbox"为复选框。type="radio"为单选框。type="image"为图形。type="hidden"用户不能看到，可以用来传递网页的一些默认的数据。

[check]只有在type="checkbox"或"radio"时使用，表示缺省被选中。

Maxlength属性在type="text"编辑框时表示最大字符长度。

Name属性代表此控件的名字。

Size属性在type="text"编辑框时表示编辑框最大宽度。

Value 如为按钮，则为标题，为编辑框，则为缺省内容。此值将按 Name/Value 对的形式传递给 WEB 服务器。

9.2.18 例子：文字输入和密码输入

```
<html>
<head>
<title>Form 的例子</title>
</head>
```

```
<body>
<form method="POST" action="--WEBBOT-SELF--">
  您的姓名：<input type="text" name="T1" size="20"><br>
  您的主页的网址：<input type="text" name="T2" size="20"><br>
  密码：<input type="password" name="T3" size="20"><br>
  <input type="submit" value="提交"><input type="reset" value="全部重写"><br>
</form>
</body>
</html>
```

9.2.19 用 FontPage 做网页的例子，使用复选框和单选按钮

用 FontPage 或 Dreamerware 要比使用记事本编辑网页方便的多。下边用 FontPage 做网页的例子。下边是具体步骤：

(1) 运行 FontPage。

(2) 单击菜单文件/新建/网页。出现新建对话框，选择常规页的普通网页，单击确定按钮，创建一个新网页。

(3) 单击菜单插入/表单/表单，增加一个新表单(form)。在表单中已有一个提交按钮和一个全部重写按钮。将光标移到提交按钮前后回车，增加 form 的尺寸。

(4) 右击提交按钮，在弹出菜单中选中菜单项表单域属性，出现按钮属性对话框，可修改名称(Name 属性)属性，值/标签(value 属性)=提交查询内容，按钮类型为提交。按确定按钮。用同样方法修改另一个按钮的标题为重置。

(5) 将光标移到第一行，单击菜单插入/表单/复选框。右击复选框，在弹出菜单中选中菜单项表单域属性，出现复选框属性对话框，修改名称(Name 属性)属性=水果 1。在复选框后键入字符：Banana。用同样方法增加另两个复选框，注意初始状态修改为选中。

(6) 用 FontPage 创建的网页文件如下：

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<meta name="GENERATOR" content="Microsoft FrontPage 4.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>New Page 2</title>
</head>
<body>
<form method="POST" action="--WEBBOT-SELF--">
  <!--webbot bot="SaveResults" U-File="fpweb:///_private/form_results.txt"
  S-Format="TEXT/CSV" S-Label-Fields="TRUE" -->
  <p><input type="checkbox" name="水果1" value="香蕉">Banana</p>
  <p><input type="checkbox" name="水果2" value="苹果" checked>Apple</p>
  <p><input type="checkbox" name="水果3" value="桔子" checked>Orange</p>
```

```
  <p><input type="submit" value="提交查询内容" name="B1"><input type="reset" value="全部重写" name="B2"></p>
</form>
</body>
</html>
```

9.3 ASP.NET 技术基础

设计静态网页有两种方法：一种是使用记事本，用 HTML 语言编写，另一种是使用可视化工具，如 FrontPage，Dreamware 等。显然，使用可视化工具要方便快捷的多。以往设计服务器端动态网页时，例如 ASP，往往只能使用记事本一行一行的写，效率很低。程序员迫切需要一种设计服务器端动态网页的可视化工具，能象使用 C#设计 Window 应用程序一样设计动态网页，使用控件类、属性和事件等面向对象的概念。为了实现这个目的，引入 ASP.NET 服务器端控件概念。静态网页中一般有一个表单(Form)，在表单中可以有多个控件，例如，列表框、编辑框、按钮等等，通过这些控件，完成一定的功能。同样，ASP.NET 服务器端控件首先引入运行在服务端 WebForm 概念，在 WebForm 中可以放入多个服务器端控件，例如，列表框、编辑框、按钮等等，所有这些控件，都是.NET 框架类库中相应类的对象，每个对象都有自己的属性、方法和事件。这些概念和编制 Windows 应用程序相应的概念基本相同。这些 ASP.NET 服务器端控件，也使用 HTML 标记描述，但这些服务器端控件并不传送这些 HTML 标记给浏览器解释，而是由 Web 服务器负责解释，翻译为所有浏览器都能解释的标准 HTML 标记后传送给浏览器解释，这样就极大地简化了服务器端动态网页的设计，也保证了生成的网页的显示效果和浏览器无关。使用 ASP.Net 技术创建的服务器端动态网页的扩展名为.aspx。

本节首先介绍 ASP.NET 服务器端控件基本概念，然后介绍使用记事本编写 ASP.NET 动态网页的方法，最后介绍如何使用 Visual Studio.NET 编写 ASP.NET 动态网页。

9.3.1 HTML 服务器端控件

ASP.Net 中的 HTML 服务器端控件和标准的 HTML 控件有着对应关系，但功能更强大。可以在程序中修改 HTML 服务器端控件的属性，能够在服务器端响应事件，支持数据绑定等。例如增加一个 HTML 服务器端控件编辑框用如下 HTML 语句：

```
<INPUT TYPE="TEXT" ID="Text1" MAXLENGTH=16 RUNAT="SERVER"/>
```

这里和标准的 HTML 控件的区别是增加了属性 RUNAT="SERVER"。属性 ID 是代表这个控件的唯一标志，和 Winndows 应用程序中的控件属性 Name 的意义是一样的。HTML 服务器端控件是为了方便原来学习 HTML 或 ASP 编写 Web 应用程序的程序员而提供的。如果，你以前是 Windows 应用程序员，建议使用 Web 服务器端控件，这些控件不但功能更强大，而使用上更象 Windows 应用程序中的控件，因此学习更方便。因此这里就不介绍 HTML 服务器端控件了。

9.3.2 Web 服务器端控件

在 ASP.NET 系统中，除了常规的 HTML 控件外，还包括 Web 服务器端控件。同 HTML 服务器端控件一样，这些控件可以在程序中修改服务器端控件的属性，能够在服务器端响应事件，支持数据绑定等。例如定义一个 Web 服务器端控件编辑框控件，方法如下：

```
<asp:TextBox id="textBox1" runat="server"/>
```

服务器端控件不但功能更强大，而且和编制 Windows 应用程序中的控件使用方法基本一致，因此学习更方便。本书的所有例子都使用 Web 服务器端控件。

9.3.3 Web Form 的事件处理

象 Windows 应用程序一样，ASP.Net 应用程序也采用事件驱动的概念，用户对浏览器的各种操作都被看作事件，事件产生后，Web 应用程序用事件处理函数响应事件。但 ASP.Net 的事件驱动和 Windows 应用程序的事件驱动有着本质上的区别。Web 应用程序的事件产生后，由于事件处理程序在 Web 服务器端，Web 应用程序把事件通过网络使用 HTTP 协议由浏览器传到 Web 服务器，在 Web 服务器执行事件处理程序，把运行结果转变为标准 HTML 标志的网页，传回浏览器。

在 Web 事件处理机制中，每一次 Web 应用程序响应事件都会使得网页重新生成。事实上，一旦服务器完成某一个网页的处理操作并将它传送至浏览器，则会随即移除该网页的所有信息，也就是说，网页中定义的对象和变量在服务器端已不存在了，网页生命周期结束。当 Web 应用程序再一次响应事件时，服务器又会将上述处理过程重做一次。基于此原因，我们说网页是无状态的——即网页变量与控件的数据值并不会保留在服务器上。

因此，我们增加事件函数时，应考虑网络传播的速度的影响，不能象 Windows 应用程序那样，响应太多的事件。在网页中，每个控件都有属性 AutoPosBack，其值为 true，事件才能自动调用事件处理函数，如果不希望响应该事件，应将该控件的属性 AutoPosBack 设为 false。

9.3.4 记事本编写 ASP.NET 动态网页

ASP.NET 中的服务器端控件也用 HTML 标记，但这些服务器端控件的 HTML 标记并不传送给浏览器解释，而是由 Web 服务器负责解释，翻译为所有浏览器都能解释的标准 HTML 标记后传送给浏览器解释。所有 ASP.NET 服务器端控件都要放到 Web 窗体(WebForm)中，Web 窗体(WebForm)也由 Web 服务器负责解释。下边是一个最简单的使用服务器端控件的例子：

```
<%@ Page language="c#" %>
<html>
  <head>
    <title>这是我的第一个 ASP.NET 网页</title>
  </head>
<script language="c#" runat=server>
    void Page_Load(Object src,EventArgs e)
      {Label1.Text="现在的时间是："+DateTime.Now;}
    void EnterBtn_Click(Object src,EventArgs e)
```

```
    {Label1.Text="现在的时间是："+DateTime.Now;}
</script>
<body>
  <form action="e1.aspx" runat=server>
  <asp:Label id="Label1" font-size="14" font-bold="true" forecolor="red" Text=""runat=server>
  </asp:Label>
  <br>
  <asp:button text="查看时间" Onclick="EnterBtn_Click" runat=server/>
  </form>
</body>
</html>
```

网页文件第一条语句表示网页中使用 C#语言。<html>表示网页文件的开始，</html>表示网页文件的的结束，网页的所有内容都应在这两个标记之间。定义在标记<html>和</html>之间的内容被分为三部分，第一部分：<head>和</head>之间可以设定网页的一些信息，<title>和</title>之间的文字显示在 IE 浏览器的标题栏中。第二部分：<script language="c#" runat=server>和</script>标记之间可以定义方法，变量或对象，language="c#"表示在此标记之间定义的方法使用 C#语言，runat=server 表示在此标记之间定义的方法运行在 Web 服务器端，这里定义了两个方法，方法 Page_Load()是 Web 服务器装载本网页时调用的方法，可以在此方法中做一些初始化工作，方法 EnterBtn_Click()是"查看时间"按钮的事件函数。第三部分：<body>和</body>之间是网页在浏览器中显示的内容。<form action="e1.aspx" runat=server>和</form>标记定义 Web 窗体(WebForm)，注意 runat=server 表示 Web 窗体由 Web 服务器解释。在 Web 窗体中增加了两个控件对象，第一个是 Label 控件，asp:Label 表示本控件是 Label 控件，id 相当 Windows 应用程序中控件的 Name 属性，用来区分不同对象，runat=server 表示次控件由 Web 服务器解释，其余是设定属性值，注意不同属性用空格分隔。第二个控件是按钮，请注意定义单击事件函数的方法。

将其以文件名 e1.aspx 存入 d:/asp 文件夹，如果 d:/asp 文件夹已被设定为 Web 站点，可以在 IE 的地址栏输入：http://localhost/c411.aspx 后，看到这个网页。在浏览器端看不到这些代码，用 IE 菜单查看/源代码，可以看到用超文本编制的网页。

9.3.5 用 Visual Studio.NET 实现 ASP.NET 动态网页

用 Visual Studio.NET 实现上节的例子。具体步骤如下：

⑴ 运 Visaul C#后，则进入开始界面，选择新建项目。打开新建项目对话框，在项目类型中选择 Visual C#项目，在模板中选择[ASP.NET Web 应用程序]，指定项目项目放置的位置为 http://localhost/e1，这里 http://localhost 代表当前激活的宿主目录，即将本应用的所有文件存入宿主目录下的文件夹 e1 中，点击“确定”，生成一个空白窗体(WebForm)。用户可在窗体中放入控件。Visual Studio.NET 为我们建立了一个应用项目。

⑵ 向项目中添加控件需要使用［Toolbox］窗口，若看不到，可以用菜单视图/工具箱打开这个窗口。

⑶ 先选中［Toolbox］窗口中［Web 窗体］类型下的［Label］条目，然后在设计的窗体中按下鼠标左键并拖动鼠标，画出一个 Label 控件。该控件用来显示一行文本。

⑷ 使用［Properties］窗口修改 Label 控件的文本内容和文本字体属性。在右下侧属性窗口中找到［text］属性，把它的值由“Label1”改为”现在的时间是：”；接着在属性窗口中找到［Font］属性，选中 Font 所在的单元格，单击 Font 属性左边的“+”号，在出现的子属性中

编辑，可以修改 Label 控件中文本的字体和字号等属性。编辑完成后，单击变成“-”号的方框隐藏 Font 的子属性；修改 Label 控件的 ForeColor 属性，可以修改 Label 中文本的颜色。

(5)从［Toolbox］窗口中选中一个 Button 控件到窗体，在［Properties］窗口中将按钮的［Text］属性分别改为”查看时间”。

(6)为单击查看时间按钮事件(Click)函数增加语句(双击 Click 事件)：

```
private void Button1_Click(object sender, System.EventArgs e)
{
    Label1.Text="现在的时间是："+DateTime.Now;
}
```

(7)为 Page_Load 事件函数增加语句：

```
private void Page_Load(object sender, System.EventArgs e)
{
    Label1.Text="现在的时间是："+DateTime.Now;
}
```

(8) 单击工具栏中蓝色箭头按钮，运行，看一下效果。也可用浏览器看一下，地址为：http://localhost/e1/WebForm1.aspx。请仔细观察，每一步骤 Visual Studio.NET 都为我们增加了什么语句。

9.3.6 Code Behind 技术

Code Behind 技术把界面设计代码和程序设计代码以不同的文件分开，对于代码的重复使用，程序的调试和维护都是十分方便的。特别是在团队开发中，可以使不同人员编辑不同文件，极大地提高了效率。Visual Studio.NET 使用了 Code Behind 技术，当我们使用 Visual Studio.Net 创建了一个 Web 应用程序，将自动创建了两个文件，其中 ASP.NET Web 网页文件 WebForm1.aspx 如下：

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false" Inherits="e2.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
    <head>
      <title> WebForm1</title>
      <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
      <meta name="CODE_LANGUAGE" Content="C#">
      <meta name="vs_defaultClientScript" content="JavaScript">
      <meta name="vs_targetSchema" content="http://schemas.microsoft.com/intellisense/ie5">
    </head>
<body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">
    </form>
</body>
</html>
```

语句<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false" Inherits="e2.WebForm1" %>中的 Codebehind="WebForm1.aspx.cs"表示网页的所有代码在文件 WebForm1.aspx.cs 中，使设计网页的外观和设计代码分离，可同时进行设计，互不影响，

也是网页和代码的逻辑关系清楚，增加了易读性。代码文件 WebForm1.aspx.cs 如下：

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
namespace e2
{
    /// <summary>
    /// WebForm1 的摘要说明。
    /// </summary>
    public class WebForm1 : System.Web.UI.Page
    {
        private void Page_Load(object sender, System.EventArgs e)
        {
            // 在此处放置用户代码以初始化页面
        }
        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: 该调用是 ASP.NET Web 窗体设计器所必需的。
            //
            InitializeComponent();
            base.OnInit(e);
        }
        /// <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.Load += new System.EventHandler(this.Page_Load);
        }
        #endregion
    }
}
```

9.3.7 ASP.NET 和 HTML 兼容

任何一个静态网页只要把其扩展名修改为 aspx，在 ASP.NET 下仍可运行，运行效果和以前相同。见下例，它是一个普通的静态网页。

```
<html>
<head>
<title>学生选课系统</title>
</head>
<body>
 <center>
 <h1>学生选课系统</h1>
 <form method="POST" action="c411.htm">
    <p>姓名：<input type="text" name="T1" size="20"></p>
    <p>学号：<input type="password" name="T2" size="20"></p>
    <p>课程：<select size="1" name="D1">
              <option selected>微积分</option>
              <option>代数与几何</option>
              <option>模拟电路</option>
              <option>机械原理</option>
         </select>
    <input type="submit" value="提交" name="B1"></p>
</form>
</center>
</body>
</html>
```

将其以文件名 c411.aspx 存入 d:/asp 文件夹，如果 d:/asp 文件夹已被设定为 Web 站点，可以在 IE 的地址栏输入：http://localhost/c411.aspx 后，看到这个网页。

ASP.NET 的设计目标之一就是尽可能地保持和现有 ASP 页面的语法及运行库的兼容。希望将现有 ASP 页面文件的扩展名改为.aspx，这些页面仍可以在 ASP.NET 中运行。在大多数情况下该目标已经实现了，但一般要对某些基本代码作出修改，因为 ASP.NET 已不再支持 VBScript 了，而且 VB 语言本身也发生了变化。

9.3.8 网页中使用 C#语句

在网页中，可以插入一些 C#语句，具体用法见下例：

```
<% @ Page Language="C#"%>
<html>
  <body>
    <center>
      <% for (int i=0;i<8;i++){ %>
        <font size="<%=i%>">这是我的第一个 ASP.NET 网页</font><br>
        <%}%>
    </center>
```

</body>
</html>

在浏览器端看不到这些代码，用 IE 菜单查看/源代码，可以看到用超文本编制的网页。这样使用 C#语句，不是一个好的习惯，不建议使用。

第十章 Web 服务器端控件

本章介绍常用的 Web 服务器端控件的属性、事件和方法，以及用 Web 服务器端控件编制服务器端动态网页的方法。

10.1 常用的 ASP.NET 服务器端控件

10.1.1 Label 控件

Label 控件用如下方法定义：

<asp:Label id="label1" font-size="14" font-bold="true" forecolor="red" Text="标签控件" runat=server></asp:Label>或者

<asp:Label id="Label1" font-size="14" font-bold="true" forecolor="red" runat=server>
标签控件</asp:Label>

下边介绍其常用的属性：

- 属性 Text：显示的文本
- 属性 ForeColor 文本的颜色，颜色可以取：红色=System.Drawing.Color.Red。黑色=System.Drawing.Color.Black 等等。
- 字体的属性：黑体为 Font.Bold=true，斜体为 Font.Italic=true 等等。
- 属性 BackColor：背景色
- 属性 id：相当 Windows 应用程序中控件的 Name 属性，用来区分不同对象。
- 属性 sp:Label：表示本控件是 Label 控件。
- 属性 runat=server 表示次控件运行在服务器段，由 Web 服务器解释。

10.1.2 TextBox 控件

Label 控件用如下方法定义：

<asp:TextBox id="textBox1" runat=server></asp:TextBox>

常用的属性如下：

- 属性：Text 显示的文本
- 属性：TextMode=SingleLine 为编辑框，TextMode=MultiLine 为多行编辑框，可以有滚动条。TextMode=PassWord 为口令编辑框。
- 属性：MaxLength 编辑框和口令编辑框时，允许输入的最多字符数。
- 属性：Rows 多行编辑框时表示行数
- 事件 TextChanged：控件中文本发生变化。

10.1.3 Button、LinkButton 和 ImageButton 控件

Button 控件已介绍过了，Text 为按钮的标题，单击事件为：Click。
LinkButton 控件：为超级链接形式的按钮，Text 为按钮超级链接形式的标题，单击事件为 Click。使用方法同 Button 控件，可为其增加单击事件 Click 的事件函数。
ImageButton 控件：有些按钮需要在按钮上增加图案，例如工具条中的按钮，可以使用 ImageButton 控件。属性 ImageUrl 为图案的路径，一般最好和网页文件放在同一个目录下，此时，控件定义如下：
<asp:ImageButton id="ImageButton1" runat="server" ImageUrl="t1.bmp"></asp:ImageButton
使用方法同 Button 控件，可为其增加单击事件 Click 的事件函数。

10.1.4 CheckBox 和 CheckBoxList 控件

CheckBoxList 控件可以创建一组若干 CheckBox 按钮，这些按钮有相同的性质。这些 CheckBox 按钮可以多选，不选或都选，可用来表示一些可共存的特性，例如一个人的爱好。下面例子在窗口中加一 Label 控件用来显示某人的爱好，增加两个 CheckBox 按钮，一个代表是否爱好音乐，一个代表是否爱好文学，每当用鼠标单击 CheckBox 按钮选择或不选择爱好，Label 控件显示实际的爱好。具体步骤如下：
(1)创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。
(2)放工具箱的 Label 控件到窗体,其属性[Text]=“你的爱好是：”。
(3)放工具箱的 CheckBoxList 控件到窗体。
(4)单击属性 Items 后的按钮，出现集合编辑器对话框。单击添加按钮，增加一个 CheckBox 按钮，修改其 Text 属性为”音乐”，用同样方法增加另一个 CheckBox 按钮，修改其 Text 属性为”文学”。注意爱好是可以多选的，因此必须用 CheckBox 控件。
(5)设定属性 AutoPosBack=true。
(6)为 CheckBox1 按钮事件(SelectedIndexChanged)函数增加语句如下：

```
private void CheckBoxList1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    String s="你的爱好是：";
    for(int i=0;i<2;i++)
        {
            if(CheckBoxList1.Items[i].Selected)
                s=s+CheckBoxList1.Items[i].Text;
        }
    Label1.Text=s;
}
```

(7)编译，运行，选中音乐显示：你的爱好是：音乐，再选中文学显示：你的爱好是：音乐文学，...。

10.1.5 RadioButton 和 RadioButtonList 控件

有一类特性是互斥的，例如性别男女，选择这类特性可用 RadioButtonList 控件，该控

件的最大特点是它有多个按钮，但只能选其中的一个按钮，下面是一个例子，两个单选按钮分别为男和女，用 Label 控件显示选择的的结果。具体步骤如下：

(1) 创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。

(2) 放工具箱的 Label 控件到窗体,其属性[Text]=“男”。

(3) 放工具箱的 RadioButtonList 控件到窗体。

(4) 单击属性 Items 后的按钮,出现集合编辑器对话框。单击添加按钮,增加一个 RadioButton 按钮，修改其 Text 属性为”男”， 修改其 Selected 属性为 true。用同样方法增加另一个 RadioButton 按钮，修改其 Text 属性为”女”，修改其 Selected 属性为法 false。

(5) 设定属性 AutoPosBack=true。

(6) 为 SelectedIndexChanged 事件增加事件函数如下：

```
private void RadioButtonList1_SelectedIndexChanged(object sender, System.EventArgs e)
        {

            if(RadioButtonList1.SelectedIndex==0)
               Label1.Text="男";
            else
                Label1.Text="女";
        }
```

(7) 编译，运行，单击 RadioButtonList 中的两个 RadioButton 按钮，显示所作的选择。请想一想和 CheckBox 按钮的区别。

10.1.6 Image 控件

Image 控件用来显示图像，其属性 AlternateText 为字符串类型，如果图形不被正确显示，则显示此字符串。属性 ImageAlign 为图形对齐方式。ImageUrl 为图形的 URL 地址。下例，增加 3 个单选按钮，根据单选按钮那个被选中，显示不同的图像。具体步骤如下：

(1) 创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。

(2) 放工具箱的 RadioButtonList 控件到窗体。

(3) 单击属性 Items 后的按钮,出现集合编辑器对话框。单击添加按钮,增加一个 RadioButton 按钮，修改 Text 属性为”图 1”，修改 Value 属性为”p1.jpg”，修改其 Selected 属性为 true。用同样方法增加另一个 RadioButton 按钮，修改其 Text 属性为”图 2”，修改 Value 属性为”p2.jpg”，修改其 Selected 属性为法 false。

(4) 设定属性 AutoPosBack=true。

(5) 为 RadioButtonList 控件的 SelectedIndexChanged 事件增加事件函数如下：

```
private void RadioButtonList1_SelectedIndexChanged(object sender, System.EventArgs e)
        {
            Image1.ImageUrl=RadioButtonList1.SelectedItem.Value;
        }
```

(6) 为 Page_Load 事件函数增加语句：

```
private void Page_Load(object sender, System.EventArgs e)
        {
            // 在此处放置用户代码以初始化页面
            Image1.ImageUrl=RadioButtonList1.Items[0].Value;
        }
```

(7) 编译，链接，运行，单击 RadioButtonList 中的两个 RadioButton 按钮，显示不同的图像。

10.1.7 HyperLink 控件

HyperLink 控件是超级链接控件，用来从一个网页定向到另一个网页。属性 Text 为设置超级链接的文字。也可以使用图形完成超级链接，ImageUrl 为图形的 URL。属性 NavigateUrl 是定向到另一个网页的 URL。属性 Target=_blank，打开一个新窗口，否则在原窗口打开。实现超级链接的例子的具体步骤如下：

(1) 创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。
(2) 放工具箱的 HyperLink 控件到窗体。
(3) 修改属性 Text=" 超级链接"。
(4) 单击属性 NavigateUrl 后的按钮，出现选择 URL 对话框，URL 类型选择为与根相关的，URL 编辑框添入/bookExample/c412.aspx。
(5) 编译，链接，运行，单击 HyperLink 控件，在原窗口代开另一个网页。如属性 Target=_blank，打开一个新窗口显示新网页。

10.1.8 Table、TableCell 和 TableRow 控件

这是一个表格控件，创建表格的具体步骤如下：

(1) 创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。
(2) 放工具箱的 Table 控件到窗体。
(3) 单击属性 Row 后的按钮，出现选择 TableRow 集合编辑器对话框，单击添加按钮，增加两行。
(4) 选择第 0 个 TableRow，单击属性 Cell 后的按钮，出现选择 TableCell 集合编辑器对话框，单击添加按钮，增加三列。修改每列的属性 Text，分别为：课程总论、刚体静力学、弹性静力学。
(5) 选择第 1 个 TableRow，单击属性 Cell 后的按钮，出现选择 TableCell 集合编辑器对话框，单击添加按钮，增加三列。修改每列的属性 Text，分别为：雅舍、孩子、音乐。
(6) 可以看到两行三列的表，运行看一下。

10.1.9 DrowDownList 控件

这是一个下拉列表控件，创建下拉列表的具体步骤如下：

(7) 创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。
(8) 放工具箱的 DrowDownList 控件到窗体。
(9) 单击属性 Items 后的按钮，出现选择 ListItem 集合编辑器对话框，单击添加按钮，增加三项。修改每项的属性 Text，分别为：课程总论、刚体静力学、弹性静力学。
(10)放工具箱的 Label 控件到窗体，id=Label1。
(11)放工具箱的 Button 控件到窗体，为单击确定按钮事件(Click)函数增加语句(双击 Click 事件)：

```
private void Button1_Click(object sender, System.EventArgs e)
        {
```

```
        Label1.Text=DropDownList1.SelectedItem.Text;
    }
```

(12)编译，链接，运行，从下拉列表选择刚体静力学，单击 Button 控件，Label1 标签控件显示刚体静力学。

10.2 ASP.Net 控件数据绑定

所谓数据绑定技术就是把数据集的某个或者某些数据绑定到控件的某些属性上面的一种技术。说的具体些，就是把数据集中某个或者某些数据绑定到 Text 控件、ListBox 控件、ComBox 等控件上的能够显示数据的属性上面。当对控件完成数据绑定后，其显示的内容将随着数据集的变化而变化。

10.2.1 数据绑定基础

ASP.NET 引入了新的数据绑定语法。这种非常灵活的语法允许开发人员不仅可以绑定到数据源，而且可以绑定到简单属性、集合、表达式甚至是从方法调用返回的结果。

DataBind 是页和所有服务器控件的方法。当需要更新被绑定的数据时，必须调用此方法。当在父控件上调用 DataBind 时，该控件的所有子控件也同时调用自己的 DataBind 方法。例如，当调用 DataList1.DataBind()后，将对 DataList 模板中的每一控件调用 DataBind 方法。在调用页的 DataBind 方法，既 Page.DataBind()，会导致调用页上的所有控件的 DataBind 方法，更新页上所有绑定数据。通常从 Page_Load 事件调用 DataBind，如下例所示。

```
protected void Page_Load(Object Src, EventArgs E) {
    DataBind();}
```

下面的示例说明如何将一个服务器控件的属性绑定到另一个服务器控件的属性。

```
<html>
<head>
    <script language="C#" runat="server">
        void SubmitBtn_Click(Object sender, EventArgs e) {
          Page.DataBind();//更新页内的所有被绑定的数据
        }
    </script>
</head>
<body>
    <h3><font face="宋体">一个服务器控件的属性绑定到另一个服务器控件的属性
</font></h3>
    <form runat=server>
        <asp:DropDownList id="StateList" runat="server">
          <asp:ListItem>苹果</asp:ListItem>
          <asp:ListItem>香蕉</asp:ListItem>
          <asp:ListItem>桔子</asp:ListItem>
        </asp:DropDownList>
        <asp:button Text="提交" OnClick="SubmitBtn_Click" runat=server/>
```

```
        <p>
        选定的水果：<asp:label text='<%# StateList.SelectedItem.Text %>'
runat=server/>
    </form>
</body>
</html>
```

网页中语句 text='<%# StateList.SelectedItem.Text %>'是将 Label 控件的 Text 属性绑定到 DropDownList 控件的属性 StateList.SelectedItem.Text。符号%#表示数据绑定。函数 SubmitBtn_Click 中的语句 Page.DataBind()更新页内的所有被绑定的数据。如果使用 Visual Studio.Net 实现，具体步骤如下：

(6)创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。

(7)放工具箱的 DrowDownList 控件到窗体。单击属性 Items 后的按钮，出现选择 ListItem 集合编辑器对话框，单击添加按钮，增加三项。修改每项的属性 Text，分别为：课程总论、刚体静力学、弹性静力学。

(8) 放工具箱的 Button 控件到窗体，为单击确定按钮事件(Click)函数增加语句(双击 Click 事件)：

```
private void Button1_Click(object sender, System.EventArgs e)
        {
        Page.DataBind();
        }
```

(9)放 Label 控件到窗体,id 为 Label1。单击属性 DataBinding 后标题为…的按钮，打开 Label1 数据绑定对话框，选择自定义绑定表达式(c)，在其下编辑框中输入：DropDownList1.SelectedItem.Text。单击确定按钮。

(10)　运行。

10.2.2　基于变量的数据绑定

ASP.NET 数据绑定语法支持绑定到公共变量、页的属性和页上其他控件的属性。下面的示例说明如何绑定到公共变量和页上的简单属性。注意这些值在 DataBind()调用前初始化。

```
<html>
<head>
    <script language="C#" runat="server">
        void Page_Load(Object sender, EventArgs e) {
            Page.DataBind();
        }
        string custID{//属性
            get {
                return "ALFKI";
            }
        }
        int orderCount{
            get {
                return 11;
            }
```

```
        }
    </script>
</head>
<body>
    <h3><font face="宋体">到页属性的数据绑定</font></h3>
    <form runat=server>
        客户：<b><%# custID %></b><br>
        未结的订单：<b><%# orderCount %></b>
    </form>
</body>
</html>
```

用 Visual Studio.Net 实现的方法见上例及书。

10.2.3 基于集合的绑定

像 DataGrid、ListBox、DrowDownList 和 HTMLSelect 这样的列表服务器控件的列表都可以绑定到数据源。例如绑定到公共语言运行库的集合类型，如 ArrayList、DataView、Hashtable 和 DataReader 等。下面的示例说明如何将 DrowDownList 的列表绑定到 ArrayList。

```
<html>
<head>
    <script language="C#" runat="server">
        void Page_Load(Object Sender, EventArgs E) {
            if (!Page.IsPostBack) {
               ArrayList values = new ArrayList();
               values.Add ("IN");
               values.Add ("KS");
               values.Add ("MD");
               values.Add ("MI");
               values.Add ("OR");
               values.Add ("TN");
               DropDown1.DataSource =values;
               DropDown1.DataBind();
            }
        }
        void SubmitBtn_Click(Object sender, EventArgs e) {
           Label1.Text = "您选择了：" + DropDown1.SelectedItem.Text;
        }
    </script>
</head>
<body>
    <h3><font face="宋体">数据绑定 DropDownList</font></h3>
    <form runat=server>
      <asp:DropDownList id="DropDown1" runat="server" />
```

```
      <asp:button Text="提交" OnClick="SubmitBtn_Click" runat=server/>
      <p>
      <asp:Label id=Label1 font-name="宋体" font-size="10.5pt" runat="server" />
    </form>
</body>
</html>
```

下面的示例说明如何把数据表绑定到 DataGrid。注意使用 DataView 类要引用命名空间 System.Data，即语句<%@ Import namespace="System.Data" %>。具体例子如下：

```
<%@ Import namespace="System.Data" %>
<html>
<head>
<script language="C#" runat="server">
  void Page_Load(Object sender, EventArgs e ) {
   if (!Page.IsPostBack) {
     DataTable dt = new DataTable();//建立一个数据表
     DataRow dr;//建立一个数据表的记录变量
     dt.Columns.Add(new DataColumn("整数值", typeof(Int32)));//增加字段
     dt.Columns.Add(new DataColumn("字符串值", typeof(string)));//包括字段名
     dt.Columns.Add(new DataColumn("日期时间值", typeof(DateTime)));//及字段类型
     dt.Columns.Add(new DataColumn("布尔值", typeof(bool)));
     for (int i = 1; i <= 9; i++) {//增加 10 个记录
        dr = dt.NewRow();//建立 1 个记录对象
        dr[0] = i;//为记录的每个字段赋值
        dr[1] = "项 " + i.ToString();
        dr[2] = DateTime.Now;
        dr[3] = (i % 2 != 0) ? true : false;
        dt.Rows.Add(dr);//把此记录加到数据表中
        }
       dataGrid1.DataSource = new DataView(dt);//为 dataGrid 指定数据源
       dataGrid1.DataBind();//数据更新
       }
    }
  </script>
</head>
<body>
    <h3><font face="宋体">到 DataView 的数据绑定</font></h3>
    <form runat=server>
        <asp:DataGrid id="dataGrid1" runat="server"
          BorderColor="black"
          BorderWidth="1"
          GridLines="Both"
          CellPadding="3"
          CellSpacing="0"
          HeaderStyle-BackColor="#aaaadd"
```

```
        />
    </form>
</body>
</html>
```

下面的示例说明如何绑定到 Hashtable。

```
<html>
<head>
    <script language="C#" runat="server">
        void Page_Load(Object sender, EventArgs e) {
            if (!Page.IsPostBack) {
                Hashtable h = new Hashtable();//注意哈希表的使用
                h.Add ("键 1", "值 1");//哈希表的每一个元素是一对键和值
                h.Add ("键 2", "值 2");
                h.Add ("键 3", "值 3");
                MyDataList.DataSource=h;//为列表框指定数据源
                MyDataList.DataBind();//数据更新
            }
        }
    </script>
</head>
<body>
    <h3><font face="宋体">到哈希表的数据绑定</font></h3>
    <form runat=server>
        <asp:DataList id="MyDataList" runat="server"
          BorderColor="black"
          BorderWidth="1"
          GridLines="Both"
          CellPadding="4"
          CellSpacing="0"
          >
            <ItemTemplate>
                <%# ((DictionaryEntry)Container.DataItem).Key %> :
                <%# ((DictionaryEntry)Container.DataItem).Value %>
            </ItemTemplate>
        </asp:DataList>
    </form>
</body>
</html>
```

标记<ItemTemplate>指定显示格式</ItemTemplate>是按指定显示格式重复显示数据源中的所有数据。本例中应显示 3 组数据，指定显示格式为：键 1：值。

下面介绍如何将 ListBox、DrowDownList 和 HTMLSelect 这样的列表服务器控件的列表绑定到数据表的某一字段上。

```
<%@ Import namespace="System.Data" %>
<html>
```

```
<head>
<script language="C#" runat="server">
  void Page_Load(Object sender, EventArgs e ) {
   if (!Page.IsPostBack) {
     DataTable dt = new DataTable();//建立一个数据表
     DataRow dr;//建立一个数据表的记录变量
     dt.Columns.Add(new DataColumn("Itemid", typeof(Int32)));//增加字段
     dt.Columns.Add(new DataColumn("ItemName", typeof(string)));//包括字段名
     dt.Columns.Add(new DataColumn("ItemDate", typeof(DateTime)));//及字段类型
     dt.Columns.Add(new DataColumn("ItemBool", typeof(bool)));
     for (int i = 1; i <= 9; i++) {//增加 10 个记录
        dr = dt.NewRow();//建立 1 个记录对象
        dr[0] = i;//为记录的每个字段赋值
        dr[1] = "项 " + i.ToString();
        dr[2] = DateTime.Now;
        dr[3] = (i % 2 != 0) ? true : false;
        dt.Rows.Add(dr);//把此记录加到数据表中
        }
       StateList.DataSource = new DataView(dt);//为 DropDownList 指定数据源
       StateList.DataTextField="ItemName";
       StateList.DataValueField="Itemid";
       StateList.DataBind();//数据更新
       }
    }
   void SubmitBtn_Click(Object sender, EventArgs e) {
    Label1.Text="StateList 的 Text="+StateList.SelectedItem.Text+":StateList 的
Value="+StateList.SelectedItem.Value;
   }

  </script>
</head>
<body>
    <h3><font face="宋体">到 StateList 的数据绑定</font></h3>
    <form runat=server>
        <asp:DropDownList id="StateList" runat="server"/>
         <asp:button Text="提交" OnClick="SubmitBtn_Click" runat=server/>
         <br>
        <asp:Label id="Label1" runat="server"/>
    </form>
</body>
</html>
```

10.2.4 基于表达式绑定

通常需要在绑定到页或控件之前操作数据。下面的示例说明如何绑定到表达式和方法的返回值。

```
<html>
<head>
    <script language="C#" runat="server">
        void Page_Load(Object Src, EventArgs E) {
            if (!Page.IsPostBack) {
               ArrayList values = new ArrayList();
               values.Add (0);
               values.Add (1);
               values.Add (2);
               values.Add (3);
               values.Add (4);
               values.Add (5);
               values.Add (6);
               DataList1.DataSource = values;
               DataList1.DataBind();
            }
        }
        String EvenOrOdd(int number) {
            if ((number % 2) == 0)
              return "偶数";
            else
              return "奇数";
        }
    </script>
</head>
<body>
  <h3><font face="宋体">到方法和表达式的数据绑定</font></h3>
  <form runat=server>
    <asp:DataList id="DataList1" runat="server"
      BorderColor="black"
      BorderWidth="1"
      GridLines="Both"
      CellPadding="3"
      CellSpacing="0"
      >
      <ItemTemplate>
       数字值：<%# Container.DataItem %><%--Container 表示数据源--%>
       偶/奇：<%# EvenOrOdd((int) Container.DataItem) %><%--绑定到函数返回值--%>
      </ItemTemplate>
```

```
    </asp:datalist>
  </form>
</body>
</html>
```

10.2.5 基于 DataBinder.Eval 方法的数据绑定

为将绑定的数据按指定数据类型转化为字符串，可以使用 String.Format 方法。请看下面的示例,该例要将数据表中字段名为"IntegerValue"的数据转换为货币的数据类型的字符串输出。

```
<%# String.Format("{0:c}",((DataRowView)Container.DataItem)["IntegerValue"])%>
```

该语法可能比较复杂，难以记忆。ASP.NET 提供了一种静态方法 DataBinder.Eval，可以将绑定的数据按指定数据类型转化为字符串。该方法使用很方便，因为它消除了开发人员为强迫将数值转换为所需的数据类型而必须做的许多显式转换。这在数据绑定模板列表内的控件时尤其有用，因为通常数据字段的类型都必须转换。为将整数显示为货币字符串，使用 #DataBinder.Eval 格式如下：

```
<%#DataBinder.Eval(Container.DataItem, "IntegerValue", "{0:c}") %>
```

DataBinder.Eval 是一个具有三个参数的方法，第一个参数是数据源的当前记录，在象 DataList、DataGrid 或 Repeater 这样的模板列表中，该参数始终是 Container.DataItem，第二个参数是数据表字段名,表示要将此字段的数据转换为第三个参数指定的数据类型的字符串，第三个参数为格式字符串，{0:c}表示货币类型。格式字符串参数是可选的。如果省略它，则 DataBinder.Eval 将此字段的数据转换为字段本身的数据类型的字符串，如下例所示，输出为字符串"true"或"false"。

```
<%# (bool)DataBinder.Eval(Container.DataItem, "BoolValue") %>
```

具体的实例如下：

```
<%@ Import namespace="System.Data" %>
<html>
<head>
    <script language="C#" runat="server">
        void Page_Load(Object sender, EventArgs e) {
            if (!Page.IsPostBack) {
                DataTable dt = new DataTable();
                DataRow dr;
                dt.Columns.Add(new DataColumn("IntegerValue", typeof(Int32)));
                dt.Columns.Add(new DataColumn("StringValue", typeof(string)));
                dt.Columns.Add(new DataColumn("DateTimeValue",
typeof(DateTime)));
                dt.Columns.Add(new DataColumn("BoolValue", typeof(bool)));
                for (int i = 0; i < 9; i++) {
                    dr = dt.NewRow();
                    dr[0] = i;
                    dr[1] = "项 " + i.ToString();
                    dr[2] = DateTime.Now;
                    dr[3] = (i % 2 != 0) ? true : false;
```

```
                dt.Rows.Add(dr);
            }
            dataList1.DataSource = new DataView(dt);
            dataList1.DataBind();
        }
    }
</script>
</head>
<body>
    <h3><font face="宋体">使用 DataBinder.Eval 进行数据绑定</font></h3>
    <form runat=server>
        <asp:DataList id="dataList1" runat="server"
          RepeatColumns="3"
          Width="80%"
          BorderColor="black"
          BorderWidth="1"
          GridLines="Both"
          CellPadding="4"
          CellSpacing="0"
          >
            <ItemTemplate>
                订购日期:<%# DataBinder.Eval(Container.DataItem, "DateTimeValue",
"{0:d}") %>

                <p>
                数量：<%# DataBinder.Eval(Container.DataItem, "IntegerValue",
"{0:N2}") %>
                <p>
                项：<%# DataBinder.Eval(Container.DataItem, "StringValue") %>
                订购日期： <asp:CheckBox id=chk1 Checked='<%#
(bool)DataBinder.Eval(Container.DataItem, "BoolValue") %>' runat=server/>
                <p>
            </ItemTemplate>
        </asp:Datalist>
    </form>
</body>
</html>
```

控件 DataList 中的 ItemTemplate 是模板控件，其功能是将控件 DataList 的数据源中的所有数据，按 ItemTemplate 模板控件所指定的格式显示。

10.2.6 列表绑定控件

列表绑定控件通用属性

1. **DataSource** 属性

DataList、DataGraid 和 Repeater 都提供了 DataSource 属性。使用 DataSource 属性指定要绑定到上述三种数据列表控件的数据源,用数据源中的数据填充上述三种数据列表控件。数据源必须是实现 System.Collections.ICollection 接口的具有相同对象的集合，例如 System.Data.DataView(见 6.3 节最后例子)、System.Collections.ArrayList(见 6.3 节例 1)和 System.Collections.Hashtable(见 6.3 节哈希表例子)。

2. **Items** 集合

Items 是一个集合属性，包含一些具有相同特征的若干对象的集合。DataList、DataGraid 和 Repeater 控件中包含多个 Items 集合属性，使用 **Items** 集合用来以编程的方式控制 DataList、DataGraid 和 Repeater 控件中的各项。例如：

a) AlternatingItem：DataGraid 中所有奇数编号行的项的集合
b) SelectedItem：当前选中的所有项的集合。
c) EditItem：正在编辑的行中所有项的集合。例子见 7.2.4 更新数据的语句
MyDataGrid.EditItemIndex = (int)E.Item.ItemIndex;。
以上各项都和数据源有关，以下各项和数据源无关。
d) Header:所有列表表头项的集合
e) Footer: 所有列表表尾项的集合
f) Separtor:DataGraid 和 Repeater 控件中分隔符线的集合
g) Page:DataGraid 在分页显示中，每页数据项的集合。

3.数据绑定和 Items 集合的创建

当为 DataList、DataGraid 和 Repeater 等列表控件的属性 DataSource 指定数据源，并执行数据绑定函数 DataBind 方法后，列表控件将创建 Items 集合，并从数据源取得显示所需的数据，可以通过 Items 属性来获得列表控件中各项的内容。注意，只有绑定到数据源的项才包含在 **Items** 集合中。页眉、页脚和分隔符不包含在该集合中。下面的示例展示如何使用 **Items** 集合来显示 DataList 控件中的项。

```
<%@ Import Namespace="System.Data" %>
<html>
  <script language = "C#" runat="server">
     ICollection CreateDataSource()
     {
        DataTable dt = new DataTable();
        DataRow dr;
        dt.Columns.Add(new DataColumn("StringValue", typeof(string)));
        for (int i = 0; i < 10; i++)
        {
           dr = dt.NewRow();
           dr[0] = "Item " + i.ToString();
           dt.Rows.Add(dr);
        }
        DataView dv = new DataView(dt);
        return dv;
     }
     void Page_Load(Object sender, EventArgs e)
     {
        if (!IsPostBack)
```

```
        {
            DataList1.DataSource = CreateDataSource();//指定数据源
            DataList1.DataBind();//数据绑定
        }
    }
    void Button_Click(Object sender, EventArgs e)
    {
        if (DataList1.Items.Count>0)//Items 项数>0
        {
            Label1.Text = "The Items collection contains: <br>";

            foreach(DataListItem item in DataList1.Items)
            {

                Label1.Text += ((DataBoundLiteralControl)item.Controls[0]).Text +
                               "<br>";
            }
        }
    }

</script>

<body>

<form runat=server>

    <h3>DataList Items Example</h3>

    <asp:DataList id="DataList1" runat="server"
         BorderColor="black"
         CellPadding="3"
         Font-Name="Verdana"
         Font-Size="8pt">

        <HeaderStyle BackColor="#aaaadd">
        </HeaderStyle>

        <AlternatingItemStyle BackColor="Gainsboro">
        </AlternatingItemStyle>

        <HeaderTemplate>

            Items
```

```
        </HeaderTemplate>

        <ItemTemplate>

            <%# DataBinder.Eval(Container.DataItem, "StringValue") %>

        </ItemTemplate>

    </asp:DataList>

      <br><br>

    <asp:Button id="Button1"
         Text="显示 DataList1 表第一列的所有内容"
         OnClick="Button_Click"
         runat="server"/>

    <br><br>

    <asp:Label id="Label1"
         runat="server"/>

  </form>

</body>
</html>
```

如果使用控件 DataGrid，有时需要知道指定行的指定列的数据，表达式 DataGrid1.Items[2].Cells[1].Text 表示网格控件 DataGrid1 的第 2 行第 1 列的文本。具体例子如下：

```
<%@ Import namespace="System.Data" %>
<html>
<head>
<script language="C#" runat="server">
  void Page_Load(Object sender, EventArgs e ) {
   if (!Page.IsPostBack) {
     DataTable dt = new DataTable();//建立一个数据表
     DataRow dr;//建立一个数据表的记录变量
     dt.Columns.Add(new DataColumn("整数值", typeof(Int32)));//增加字段
     dt.Columns.Add(new DataColumn("字符串值", typeof(string)));//包括字段名
     dt.Columns.Add(new DataColumn("日期时间值", typeof(DateTime)));//及字段类型
     dt.Columns.Add(new DataColumn("布尔值", typeof(bool)));
     for (int i = 1; i <= 9; i++) {//增加 10 个记录
        dr = dt.NewRow();//建立 1 个记录对象
        dr[0] = i;//为记录的每个字段赋值
```

```
        dr[1] = "项 " + i.ToString();
        dr[2] = DateTime.Now;
        dr[3] = (i % 2 != 0) ? true : false;
        dt.Rows.Add(dr);//把此记录加到数据表中
        }
       dataGrid1.DataSource = new DataView(dt);//为 dataGrid 指定数据源
       dataGrid1.DataBind();//数据更新
       }
    }
   void Button_Click(Object sender, EventArgs e)
   {
    Label1.Text=dataGrid1.Items[2].Cells[1].Text;
   }
</script>
</head>
<body>
    <form runat=server>
        <asp:DataGrid id="dataGrid1" runat="server"
          BorderColor="black"
          BorderWidth="1"
          GridLines="Both"
          CellPadding="3"
          CellSpacing="0"
          HeaderStyle-BackColor="#aaaadd"
        />
      <br><br>
        <asp:Button id="Button1"
           Text="显示 DataGrid1 表第二行第一列的内容"
           OnClick="Button_Click"
           runat="server"/>
      <br><br>
        <asp:Label id="Label1" runat="server"/>

    </form>
</body>
</html>
```

3. Style

使用 DataList 和 DataGraid 的属性 Style 可定义控件的外观。见下例：

```
<asp:DataGrid id="grid" runat="server" visible="false"
        CssClass="Shadow" BackColor="white"
        CellPadding="2" CellSpacing="2" GridLines="none"
        BorderStyle="solid" BorderColor="black" BorderWidth="1"
        font-size="x-small" font-names="verdana">
```

```
        <AlternatingItemStyle BackColor="palegoldenrod" />
        <ItemStyle BackColor="beige" />
        <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
</asp:DataGrid>
```

其中定义了控件 DataGraid 的背景为白色，边界为黑实线，宽度为 1 个象素，字体为"x-small"，字体名字为"verdana"。奇数行的背景颜色，标题的字的颜色，背景色。数据项的背景色等等。显示效果见 7.1.2 的例子。

4. ItemTemplate 属性

控件 DataList 中的 ItemTemplate 是模板控件，其功能是将控件 DataList 的数据源中的所有数据，按 ItemTemplate 模板控件所指定的格式显示。**DataList** 控件中项的外观由 ItemStyle 属性控制。还可以使用 AlternatingItemTemplate 属性来控制 **DataList** 控件中交替项的内容。具体例子见 6.6 节中 3.数据绑定和 Items 集合的创建的例子。

DataGraid 控件没有 ItemTemplate 模板，可使用模板列控件 TemplateColumn，在模板列控件中增加 ItemTemplate 模板自己定义该列的显示控件或显示格式，具体例子见 7.1.6 列类型的第 3 和第 4 个例子。还可以在其中增加 EditItemTemplate 模板自己定义该列在编辑时使用的控件，具体例子见 7.2.5 节例子。

Repeater 控件的 ItemTemplate 模板见 8.4 节例子。

5. 模板中的数据绑定

模板中的数据绑定的例子见上一节中的例子。

6.6.2 使用列表绑定控件

```
<%@ Import Namespace="System.Data" %>

<html>
  <script language = "C#" runat="server">

    ICollection CreateDataSource()
    {
      DataTable dt = new DataTable();
      DataRow dr;

      dt.Columns.Add(new DataColumn("StringValue", typeof(string)));

      for (int i = 0; i < 10; i++)
      {
        dr = dt.NewRow();
        dr[0] = "Item " + i.ToString();
        dt.Rows.Add(dr);
      }

      DataView dv = new DataView(dt);
      return dv;
    }
```

```
      void Page_Load(Object sender, EventArgs e)
      {
         if (!IsPostBack)
         {
            DataList1.DataSource = CreateDataSource();
            DataList1.DataBind();
         }
      }

      void Button1_Click(Object sender, EventArgs e)
      {

         if (DropDown1.SelectedIndex == 0)
            DataList1.RepeatDirection = RepeatDirection.Horizontal;
         else
            DataList1.RepeatDirection = RepeatDirection.Vertical;

         if (DropDown2.SelectedIndex == 0)
            DataList1.RepeatLayout = RepeatLayout.Table;
         else
            DataList1.RepeatLayout = RepeatLayout.Flow;

         DataList1.RepeatColumns=DropDown3.SelectedIndex+1;

         if ((Check1.Checked ==true) && (DataList1.RepeatLayout ==
RepeatLayout.Table))
         {
            DataList1.BorderWidth = Unit.Pixel(1);
            DataList1.GridLines = GridLines.Both;
         }
         else
         {
            DataList1.BorderWidth = Unit.Pixel(0);
            DataList1.GridLines = GridLines.None;
         }
      }

   </script>

<body>

   <form runat=server>

      <h3>DataList Example</h3>
```

```
<asp:DataList id="DataList1" runat="server"
     BorderColor="black"
     CellPadding="3"
     Font-Name="Verdana"
     Font-Size="8pt">

   <HeaderStyle BackColor="#aaaadd">
   </HeaderStyle>

   <AlternatingItemStyle BackColor="Gainsboro">
   </AlternatingItemStyle>

   <HeaderTemplate>

      Items

   </HeaderTemplate>

   <ItemTemplate>

      <%# DataBinder.Eval(Container.DataItem, "StringValue") %>

   </ItemTemplate>

   <AlternatingItemTemplate>

      *
      <%# DataBinder.Eval(Container.DataItem, "StringValue") %>

   </AlternatingItemTemplate>

</asp:DataList>

<p>
<hr noshade align="left" width="300px">

RepeatDirection:

<asp:DropDownList id=DropDown1 runat="server">

   <asp:ListItem>Horizontal</asp:ListItem>
   <asp:ListItem>Vertical</asp:ListItem>
```

```
        </asp:DropDownList><br>

        RepeatLayout:

        <asp:DropDownList id=DropDown2 runat="server">

           <asp:ListItem>Table</asp:ListItem>
           <asp:ListItem>Flow</asp:ListItem>

        </asp:DropDownList><br>

        RepeatColumns:

        <asp:DropDownList id=DropDown3 runat="server">

           <asp:ListItem>1</asp:ListItem>
           <asp:ListItem>2</asp:ListItem>
           <asp:ListItem>3</asp:ListItem>
           <asp:ListItem>4</asp:ListItem>
           <asp:ListItem>5</asp:ListItem>

        </asp:DropDownList><br>

        Show Borders:

        <asp:CheckBox id=Check1 runat="server" /><p>

        <asp:LinkButton id=Button1
             Text="Refresh DataList"
             OnClick="Button1_Click"
             runat="server"/>

    </form>

</body>
</html>
```

如果使用 Visual Studio.Net 实现模板，具体步骤如下：

本节小结

1. ASP.NET 声明性数据绑定语法使用 <%# %> 表示法。
2. 可以绑定到数据源、页或其他控件的属性、集合、表达式以及从方法调用返回的结果。
3. 列表控件可以绑定到支持 **ICollection**、**IEnumerable** 或 **IListSource** 接口的集合，如 **ArrayList**、**Hashtable**、**DataView** 和 **DataReader**。

4. **DataBinder.Eval** 是用于晚期绑定的静态方法。它的语法可能比标准数据绑定语法简单，但性能较低。

10.3 数据验证控件

用户输入了数据，在提交前，首先要对输入的数据进行验证。当然，可以自己编程序进行验证。ASP.NET 提供了一些验证控件，可以不用编程完成对输入的数据进行验证。本节介绍如何使用这些数据验证控件。

10.3.1 数据验证概述

对用户输入的数据进行验证，可以在客户端进行。实现原理是当用户输入了信息并单击提交按钮后，用在客户端运行的 JavaScript 脚本或 VBScript 脚本对数据验证，只有所有数据正确，才能发送到服务器端处理。此种方法的优点是运行在客户端，因此反应速度快，减轻了服务器和网络的负载。缺点是由于 JavaScript 脚本或 VBScript 脚本是以明文的方式嵌入在 HTML 文档中，客户端可以看到这些脚本程序，如果用户把这段脚本删除，网页也就失去了验证功能，因此这种方法是不安全的。

另一种数据验证方法是在服务器端进行，当用户输入了信息并单击提交按钮后，把数据立刻发送到服务器端，在服务器端验证，如果验证不通过，返回错误信息。这种方法虽然在响应速度比较慢，增加了服务器的负担，但可靠性上要强的很多。

ASP.NET 提供了一些验证控件，可以不用编程完成对输入的数据进行验证。下边是一个使用验证控件简单的例子，该例以数据验证控件 RequiredFieldValidator 为例，介绍数据验证控件属性的使用方法。有些数据用户是必须输入的，这些数据可以用编辑控件，单选或多选按钮等控件输入。可以用控件 RequiredFieldValidator 对这些控件输入的数据进行验证，检查用户是否输入了数据。控件 RequiredFieldValidator 的属性 ControlToValidate 的值选择要验证的控件的 id 值，可以是编辑控件，单选或多选按钮等。属性 ErrorMessage 是发生错误时，提示的错误信息。用户用编辑控件 textBox1 输入姓名，要求必须输入。用控件 RequiredFieldValidator1 对其输入进行验证，因此属性 ControlToValidate= textBox1。属性 ErrorMessage=”必须输入姓名”。当单击提交按钮后，如果用户没有输入姓名，则用”必须输入姓名”提示用户。

```
<html>
<body>
    <form id="Form1" method="post" runat="server">
    <p>姓名：<asp:TextBox id=" TextBox1" runat="server"/></p>
    <p><asp:Button Text="提交" runat=server/></p>
    <asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
    ErrorMessage="RequiredFieldValidator" ControlToValidate="RadioButtonList1">
    </asp:RequiredFieldValidator>
    </form>
</body>
</html>
```

10.3.2 常用的验证控件

.Net 框架类库中提供以下几种验证控件：

- RequiredFieldValidator 控件
- 自定义数据验证控件 CustomValidator 控件
- ValidationSummary 控件
- CompareValidator 控件
- RegularExpressionValidator 控件

10.3.3 验证控件常用的属性

- 属性 ControlToValidate：要验证的控件的 id 值。
- 属性 ErrorMessage：发生错误时，提示的错误信息。
- 属性 Display：
- 属性 IsValid：
- 属性 Text：

10.3.4 RequiredFieldValidator

上边已介绍用记事本编辑网页如何使用此控件，下边的例子用 Visual Studio.Net 编辑。该例子增加一个 RadioList 控件，输入卡的类型，增加一个编辑控件，输入编号，两者都要求必须输入，用两个 RequiredFieldValidator 控件验证。步骤如下：

(1)创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。

(2)放工具箱的 Label 控件到窗体，其属性[Text]=“RequiredFieldValidator 控件的使用”。Id=Label1。

(3)放工具箱的 Label 控件到窗体，其属性[Text]=“输入卡号”。

(4)放工具箱的 RadioButtonList 控件到窗体，id=RadioButtonList1。

(5)单击属性 Items 后的按钮，出现集合编辑器对话框。单击添加按钮，增加一个 RadioButton 按钮，修改其 Text 属性为”苹果卡”，修改其 Selected 属性为法 false。用同样方法增加另一个 RadioButton 按钮，修改其 Text 属性为”橡胶卡”，修改其 Selected 属性为法 false。

(6)放工具箱的 Label 控件到窗体，其属性[Text]=“输入编号”。

(7)放工具箱的 TextBox 控件到窗体，id=TextBox1。

(8)放工具箱的 RequiredFieldValidator 控件到窗体，属性 ControlToValidate=RadioButtonList1，属性 ErrorMessage=”必须输入卡类型”。

(9)放工具箱的 RequiredFieldValidator 控件到窗体，属性 ControlToValidate=TextBox1，属性 ErrorMessage=”必须输入编号”。

(10) 放工具箱的 Button 控件到窗体，为其增加单击事件函数如下：

```
private void Button1_Click(object sender, System.EventArgs e)
        {
            if(Page.IsValid==true)
                Label1.Text="已输入了数据";
```

```
            else
                Label1.Text="至少有一项未输入了数据";
        }
```

(11) 如运行出错，把 C:\inetpub\wwwroot\aspnet_client 文件夹拷贝到 D:\Asp 文件夹下。运行

10.3.5 自定义数据验证控件 CustomValidator 控件

CustomValidator 控件允许编程者自己定义一个函数对数据进行验证。一般数据验证分为客户端验证和服务器端验证，可以修改验证控件的属性 ClientTarget 改变在那端验证，例如：Page.ClientTarget=ClientTarget.Downlevel 语句表示要在服务器端验证，而语句 Page.ClientTarget=ClientTarget.Uplevel 表示在客户端验证，在客户端验证必须在发布目录下包含 C:\inetpub\wwwroot\aspnet_client 文件夹。因此，编程者要根据在那一端验证，编写不同的函数，在服务器端验证函数定义如下：

void ServerValidate(object source, ServerValidateEventArgs args){//验证语句}

在客户端验证函数定义如下：？

void ClientValidate(source,value){//验证语句}

书中的例子如下：

```
<%@ Page Language="C#" %>

<html>
<head>


    <script runat=server>

        void ValidateBtn_OnClick(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                lblOutput.Text = "Page is valid.";
            }
            else
            {
                lblOutput.Text = "Page is not valid!";
            }
        }

        void ServerValidate(object source, ServerValidateEventArgs args)
        {
            try
            {
                int i = int.Parse(args.Value);
                args.IsValid = ((i%2) == 0);
```

```
            }

            catch
            {
                args.IsValid = false;
            }
        }

    </script>

</head>
<body>

    <form runat="server">

        <h3>CustomValidator Example</h3>

        <asp:Label id=lblOutput runat="server"
                Text="Enter an even number:"
                Font-Name="Verdana"
                Font-Size="10pt" /><br>

        <p>

        <asp:TextBox id="Text1"
                runat="server" />

        <asp:CustomValidator id="CustomValidator1"
      ControlToValidate="Text1" ErrorMessage="Not an even number!"
      OnServerValidate="ServerValidate" runat="server"/>

        <p>

        <asp:Button id="Button1"
                Text="Validate"
                OnClick="ValidateBtn_OnClick"
                runat="server"/>

    </form>

</body>
</html>
```

</html>

用Visual Studio.NET实现此例，具体步骤如下：

(1)创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。

(2)放工具箱的 Label 控件到窗体，其属性[Text]="CustomValidator 控件的使用"。Id=Label1。

(3)放工具箱的 Label 控件到窗体，其属性[Text]="键入一个偶数"。

(4)放工具箱的 TextBox 控件到窗体，id=TextBox1。

(5)放工具箱的 CustomValidator 控件到窗体，id=CustomValidator1，属性 ControlToValidate=TextBox1，属性 ErrorMessage=Not an even number!。

(6)放工具箱的 Button 控件到窗体，为其增加单击事件函数如下：

```
void ValidateBtn_OnClick(Object sender,EventArgs e)
       {   If (Page.IsValid)
              lblOutput.Text = "Page is Valid!";
           else
              lblOutput.Text = "Page is InValid!";
        }
```

(7)为 CustomValidator 控件 ServerValidate 事件增加事件函数如下

```
private void CustomValidator1_ServerValidate(object source,
System.Web.UI.WebControls.ServerValidateEventArgs args)
        {
            try
            {
                int i = int.Parse(args.Value);
                args.IsValid = ((i%2) == 0);
            }

            catch
            {
                args.IsValid = false;
            }
        }
```

(8)

10.3.6 ValidationSummary 控件

当用户提交了数据后，所有验证控件对数据进行验证，如果没有错误，设置 Page.IsValid=true，否则=false。如果在页面中放置了控件 ValidationSummary，它将自动显示发现错误的数据验证控件的属性 ErrorMessage 的内容。可以在 5.2.1 的例子中，增加一个 ValidationSummary 控件，运行后看一下效果。

10.3.7 CompareValidator 控件

CompareValidator 控件可以对两个控件输入的值进行比较。属性 ControlToValidate=控制对象的 id，属性 ValueToCompare=要比较得值，属性 Type=比较的数据类型，属性 Operator=如何比较，可以是：Equal、NotEqual、GreatrThan、GreatrThanEqual、LessThan、LessThanEqual。用 Visual Studio.NET 实现此例，具体步骤如下：

(1)创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。

(2)放三个 Label 控件到窗体，其属性[Text]分别为"First string"、"Second string"和"lblOutput"。id 分别为 Label1、Label2 和 lblOutput。

(3)在 Label1 和 Label2 控件后，分别放置 TextBox 控件，id 分别为 TextBox1 和 TextBox2。

(4)放工具箱的 CompareValidator 控件到窗体，属性 ControlToValidate=TextBox1，属性 ControlToCompare=TextBox2，属性 ErrorMessage=Not an even number!，id=Compare1。

(5)放两个 ListBox 控件到窗体，id 分别为 ListOperator 和 ListType。

(6)单击 ListOperator 属性 Items 旁的三个小点，在 ListItem 编辑器对话框中单击添加按钮，增加 6 个选项，属性 Text 分别为：Equal、NotEqual、GreatrThan、GreatrThanEqual、LessThan、LessThanEqual，同时，属性 Value 也变为相应的值。

(7)单击 ListType 属性 Items 旁的三个小点，在 ListItem 编辑器对话框中单击添加按钮，增加 5 个选项，属性 Text 分别为：String、Integer、Double、Date、Currency，同时，属性 Value 也变为相应的值。

(8)设置两个 ListBox 控件属性 SelectionMode 为 Single，不允许多选。(为什么？)

(9)设置两个 ListBox 控件属性 AutoPostBack=true。

(10) 为 ListOperator 事件(SelectedIndexChenged)增加事件函数如下：

```
private void ListOperator_SelectedIndexChanged(object sender, System.EventArgs e)
        {
            Compare1.Operator=(ValidationCompareOperator)ListOperator.SelectedIndex;
            Compare1.Validate();
        }
```

(11) 为 ListType 事件(SelectedIndexChenged)增加事件函数如下：

```
private void ListType_SelectedIndexChanged(object sender, System.EventArgs e)
        {
            Compare1.Type=(ValidationDataType)ListType.SelectedIndex;
            Compare1.Validate();
        }
```

(12) 放工具箱的 Button 控件到窗体，为其增加单击事件函数如下：

```
private void Button1_Click(object sender, System.EventArgs e)
        {
            //Label1.Text=DropDownList1.SelectedItem.Text;
            if(Page.IsValid==true)
                lblOutput.Text="数据有效";
            else
                lblOutput.Text="数据无效";
        }
```

(13) 运行，看一下效果。增加列表内容用如下语句

```
    listBox1.Items.Add("Item9");//在列表最后增加一项
    listBox1.SetSelected(1, true);//第一项被选中
  listBox1.EndUpdate();//更新
```

5.2.5 **RangeValidator** 控件

RangeValidator 控件测试输入控件的值是否在指定范围内。**RangeValidator** 控件使用四个关键属性执行验证。ControlToValidate 属性包含要验证的输入控件。MinimumValue 和 MaximumValue 属性指定有效范围的最大值和最小值。BaseCompareValidator.Type 属性用于指定要比较的值的数据类型。在执行验证操作之前，要比较的值被转换为此数据类型。可以进行比较的不同数据类型：字符串数据类型 **String**、32 位有符号整数数据类型 **Integer**、双精度浮点数数据类型 **Double**、日期数据类型 **Date**、一种可以包含货币符号的十进制数据类型 **Currency**。

下面的示例说明如何在 Web 页上创建 **RangeValidator** 控件，以检查输入到输入控件的值是否在比较范围内。

```
<%@ Page Language="C#" %>
<html>
<head>
   <script runat="server">
      void ButtonClick(Object sender, EventArgs e)
      {
         if (Page.IsValid)
         {
            Label1.Text="Page is valid.";
         }
         else
         {
            Label1.Text="Page is not valid!!";
         }
      }
   </script>
</head>

<body>
   <form runat="server">
      <h3>RangeValidator Example</h3>
      Enter a number from 1 to 10:
      <br>
      <asp:TextBox id="TextBox1"
           runat="server"/>
      <br>
      <asp:RangeValidator id="Range1"
           ControlToValidate="TextBox1"
           MinimumValue="1"
```

```
            MaximumValue="10"
            Type="Integer"
            EnableClientScript="false"
            Text="The value must be from 1 to 10!"
            runat="server"/>
        <br><br>
        <asp:Label id="Label1"
            runat="server"/>
        <br><br>
        <asp:Button id="Button1"
            Text="Submit"
            OnClick="ButtonClick"
            runat="server"/>
    </form>
</body>
</html>
```

10.3.8 RegularExpressionValidator 控件

RegularExpressionValidator 控件也叫正则表达式控件，该控件用来检查输入控件的值是否匹配正则表达式定义的模式。这类验证允许您检查可预知的字符序列，比如身份证号码、电子邮件地址、电话号码和邮编中的字符序列。本节首先讲解一些正则表达式的基本知识，然后将这些基本知识用于数据验证控件。

1．基本模式

模式，是正规表达式最基本的元素，它们是一组描述字符串特征的字符。模式可以很简单，由普通的字符串组成；也可以非常复杂，可以用特殊的字符表示一个范围内的字符、重复出现等。最简单的匹配就是一个字符串，这种情况下，如果一个字符串含有这个字符串，那么那个字符串就认为是符合匹配要求的。

- “^”头匹配

这个模式包含一个特殊的字符 ^，表示该模式只匹配那些以紧接其后的以字符串。例如：^front，表示以”front”开头的字符串是匹配的，而不以”front”开头的字狩串是不匹配的。

- “$”尾匹配

尾匹配“$”的意义是，只有那些以“$”号前面的字符串结尾的字符串才符合匹配的要求，例如：tail$，表示那些以”tail”结尾的字符串是匹配的。结合使用“^”和“S”，可以提供一种整个字符串匹配的模式：^whole$，就表示仅有“whole”字符串符合匹配要求。

2．转义序列

所谓转义序列其实就是一些无法直接在正则表达式中使用的字符，例如标点符号、空格、回车、换行、制表符等。所有的转义序列都用反斜杠(\)打头。和“C”语言中类似，转义序列也是以“\”开头的，如：

换行：\n

回车：\r

制表符：\t

由于在正则表达式中“ ^ ”、“S”和“＋”等都有特殊的意义，所以在正则表达式中也需要使用转义序列来表示“ \ ^ ”“ \ $ ”和“ \ ＋”。

3. 字符簇

在 Internet 中，正规表达式通常用来舰用户的输人,当用户提交一个 Form 以后，要判断输入的电话号码、地址、Enail 地址、信用卡号码、邮政编码等是否有效，用普通的基于字面的字符是不够的，需要一种可以设定一个字符集合的方法，在正则表达式中，这种方法称为字符簇，一个字符簇是使用方挂号括起来的。下面举一个例子：[ABCabc],上面的字符簇例子表示如果一个字符是“A”或“B”或“C“或“a”或“b”或“C“,那么就符合匹配要求。

当需要一个有顺序的返回的时候，可以使用连字号来表示二个字符的范围，如：[a-z]匹配所有的小写字母。[A-Z]匹配所有的大写字母。［az-AZ］匹配所有字母。[0－9]匹配所有的数字。［0－9\．\-］ 匹配所有的数字，句号和减号。［\f\r\t\n］匹配所有的白字符。

前面曾经提到 ^ 表示字符串的开头，但它还有另外一个含义。当在一组方括号里使用 ^ 时，它表示“非”或“排除”的意思，常常用来剔除某个字符。例如，如果要求第一个字符不能是小写字母：^[^a-z], 这个模式与“A4”、“7b”及“+a”是匹配的，但与”a2’、“c6”是不匹配的。下面是几个排除特定字符字符的例子：[“a－z”]，除了小写字母以外的所有字符。[^“0－9”]，除了数字之外的所有字符。[^\”\’]，除了双引号（”）和单引号（’）之外的所有字符。特殊字符”．”（点号）在正规表达式中用来表示除了“新行”之外的所有字符。所以模式”^．5$”与任何两个字符的、以数字 5 结尾和以其他非“新行”字符开头的字符串匹配。模式”.”可以匹配任何字符串，除了空串和只包括一个“新行”的字符串。

4. 重复

到现在为止，已经讨论了如何去匹配一个字母或数字，但更多的情况下，可能要匹配一个单词或一组数字。一个单词由若干个字母组成，一组数字由若干个单数组成。

正则表达式提供了“｛｝”来执行重复，跟在字符或字符簇后面的花括号（｛｝）用来确定前面的内容重复出现的次数。其中｛n，m｝表示可能重复 n 到 m 次并且包括 n 和 m 次，｛n，｝表示可能重复 n 次或大于 n 次。例如：^a{4}$表示 aaaa。^a(2，4）$表示 aa,aaa,或 aaaa。^a{2,}$表示包含多于两个 a 的字符串。.{2}表示所有的两个字符。

下面是常用的一些模式：

^[a-zA-Z0-9_]{1,}$表示所有包含一个以上的字母、数字或下划线的字符串。

^[0-9]{1,}$ 表示所有的整数。

^\-{0,1}[0-9]{0,}\.{0,1}[0-9]{0,}$ 表示所有小数。

正则表达式提供了一些简写的特殊字符，可以让表达式容易理解：

?	{0，1}
*	{0，}
+	{1，}

例如：^[0-9]+$ 表示所有的整数。^[0-9]+$ 表示所有的整数。^\-?[0-9]*\.?[0-9]+$表示所有小数。

下例说明如何使用 **RegularExpressionValidator** 验证一个 5 位数的邮编。

```
<%@ Page Language="C#" %>
 <html>
```

```
<head>
   <script runat="server">
       void ValidateBtn_Click(Object sender, EventArgs e)
      {
         if (Page.IsValid)
         {
            lblOutput.Text = "Page is Valid!";
         }
         else
         {
            lblOutput.Text = "Page is InValid! :";
         }
      }
    </script>
 </head>

<body>
    <h3>RegularExpressionValidator Example</h3>
   <p>
    <form runat="server">
       <table bgcolor="#eeeeee" cellpadding="10">
         <tr valign="top">
            <td colspan="3">
               <asp:Label ID="lblOutput"
                     Text="Enter a 5 digit zip code"
                     runat="server"/>
            </td>
         </tr>
         <tr>
            <td colspan="3">
               <b>Personal Information</b>
            </td>
         </tr>
         <tr>
            <td align="right">
               Zip Code:
            </td>
            <td>
               <asp:TextBox id="TextBox1"
                     runat="server"/>
            </td>
            <td>
               <asp:RegularExpressionValidator id="RegularExpressionValidator1"
                     ControlToValidate="TextBox1"
```

```
                    ValidationExpression="\d{5}"
                    Display="Static"
                    ErrorMessage="Zip code must be 5 numeric digits"
                    EnableClientScript="False"
                    runat="server"/>
            </td>
          </tr>
          <tr>
            <td></td>
            <td>
               <asp:Button text="Validate"
                    OnClick="ValidateBtn_Click"
                    runat=server />
            </td>
            <td></td>
          </tr>
       </table>
     </form>
 </body>
 </html>
```

如用 Visual Studio.NET 实现，只需修改属性 ValidationExpressio 即可。

10.4 DataGraid 控件

10.4.1 DataGrid 控件概述

10.4.2 DataGrid 控件绑定数据库表

例子 e10_4_1 联接一个数据库，用 DataGrid 控件显示的例子如下：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>DataGrid 显示数据库表</title>
<script runat=server>
public void Page_Load(Object sender, EventArgs e)
{
        string txtConn="DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
```

```
        string txtCommand="SELECT employeeid, firstname, lastname FROM
Employees";
        SqlConnection conn = new SqlConnection(txtConn);
        SqlDataAdapter da = new SqlDataAdapter(txtCommand, conn);
        DataSet ds = new DataSet();
        da.Fill(ds, "MyTable");
        grid.DataSource =ds.Tables["MyTable"];//为网格控件指定数据源
        grid.DataBind();//数据绑定
//      grid.Visible =true;
}
</script>
<body bgcolor="ivory" style="font-family:arial;font-size:9pt">
<form runat=server>
    <asp:DataGrid id="grid" runat="server" visible="false"
        AutoGenerateColumns="true"
        CssClass="Shadow" BackColor="white"
        CellPadding="2" CellSpacing="2" GridLines="none"
        BorderStyle="solid" BorderColor="black" BorderWidth="1"
        font-size="x-small" font-names="verdana">
        <AlternatingItemStyle BackColor="palegoldenrod" />
        <ItemStyle BackColor="beige" />
        <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
     </asp:DataGrid>
</form>
</body>
</html>
```

其中字符串 txtConn="DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;"叫联接字符串，DATABASE 时数据源，SERVER 是数据库服务器，UID 是用户名，PWD 为密码。语句 SqlConnection conn=new SqlConnection(txtConn)用来建立一个连接。字符串 txtCommand ="SELECT employeeid, firstname, lastname FROM Employees"是一个 SQL 语句，用来从表 Employees 中取出字段 employeeid、firstname、lastname 的所有记录。语句 SqlDataAdapter da=new SqlDataAdapter(txtCommand,conn)用来建立一个 SqlDataAdapter 对象，它负责读取数据库数据。用 DataGrid 控件显示表 Employees 中字段 employeeid、firstname、lastname 的数据。Page_Load()函数说明了显示数据的必要步骤。特别要注意 DataGrid 控件数据绑定的方法。

10.4.3 DataGrid 控件对数据库记录分页显示

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Data" %>

<html>
<title>分页</title>
```

```
<script runat="server">
public void Page_Load(Object sender, EventArgs e)
{
        DataGrid1.DataSource = CreateDataSource();
        DataGrid1.DataBind();
}

private DataTable CreateDataSource()
{
        String strConn = "DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
        String strCmd ="SELECT firstname, lastname FROM employees";
        SqlDataAdapter cmd = new SqlDataAdapter(strCmd, strConn);

        DataSet oDS = new DataSet();
        cmd.Fill(oDS, "EmployeesList");

        return oDS.Tables["EmployeesList"];
}

public void PageIndexChanged(Object sender, DataGridPageChangedEventArgs e)
{
        DataGrid1.CurrentPageIndex = e.NewPageIndex;
        DataGrid1.DataSource = CreateDataSource();
        DataGrid1.DataBind();
}

</script>


<body bgcolor="ivory" style="font-family:arial;font-size:xsmall">

<!-- ASP.NET topbar -->
<h2>分页</h2>
<form runat="server">

<asp:datagrid runat="server" id="DataGrid1"
        Font-Size="Smaller" Font-Names="Verdana"
        CellPadding="2" CellSpacing="2"
        CssClass="Shadow" BackColor="White"
        BorderStyle="solid" BorderColor="black" BorderWidth="1"
        AllowPaging="True"
        PageSize="3"
        OnPageIndexChanged="PageIndexChanged">
```

```
        <PagerStyle PageButtonCount="3" Font-Bold="true" Mode="NumericPages"
BackColor="palegreen" />
        <AlternatingItemStyle BackColor="Gainsboro" />
        <ItemStyle BackColor="White" />
        <HeaderStyle Font-Bold="True" ForeColor="White" BackColor="Navy" />
</asp:datagrid>

</form>
</body></html>
```

AllowPaging="True"表示允许分页，PageSize="3"表示每页 3 个记录，<PagerStyle PageButtonCount="3" Font-Bold="true" Mode="NumericPages" BackColor="palegreen" />中 PageButtonCount="3"表示有 3 个按钮，Mode="NumericPages"表示按钮形式为数字，也可以修改为 Mode="NextPrev"，则在网格下部出现<和>，单击<，转向前页，单击>，转向后页。也可以改为字符，例如修改为“前页”和“后页”，可以修改属性 PrevPageText=”前页”，NextPageText=”后页”。OnPageIndexChanged="PageIndexChanged"表示单击按钮事件函数是 PageIndexChanged。

10.4.4 DataGrid 控件对记录排序

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>按列排序</title>
<script runat="server">
    SqlConnection conn;
    protected void Page_Load(Object Src, EventArgs E)
    {
        conn = new SqlConnection("server=localhost;uid=sa;pwd=;database=pubs");
        if (!IsPostBack)
            BindGrid("au_id");
    }
    protected void MyDataGrid_Sort(Object sender, DataGridSortCommandEventArgs e)
    {
        BindGrid(e.SortExpression);
    }
    public void BindGrid(String sortfield)
    {
      SqlDataAdapter myCommand = new SqlDataAdapter("select * from Authors",conn);
      DataSet ds = new DataSet();
      myCommand.Fill(ds, "Authors");
      DataView Source = ds.Tables["Authors"].DefaultView;
      Source.Sort = sortfield;
```

```
      MyDataGrid.DataSource=Source;
      MyDataGrid.DataBind();
    }
</script>
<body>
  <h3><font face="Verdana">按列排序</font></h3>
  <form runat="server">
    <ASP:DataGrid id="MyDataGrid" runat="server" OnSortCommand="MyDataGrid_Sort"
      Width="700"
      BackColor="#ccccff"
      BorderColor="black"
      ShowFooter="false"
      CellPadding=3
      CellSpacing="0"
      Font-Name="Verdana"
      Font-Size="8pt"
      HeaderStyle-BackColor="#aaaadd"
      AllowSorting="true"
    />
  </form>
</body>
</html>
```

AllowSorting="true"表示允许排序，允许排序的列标题有一下划线，单击标题将产生事件，事件函数由 OnSortCommand="MyDataGrid_Sort"定义。现如果用手工产生每一列，排序的例子如下：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<title>按列排序</title>
<script runat="server">

    SqlConnection conn;

    protected void Page_Load(Object Src, EventArgs E)
    {
        conn = new
SqlConnection("DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;");

        if (!IsPostBack)
            BindGrid("lastname");
    }
```

```
    protected void MyDataGrid_Sort(Object sender, DataGridSortCommandEventArgs e)
    {
        BindGrid(e.SortExpression);
    }

    public void BindGrid(String sortfield)
    {
        SqlDataAdapter myCommand = new SqlDataAdapter("SELECT employeeid,
firstname, lastname, title, country FROM Employees", conn);

        DataSet ds = new DataSet();
        myCommand.Fill(ds, "MyList");

        DataView Source = ds.Tables["MyList"].DefaultView;
        Source.Sort = sortfield;

        MyDataGrid.DataSource=Source;
        MyDataGrid.DataBind();
    }

</script>

<body>

  <h3><font face="Verdana">按列排序</font></h3>

  <form runat="server">

    <asp:DataGrid id="MyDataGrid" runat="server"
         AutoGenerateColumns="false"
         CssClass="Shadow" BackColor="white"
         CellPadding="2" CellSpacing="2" GridLines="none"
         BorderStyle="solid" BorderColor="black" BorderWidth="1"
         font-size="x-small" font-names="verdana"
         AllowSorting="true"
         OnSortCommand="MyDataGrid_Sort">

         <AlternatingItemStyle BackColor="palegoldenrod" />
         <ItemStyle BackColor="beige" />
         <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />

        <columns>
            <asp:BoundColumn runat="server" DataField="employeeid" HeaderText="
编号" >
```

```
            <itemstyle backcolor="lightblue" font-bold="true" />
        </asp:BoundColumn>
        <asp:BoundColumn runat="server" DataField="firstname"
            HeaderText="名字" />
        <asp:BoundColumn runat="server" DataField="lastname"
            HeaderText="姓" SortExpression="lastname" />
        <asp:BoundColumn runat="server" DataField="title"
            HeaderText="Position" />
        <asp:BoundColumn runat="server" DataField="country"
            HeaderText="国家" SortExpression="country" />
      </columns>
    </asp:DataGrid>

  </form>

</body>
</html>
```

在 DataGrid 的 5 列中，只有 DataField="lastname"和 DataField="country"有 SortExpression 属性定义，分别为："lastname"和"country"，因此，只有此两列可以排序，表达式 SortExpression= "lastname"表示此列可以排序，按字段"lastname"排序。如果增加语句 Source.Sort += " DESC"，则降序排序，如无字符" DESC"为生序排序。

10.4.5 用 BoundColumn 列将标题改为中文

DataGrid 控件的属性 AutoGenerateColumns=true，将根据数据源的内容自动填充表格，标题默认为是字段名，由于避免不兼容，字段名一般用英文，如希望如将标题改为中文，可以置 AutoGenerateColumns=false，使用列控件 BoundColumn 手工填充，其中属性 HeaderText 是标题字符串，可以改成中文，属性 DataField 是该列显示的字段名。见下例：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>DataGrid 标题改为中文</title>
<script runat=server>
public void Page_Load(Object sender, EventArgs e)
{
        string txtConn="DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
        string txtCommand="SELECT employeeid, titleofcourtesy, firstname,
lastname, title, country FROM Employees";
        SqlConnection conn = new SqlConnection(txtConn);
        SqlDataAdapter da = new SqlDataAdapter(txtCommand, conn);
        DataSet ds = new DataSet();
        da.Fill(ds, "MyTable");
        grid.DataSource = ds.Tables["MyTable"];
```

```
//       grid.DataBind();
}
</script>
<body bgcolor="ivory" style="font-family:arial;font-size:9pt">
<form runat=server>
   <asp:DataGrid id="grid" runat="server"
         AutoGenerateColumns="false"
         CssClass="Shadow" BackColor="white"
         CellPadding="2" CellSpacing="2" GridLines="none"
         BorderStyle="solid" BorderColor="black" BorderWidth="1"
         font-size="x-small" font-names="verdana">
         <AlternatingItemStyle BackColor="palegoldenrod" />
         <ItemStyle BackColor="beige" />
         <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
     <columns>
       <asp:BoundColumn runat="server" DataField="employeeid" HeaderText="编号">
           <itemstyle backcolor="lightblue" font-bold="true" />
       </asp:BoundColumn>
      <asp:BoundColumn runat="server" DataField="titleofcourtesy" HeaderText="
称呼" />
       <asp:BoundColumn runat="server" DataField="firstname" HeaderText="姓" />
       <asp:BoundColumn runat="server" DataField="lastname" HeaderText="名" />
     </columns>
   </asp:DataGrid>
</form>
</body>
</html>
```

10.4.6 增加按钮列

如果用手工创建 DataGrid 表格，除了以上介绍的列控件 BoundColumn 外，还包括控件 ButtonColumn，用来创建一列按钮，可以为按钮增加一个事件函数，控件 HyperLinkColumn 用来创建一列超级链接字符，控件 EditCommandColumn，将自动和编辑命令相关联。控件 TemplateColumn，将按指定模板创建所显示的列。

例子 e10_4_4 是控件 ButtonColumn 的用法：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>DataGrid 增加按钮列</title>
<script runat=server>
string txtConn="DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
public void Page_Load(Object sender, EventArgs e)
{
```

```
        string txtCommand="SELECT employeeid, firstname, lastname FROM Employees";
        SqlConnection conn = new SqlConnection(txtConn);
        SqlDataAdapter da = new SqlDataAdapter(txtCommand, conn);
        DataSet ds = new DataSet();
        da.Fill(ds, "MyTable");
        grid.DataSource = ds.Tables["MyTable"];
        grid.DataBind();
}
public void HandleCommands(Object sender, DataGridCommandEventArgs e)
{
         if(e.CommandName=="moreinfo")
         {
           int nEmpID=(int)grid.DataKeys[e.Item.ItemIndex];
           SqlConnection conn=new SqlConnection(txtConn);
           String strCmd="SELECT * FROM Employees "+
                         "WHERE employeeid="+nEmpID.ToString();
           SqlCommand cmd=new SqlCommand(strCmd,conn);
           conn.Open();
           SqlDataReader dr=cmd.ExecuteReader();
           dr.Read();
           MoreInfo.Text=BuildMoreInfoText(dr);

                  dr.Close();
                  conn.Close();
         }
}
private String BuildMoreInfoText(SqlDataReader dr)
{
        StringBuilder sb=new StringBuilder();
        sb.Append("<b>");
        sb.Append(dr["lastname"]+"," + dr["firstname"]);
        sb.Append("</b><br>");
        sb.Append(dr["title"] + "<hr>");
        sb.Append("<i>");
        sb.Append(dr["notes"]);
        sb.Append("</i>");
        return sb.ToString();
}
</script>
<body bgcolor="ivory" style="font-family:arial;font-size:9pt">
<form runat=server>
  <asp:DataGrid id="grid" runat="server"
     AutoGenerateColumns="false"
     CssClass="shadow" BackColor="white"
```

```
    CellPadding="2" CellSpacing="2" GridLines="none"
    BorderStyle="solid" BorderColor="black" BorderWidth="1"
    font-size="x-small" font-names="verdana"
    DataKeyField="employeeid"
    OnItemCommand="HandleCommands">
    <AlternatingItemStyle BackColor="palegoldenrod" />
    <ItemStyle BackColor="beige" />
    <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true"/>
  <columns>
    <asp:BoundColumn runat="server" DataField="employeeid" HeaderText="编号"/>
    <asp:BoundColumn runat="server" DataField="firstname" HeaderText="名" />
    <asp:BoundColumn runat="server" DataField="lastname" HeaderText="姓" />
    <asp:ButtonColumn runat="server" Text="更多信息" CommandName="moreinfo">
        <itemstyle backcolor="lightblue" font-bold="true" />
    </asp:ButtonColumn>
  </columns>
 </asp:DataGrid>
<asp:Label runat="server" id="MoreInfo" />
</form>
</body>
</html>
```

网页中 OnItemCommand="HandleCommands"的 HandleCommands 是按钮列按钮的事件函数，如果有多个按钮列，都用此函数响应。按钮列按钮的事件函数 HandleCommands 中的语句 if(e.CommandName=="moreinfo")是判断是哪一个按钮发的事件。

10.4.7 增加 HyperLinkColumn 列

例子 e10_4_5 下例是控件 HyperLinkColumn 的用法：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>DataGrid 增加 HyperLinkColumn 列</title>
<script runat=server>
  string txtConn="DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
  public void Page_Load(Object sender, EventArgs e)
  {
    string txtCommand="SELECT employeeid, firstname, lastname FROM Employees";
    SqlConnection conn = new SqlConnection(txtConn);
    SqlDataAdapter da = new SqlDataAdapter(txtCommand, conn);
    DataSet ds = new DataSet();
    da.Fill(ds, "MyTable");
    grid.DataSource = ds.Tables["MyTable"];
    grid.DataBind();
```

```
}
</script>
<body bgcolor="ivory" style="font-family:arial;font-size:9pt">
<form runat=server>
  <asp:DataGrid id="grid" runat="server"
     AutoGenerateColumns="false"
     CssClass="shadow" BackColor="white"
     CellPadding="2" CellSpacing="2" GridLines="none"
     BorderStyle="solid" BorderColor="black" BorderWidth="1"
     font-size="x-small" font-names="verdana"
     DataKeyField="employeeid">
     <AlternatingItemStyle BackColor="palegoldenrod" />
     <ItemStyle BackColor="beige" />
     <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
  <columns>
     <asp:BoundColumn runat="server" DataField="employeeid" HeaderText="编号" />
     <asp:BoundColumn runat="server" DataField="firstname" HeaderText="名" />
     <asp:BoundColumn runat="server" DataField="lastname" HeaderText="姓" />
     <asp:HyperLinkColumn runat="server"
          HeaderText="链接另一网页"
          DataNavigateUrlField="employeeid"
          DataNavigateUrlFormatString="e10_4_5A.aspx?id={0}"
          DataTextField="lastname"
          DataTextFormatString="关于{0}更多信息"
          Target="frInfo">
          <ItemStyle BackColor="lightblue" font-bold="true" />
     </asp:HyperLinkColumn>
   </columns>
 </asp:DataGrid>
<asp:Label runat="server" id="MoreInfo" />
</form>
</body>
</html>
```

控件 HyperLinkColumn 中，DataNavigateUrlFormatString="例子 e10_4_5A.aspx? id ={0}"是超级链接的网页，本例是 c716-1A.aspx，?id={0}是传递的参数，{0}是一个变量，对应 DataNavigateUrlField 指定的数据库表字段，本例为 employeeid，DataTextFormatString 为超级链接字符，本例为"关于{0}更多信息"。

例子 e10_4_5A.aspx 如下：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>More Info</title>
<script runat=server>
```

```
public void Page_Load(Object sender, EventArgs e)
{
   int nEmpID = Convert.ToInt16(Request["id"]);
   if (nEmpID<1)
   {
     Response.Write("No valid ID specified.");
     Response.End();
   }
   else
     GetMoreInfo(nEmpID);
}
public void GetMoreInfo(int nEmpID)
{
   String strConn = "DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
   SqlConnection conn = new SqlConnection(strConn);
   String strCmd = "SELECT * FROM Employees " +
                "WHERE employeeid = " + nEmpID.ToString();
   SqlCommand cmd = new SqlCommand(strCmd, conn);
   conn.Open();
   SqlDataReader dr = cmd.ExecuteReader();
   dr.Read();
   MoreInfo.Text = BuildMoreInfoText(dr);
   dr.Close();
   conn.Close();
}
private String BuildMoreInfoText(SqlDataReader dr)
{
   StringBuilder sb = new StringBuilder();
   sb.Append("<b>");
   sb.Append(dr["employeeid"] + " - ");
   sb.Append(dr["lastname"] + ", " + dr["firstname"]);
   sb.Append("</b><br>");
   sb.Append(dr["title"] + "<hr>");
   sb.Append("<i>");
   sb.Append(dr["notes"]);
   sb.Append("</i>");
   return sb.ToString();
}
</script>
<body>
<asp:Label runat="server" id="MoreInfo" />
</body>
</html>
```

10.4.8 增加 EditCommandColumn 列

10.4.9 控件 TemplateColumn 的用法

例子 e10_4_5A 是控件 TemplateColumn 的用法，本例用单选按钮显示 bool 字段。

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>显示布尔变量</title>
<script runat="server">
public void Page_Load(Object sender, EventArgs e)
{
   if(!Page.IsPostBack)
   {
     SqlConnection conn=new
SqlConnection("DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;");
     SqlDataAdapter da=new SqlDataAdapter("SELECT employeeid, titleofcourtesy,
firstname, lastname, title, ISNULL(reportsto,0) AS boss FROM Employees", conn);
     DataSet ds = new DataSet();
     da.Fill(ds, "MyTable");
     grid.DataSource = ds.Tables["MyTable"];

     grid.DataBind();
   }
}
bool HasBoss(int bossID)
{
   if (bossID != 0)
       return true;
   return false;
}
</script>
<body bgcolor="ivory" style="font-family:arial;font-size:9pt">
<form runat="server">
   <asp:DataGrid id="grid" runat="server"
       AutoGenerateColumns="false"
       CssClass="Shadow" BackColor="white"
       CellPadding="2" CellSpacing="0"
       BorderStyle="solid" BorderColor="black" BorderWidth="1"
       font-size="x-small" font-names="verdana">
```

```
      <AlternatingItemStyle BackColor="palegoldenrod" />
      <ItemStyle BackColor="beige" />
      <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
    <columns>
      <asp:BoundColumn runat="server" HeaderText="ID" DataField="编号">
      <itemstyle backcolor="lightblue" font-bold="true" />
      </asp:BoundColumn>
      <asp:TemplateColumn runat="server" HeaderText="Employee Name">
        <itemtemplate>
        <asp:label runat="server"
        style="margin-left:5;margin-right:5"
        Text='<%# DataBinder.Eval(Container.DataItem,"TitleOfCourtesy")+"<b>"+
              DataBinder.Eval(Container.DataItem,"LastName")+"</b>"+","+
              DataBinder.Eval(Container.DataItem,"FirstName") %>' />
        </itemtemplate>
      </asp:TemplateColumn>
      <asp:TemplateColumn HeaderText="Reports"
            headerstyle-horizontalalign="Center"
            itemstyle-horizontalalign="Center">
      <itemtemplate>
      <asp:checkbox runat="server" enabled="false" checked=
          '<%# HasBoss((int)DataBinder.Eval(Container.DataItem,"boss")) %>'/>
      </itemtemplate>
      </asp:TemplateColumn>
      <asp:BoundColumn runat="server" DataField="title" HeaderText="Position" />
    </columns>
</asp:DataGrid>
</form>
</body>
</html>
```

例子 e10_4_5B 用图形按钮显示 bool 字段。

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<title>用图像显示布尔变量</title>
<script runat="server">
public void Page_Load(Object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
      SqlConnection conn = new
SqlConnection("DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;");
```

```
        SqlDataAdapter da = new SqlDataAdapter("SELECT employeeid, titleofcourtesy,
firstname,lastname,title,ISNULL(reportsto,0)AS boss FROM Employees",conn);
        DataSet ds=new DataSet();
        da.Fill(ds,"MyTable");
        grid.DataSource = ds.Tables["MyTable"];
        grid.DataBind();
     }
}
String GetProperGifFile(int bossID)
{
   if (bossID != 0)
        return "checked.gif";//checked.gif 应和 e10_4_5B.aspx 在同一文件夹中
   return "unchecked.gif";
}
</script>
<body bgcolor="ivory" style="font-family:arial;font-size:9pt">
<asp:DataGrid id="grid" runat="server"
     AutoGenerateColumns="false"
     CssClass="Shadow" BackColor="white"
     CellPadding="2" CellSpacing="0"
     BorderStyle="solid" BorderColor="black" BorderWidth="1"
     font-size="x-small" font-names="verdana">
     <AlternatingItemStyle BackColor="palegoldenrod" />
     <ItemStyle BackColor="beige" />
     <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
  <columns>
     <asp:BoundColumn runat="server" HeaderText="ID" DataField="employeeid">
       <itemstyle backcolor="lightblue" font-bold="true" />
     </asp:BoundColumn>
     <asp:TemplateColumn runat="server" HeaderText="Employee Name">
         <itemtemplate>
            <asp:label runat="server"
              style="margin-left:5;margin-right:5"
            Text='<%#DataBinder.Eval(Container.DataItem,"TitleOfCourtesy")+"<b>"
            +DataBinder.Eval(Container.DataItem,"LastName")+"</b>"+","+
                  DataBinder.Eval(Container.DataItem, "FirstName") %>' />
         </itemtemplate>
     </asp:TemplateColumn>
     <asp:TemplateColumn HeaderText="Reports"
         headerstyle-horizontalalign="Center"
         itemstyle-horizontalalign="Center">
         <itemtemplate>
            <asp:image runat="server" imageurl='<%#
         GetProperGifFile((int)DataBinder.Eval(Container.DataItem,"boss"))%>'/>
```

```
        </itemtemplate>
    </asp:TemplateColumn>
    <asp:BoundColumn runat="server" DataField="title" HeaderText="Position" />
        </columns>
    </asp:DataGrid>

</form>

</body>
</html>
```

7.1.9 使用自己的数据库

- 用 Access 建立数据库

用 Access 建立数据库：db1.mdb。建立 Student 表，记录所有学生信息。包括字段 StudentNum(学生编号)，字节类型，必填字段，默认值为空，StudentNum 为主关键字。字段 StudentName(学生姓名)，文本，字段大小 8，必填字段，默认值为空。字段 StudentSex(性别)，文本，字段大小 2。增加若干数据。存数据库的路径为：D:\asp\StudentDb.mdb

- 例子如下：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="System.Data" %>

<html>
<title>DataGrid</title>

<script runat=server>
public void OnLoadData(Object sender, EventArgs e)
{
    string                          txtConn="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\\ASP\\bookExample\\db1.mdb";
    string txtCommand="SELECT * FROM student";
    OleDbConnection conn = new OleDbConnection(txtConn);
    OleDbDataAdapter da = new OleDbDataAdapter(txtCommand, conn);

    DataSet ds = new DataSet();
    da.Fill(ds, "MyTable");

    // Display the data
    grid.DataSource = ds.Tables["MyTable"];
    grid.DataBind();
    grid.Visible = true;
}
</script>
```

```
<body bgcolor="ivory" style="font-family:arial;font-size:9pt">

<!-- ASP.NET topbar -->
<h2>数据库的联接</h2>

<form runat=server>
    <asp:linkbutton runat="server" id="btnLoad" text="Go get data..."
onclick="OnLoadData" />

    <asp:DataGrid id="grid" runat="server" visible="false"
    AutoGenerateColumns="true"
    CssClass="Shadow" BackColor="white"
    CellPadding="2" CellSpacing="2" GridLines="none"
    BorderStyle="solid" BorderColor="black" BorderWidth="1"
    font-size="x-small" font-names="verdana">

    <AlternatingItemStyle BackColor="palegoldenrod" />
    <ItemStyle BackColor="beige" />
    <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
     </asp:DataGrid>

</form>

</body>
</html>
```

10.5 AdRotator 控件

Web 页上的广告通常采用广告条(小图片)的形式，单击时使用户重定向到广告商的 Web 页。使用 AdRotator Web 服务器控件能够显示广告条并在一系列广告条间循环。AdRotator 自动进行循环处理，在每次刷新页面时更改显示的广告。使用 AdRotator 控件显示广告步骤：

(1) 创建一个 Web 应用程序框架，项目名为 UseAdRotator。

(2) 新建一个 XML 文件。单击菜单项"项目/添加新项"，弹出标题为添加新项的窗口，在窗口中选中 XML 文件，文件名为 ads.xml，单击打开按钮，增加一个 XML 文件。文件如下：

```
<Advertisements>
   <Ad>
 <ImageUrl>d:\\asp\\bookExample\\p2.JPG</ImageUrl>
 <NavigateUrl>http://www.sohu.com</NavigateUrl>
 <AlternateText>search anything</AlternateText>
 <Impressions>10</Impressions>
 <Keyword>Topic1</Keyword>
```

```
<Caption>This is the caption for Ad#1</Caption>
</Ad>

<Ad>
<ImageUrl>d:\\asp\\bookExample\\baobao048.jpg</ImageUrl>
<NavigateUrl>http://www.sina.com</NavigateUrl>
<AlternateText>sina main site</AlternateText>
<Impressions>10</Impressions>
<Keyword>Topic2</Keyword>
<Caption>This is the caption for Ad#2</Caption>
</Ad>

</Advertisements>
```

(3) 在窗体中放置控件 AdRotator，其属性 Name=AdRotator1。

(4) 修改控件 AdRotator 的属性 AdvertisementFile,单击其后的按钮，出现选择 XML 文件对话框，在对话框中 URL(U)处填入 ads.xml。

(5) 运行，可以看到一幅图，鼠标移到图中，变为手形，单击可以转到搜狐网站。刷新，可以看到另一幅图。

XML 文件包含以下预定义的属性。只有 ImageUrl 属性是必需的:

- ImageURL 要显示的图像的 URL
- NavigateURL 单击 AdRotator 控件时定位到的页面的 URL。
- AlternateText 图像不可用时显示的文本（常为图片浏览工具提示）。
- Keyword 可用于筛选特定广告的广告类别。通过设置 AdRotator 的 KeywordFilter 属性以筛选出 XML 文件中特定类别的广告。
- Impression 指示广告的可能显示频率的数值。在 XML 文件中，所有 impression 值的总和不能超过 2,048,000,000 - 1

10.6 Calender 控件

Calendar Web 服务器控件在 Web 窗体页上显示一个传统的单月份日历(包含该月在内的 6 周)。用户可使用该日历查看和选择日期。Calendar Web 服务器控件最简单的用法如下：(C7-2A.aspx)

```
<html>
<head>
    <script language="C#" runat="server">
        void Date_Selected(object s, EventArgs e) {
            Label1.Text = "Selected date is: " + Calendar1.SelectedDate.ToShortDateString();
        }
    </script>
</head>
<body>
    <h3><font face="Verdana">Calendar Example</font></h3>
    <form runat=server>
        <asp:Calendar id=Calendar1 onselectionchanged="Date_Selected" runat="server" />
```

```
        <p>
        <asp:Label id=Label1 runat="server" />
    </form>
</body>
</html>
```

其中事件 onselectionchanged="Date_Selected"是用户改变选择日期时产生的事件。SelectionMode 属性设定 Calendar 控件中可选择的时间段，Day:可选择任一天；DayWeek:可选择任一天或一周；DayWeekMonth:可选择任一天、一周或一月；None:不能选择日期。C7-2A.aspx 网页显示了 SelectionMode 属性选择不同质的效果。

```
<html>
<head>
    <script language="C#" runat="server">
        void Page_Load(Object Sender, EventArgs e) {
            Calendar1.SelectionMode = (CalendarSelectionMode)lstSelMode.SelectedIndex;
            if (Calendar1.SelectionMode == CalendarSelectionMode.None)
                Calendar1.SelectedDates.Clear();
        }
        void Date_Selected(object s, EventArgs e) {
            switch (Calendar1.SelectedDates.Count) {
                case (0):    //None
                    Label1.Text = "No dates are currently selected";
                    break;
                case (1):    //Day
                    Label1.Text = "The selected date is " +
Calendar1.SelectedDate.ToShortDateString();
                    break;
                case (7):    //Week
                    Label1.Text = "The selection is a week beginning " +
Calendar1.SelectedDate.ToShortDateString();
                    break;
                default:     //Month
                    Label1.Text = "The selection is a month beginning " +
Calendar1.SelectedDate.ToShortDateString();
            break;
            }
        }
    </script>
</head>
<body>
    <h3><font face="Verdana">Date Selection Modes</font></h3>
    <p>
    <form runat=server>
        Choose a Selection Mode:
        <asp:DropDownList id="lstSelMode" runat=server
```

```
            AutoPostBack=true>
            <asp:ListItem Value="None" >None</asp:ListItem>
            <asp:ListItem Selected Value="Day" >Day</asp:ListItem>
            <asp:ListItem Value="DayWeek" >DayWeek</asp:ListItem>
            <asp:ListItem Value="DayWeekMonth" >DayWeekMonth</asp:ListItem>
        </asp:DropDownList>

        <p>
        <asp:Calendar id=Calendar1 runat="server"
            onselectionchanged="Date_Selected"
            DayNameFormat="FirstLetter"
            Font-Name="Arial" Font-Size="12px"
            Height="180px" Width="200px"
            SelectorStyle-BackColor="gainsboro"
            TodayDayStyle-BackColor="gainsboro"
            DayHeaderStyle-BackColor="gainsboro"
            OtherMonthDayStyle-ForeColor="gray"
            TitleStyle-BackColor="gray"
            TitleStyle-Font-Bold="True"
            TitleStyle-Font-Size="12px"
            SelectedDayStyle-BackColor="Navy"
            SelectedDayStyle-Font-Bold="True"
            />
        <p>
        <asp:Label id=Label1 runat="server" />
    </form>
</body>
</html>
```

1、显示和选择日期

可视日期：该日期确定日历中显示哪个月份。在日历中，用户可在不同的月份之间移动，从而在不影响当前日期的情况下更改可视日期。

选定的一个或多个日期：在该控件中用户可通过设置 SelectionMode 属性选择单个日、单个周或单个月份，但只能选择连续的日期。

可设置日历的属性以更改日历的颜色、尺寸、文本以及其他可视特性。默认情况下，该控件显示月中各天、周中各天的标头、带有月份名和年份的标题、用于选择月份中各天的链接及用于移动到下个月和上个月的链接。可以通过设置控制控件中不同部分的样式的属性，来自定义 Calendar 控件的外观。

SelectionMode 属性： 设定 Calendar 控件中可选择的时间段

Day:可选择任一天； DayWeek:可选择任一天或一周；

DayWeekMonth:可选择任一天、一周或一月

None:不能选择日期

SelectDate 属性实现选择日期：按所定义的外观样式显示运行时所选定的一天、一周或一月

控件的 DayRender 事件:当在 Calendar 控件中创建（显示）每个日期单元格时，均会引发 DayRender 事件。通过在 DayRender 事件的事件处理程序中提供代码，可以在创建日期单

元格时控制其内容和格式设置。事件处理程序接收一个 DayRenderEventArgs 类型的参数，它包含与此事件相关的数据。DayRenderEventArgs 属性：
Cell：获取表示 Calendar 控件单元格的 TableCell 对象。
Day：获取表示 Calendar 控件中日期的 CalendarDay。
例 dayreader.aspx 为 DayRender 事件编写处理程序，使所显示月份中日期的背景色为黄色。它还说明如何通过向单元格添加 System.Web.UI.LiteralControl 来自定义单元格的内容。

10.7 Visual Studio.Net 实现留言板

本例有两个窗口，主窗口负责输入留言，包括输入用户名，留言主题，留言内容，用三个编辑框，输入完毕后，单击提交按钮，将留言存入数据库。单击另一个查看留言按钮，可链接到另一个显示留言窗口。显示留言窗口包括一个 DataGraid 控件，用来显示所有的留言的用户名，主提，留言序号，及按钮列，单击相应按钮，显示当前记录的留言内容。单击返回主窗口按钮，返回主窗口。下边是具体步骤：

(1) 用 Access2000 建立数据库: LiuYanBan.mdb。建立 LiuYanTable 表，记录所有留言信息。包括字段 LiuYanID(留言编号)，自动编号类型，为主关键字。字段 LiuYanName(留言者姓名)，文本，字段大小 10，必填字段，默认值为空。字段 LiuYanTitle(留言标题)，文本，字段大小 30，必填字段，默认值为空。字段 LiuYanTime(留言时间)，时间类型。字段 LiuYanContent(留言内容)，备注字段，必填字段，默认值为空。增加若干数据。存数据库的路径为：D:\asp\ LiuYanBan.mdb，假设文件夹 asp 已设为 Web 网站目录。

(2) 创建一个 Web 应用程序框架，选择菜单命令建立一个新空白窗体。项目名为 LiuYanBan。

(3) 修改 WebForm 属性，单击属性 Style 后标题为...的按钮，打开样式生成器对话框，可以修改 WebForm 的各种风格。单击对话框左侧的各个选项：字体、背景、文本、位置、布局、边缘、列表、其他，可以按自己的爱好修改相应的内容，这里不作修改，全部采用默认值。

(4) 放工具箱的 4 个 Label 控件到窗体。修改属性 Text 分别为：留言板主窗体、用户名、留言主题、留言内容。

(5) 放工具箱的 3 个 TexbBox 控件到窗体。修改属性 Text 都为空，ID=TexbBox1 编辑框用来输入用户名，ID=TexbBox2 编辑框用来输入留言主题，ID=TexbBox3 编辑框用来输入留言内容，其属性 TextMode=MultiLine。由于此三项要求必须输入数据，因此应增加 3 个验证控件。

(6) 放工具箱的 Button 控件到窗体，Text=”提交留言”。

(7) 在窗体中放置控件 oleDbConnection，其属性 Name=oleDbConnection1。单击控件 oleDbConnection 属性 ConnectionString 的下拉列表的箭头，在列表中选择新建连接，打开数据连接属性对话框，选择提供程序页，选择 OLE DB 提供程序为 Microsoft Jet 4.0 OLE DB Provider 后，单击下一步按钮，选择数据库名称为 D:\asp\ LiuYanBan.mdb，用户名称为 Admin，空白密码，单击测试连接按钮，应出现测试连接成功对话框。按确定按钮退出。

(8) 在窗体中放置控件 oleDbCommand,其属性 Name= oleDbCommand1。单击控件 oleDbCommand 属性 Connection 的下拉列表的箭头，在列表中单击现有前的+后，选择已有的连接 oleDbConnection1。

(9) 为单击提交留言按钮事件(Click)函数增加语句(双击 Click 事件)：

```
private void Button1_Click(object sender, System.EventArgs e)
```

```
        {
            oleDbConnection1.Open();
//自动增加字段不必写入
            oleDbCommand1.CommandText="Insert Into
LiuYanTable(LiuYanName,LiuYanTitle,LiuYanTime,LiuYanContent) Values('" + TextBox1.Text + "','"
+ TextBox2.Text + "','" + DateTime.Now + "','" + TextBox3.Text + "')";

            oleDbCommand1.ExecuteNonQuery();
            oleDbConnection1.Close();
            TextBox1.Text="";
            TextBox2.Text="";
            TextBox3.Text="";
        }
```

(10)单击文件/添加新项(w)...菜单项，出现添加新项对话框，选择 Web 窗体，窗体名为：WebForm2.aspx，单击打开按钮，创建新窗体。

(11)在 WebForm1 放工具箱的 HyperLink 控件到窗体，Text="查看留言"，单击属性 NavigateUrl 后的按钮，出现选择 URL 对话框，选择 URL 类型为与根相关的，URL 编辑框添入/LiuYanBan/WebForm2.aspx。

(12)在 WebForm2 窗体中放置控件 oleDbConnection，其属性 Name=oleDbConnection1。单击控件 oleDbConnection 属性 ConnectionString 的下拉列表的箭头，在列表中选择前边建立的数据库连接。

(13)在 WebForm2 窗体中放置控件 oleDbDataAdapter，出现添加数据适配器向导对话框，单击下一步按钮，单击下拉列表的箭头，在列表中选择前边建立的数据库连接。单击下一步按钮。

(14)选择使用 SQL 语句单选按钮。单击下一步按钮。

(15)单击高级选项按钮，在高级 SQL 选项对话框中，所有多选按钮都不选。单击确定按钮。

(16)单击查询生成器按钮，在添加表对话框中，选中 LiuYanBan 数据库，单击添加按钮。再按关闭按钮，关闭添加表对话框。

(17)选中所有字段，按 LiuYanID 降序排列，单击确定按钮。

(18)单击确定按钮。单击完成按钮。

(19)单击 sqlDataAdapter1 选中它，单击菜单项数据/生成数据集…，打开生成数据集对话框，他选择默认值。按确定按钮退出。增加控件 dataSet，其属性 Name=dataSet1。

(20)在 WebForm2 窗体中放置控件 dataView，其属性 Name=dataView1。单击控件 dataView1 属性 Table 的下拉列表的箭头，在列表中单击现有前的+后，选择 dataSet1 中的 LiuYanTable。

(21)在窗体中放置控件 Label，其属性 Name=Label1。

(22)在窗体中放置控件 DataGrid，其属性 Name=DataGrid1。右击 DataGrid1，在弹出菜单中选择菜单项自动套用格式，在对话框中选用自己喜欢的格式。

(23)右击 DataGrid1，在弹出菜单中选择菜单项属性生成器，在 DataGrid 属性对话框中，选中左侧的选项：常规。设置数据源为：dataView1。选中显示页眉，显示页脚，允许排序。选中左侧的选项：列。不选中在运行时自动创建列。将字段：LiuYanName、LiuYanTitle、LiuYanTime 从左侧的列表框移到右侧的列表框，表示显示此三个字段。见页眉文本改成中文：留言者姓名、留言标题、留言时间。增加一个 Select 按钮，增加一个按钮列，页眉为：单击按钮查看留言。命令名为：ReadContent。选中左侧的选

项：分页。选中允许分页。

(24)为按钮列增加事件函数，DataGraid 所有按钮都产生事件：ItemCommand，根据命令名加以区分是哪一个按钮发的命令。事件函数如下：

```
private void DataGrid1_ItemCommand(object source,System.Web.UI.WebControls.
DataGridCommandEventArgs e)
{
    if (e.CommandName == "ReadContent")
    {
       Label1.Text=dataSet11.Tables["LiuYanTable"].Rows[e.Item.ItemIndex]["LiuYanContent"].
ToString();
    }
}
```

(25)为 Page_Load 事件函数增加语句：

```
        private void Page_Load(object sender, System.EventArgs e)
        {
            oleDbDataAdapter1.Fill(dataSet11);
            if(!Page.IsPostBack)
            {
                DataGrid1.CurrentPageIndex=0;
                DataGrid1.DataBind();
            }
            // 在此处放置用户代码以初始化页面
        }
```

(26)为 DataGraid1 的 DataGrid1_PageIndexChanged 事件函数增加语句：

```
private void DataGrid1_PageIndexChanged(object
source,System.Web.UI.WebControls.DataGridPageChangedEventArgs e)
        {
            DataGrid1.CurrentPageIndex=e.NewPageIndex;
            DataGrid1.DataBind();
        }
```

(27)在 WebForm2 放工具箱的 HyperLink 控件到窗体，Text=”输入留言”，单击属性 NavigateUrl 后的按钮，出现选择 URL 对话框，选择 URL 类型为与根相关的，URL 编辑框添入/LiuYanBan/WebForm1.aspx。

(28)运行，出现 WebForm1，可以输入一条留言，单击提交按钮，再单击超级链接查看留言，转到 WebForm2，单击查看留言按钮，可以在 Label1 处看到留言，单击超级链接输入留言，转到 WebForm1。

第十一章　ASP.NET 内建对象

ASP.NET 为保持浏览用户的数据和信息，内建了许多对象，包括 Application、Response、Request、cookie、Sessions、Cache 和 Server 等对象，以及它们的大量的方法。通过这些对象，可以提供网络开发必不可少的功能，例如当前目录的获得、在线人数、访问网站总人数、网上商店中的购物筐等等。

11.1 Request 对象

Request 对象主要有以下用途：第一用来来在不同网页之间传递数据，第二是 Web 服务器可以使用 Request 对象获取用户所使用的浏览器的信息，第三是 Web 服务器可以使用 Request 对象显示 Web 服务器的一些信息，最后，可以用 Request 对象获得 Cookie 信息。本节主要介绍前三种用途，后边有一节专门介绍 Cookie。

11.1.1 用 Request 对象获取另一个网页传递的数据

从一个网页链接到另一个网页时，可能需要传递一些数据到另一个网页。两个 Web 网页之间一般通过表单(From)传递，具体传递方法有两个：Post 和 Get。当数据传递到另一个网页时，另一个网页用 Request 对象的方法取出这些数据。见下例：(e11_1A.aspx)

```
<html>
<body>
  <form action=e11_1B.aspx  method=POST  runat=server>
    <asp:Label id=label1"  runat=server>用户名：</asp:Label>
    <asp:TextBox id="textBox1" Text="" runat=server></asp:TextBox>
    <asp:button text="提交"  runat=server/>
  </form>
</body>
</html>
```

其中action是用户单击此按钮后，响应用户程序网页的URL，这里是e11_1B.aspx。语句method=POST是数据用POST方法传到e11_1B.aspx，也可以是get方法。在e11_1B.aspx网页中，是不能用string s= textBox1.Text语句得到输入的内容的，因为textBox1是另一个网页的对象。必须用语句string s=Request.Form("textBox1")得到输入的内容。如果将属性method="POST"改为method="GET"，用语句string s=Request.QueryString("textBox1")得到输入的内容。下边是e11_1B.aspx网页完整文件：

```
<html>
  <script language="c#" runat=server>
    void Page_Load(Object src,EventArgs e)
    {//如用 GET 方法，修改为："用户名："+Request.QueryString("textBox1");
      Label1.Text="用户名："+Request.Form("textBox1");//用 POST 方法使用的语句
```

```
    }
  </script>
<body>
  <form runat=server>
    <asp:Label id="Label1" runat=server></asp:Label>
  </form>
</body>
</html>
```

如 Button 按钮改为 HyperLink 控件如何使用

11.1.2 用 Request 对象获取客户端浏览器的信息

不同浏览器或相同浏览器的不同版本支持不同的功能，Web 应用程序可能要根据不同的浏览器采取不同的措施，可用 HttpRequest.Browser 属性的 HttpBrowserCapabilities 对象获得用户使用的浏览器信息。见下例：

```
<html>
  <script language="c#" runat=server>
    void Page_Load(Object src,EventArgs e)
    {
      string s="浏览器的特性如下："+"<br>";
      HttpBrowserCapabilities bc=HttpRequest.Browser;
      S+="Type="+bc.Type+"<br>";
      S+="Name="+bc.Browser+"<br>";
      S+="Version="+bc.Version+"<br>";
      S+="Major Version="+bc.MajorVersion+"<br>";
      S+="Minor Version="+bc.MinorVersion+"<br>";
      S+="Platform="+bc.Platform+"<br>";
      S+="Is Beta="+bc.Beta+"<br>";
      S+="Is Crawler="+bc.Crawler +"<br>";
      S+="Is AOL="+bc.AOL +"<br>";
      S+="Is Win16="+bc.Win16+"<br>";
      S+="Is Win32="+bc.Win32+"<br>";
      S+="Supports Frames="+bc.Frames+"<br>";
      S+="Supports Tables="+bc.Tables+"<br>";
      S+="Supports Cookies="+bc.Cookies+"<br>";
      S+="Supports VB Script="+bc.VBScript+"<br>";
      S+="Supports Java Script="+bc.JavaScript+"<br>";
      S+="Supports Java Applets="+bc.JavaApplets +"<br>";
      S+="Supports ActiveX Controls="+bc.ActiveXControls+"<br>";
      S+="CDF="+bc.CDF+"<br>";
      Label1.Text=s;
    }
  </script>
<body>
```

```
  <form runat=server>
    <asp:Label id="Label1" runat=server></asp:Label>
  </form>
</body>
</html>
```

11.1.3 用 Request 对象获取服务器信息

```
<html>
  <script language="c#" runat=server>
    void Page_Load(Object src,EventArgs e)
    {
      string s="服务器的特性如下："+"<br>";
      foreach(string Name in Request.ServerVariables)
      {
      s+=Name+":"+Request.ServerVariables(Name)+"<br>"
      }
      Label1.Text=s
    }
  </script>
<body>
  <form runat=server>
    <asp:Label id="Label1" runat=server></asp:Label>
  </form>
</body>
</html>
```

11.2 Response 对象

与 Request 是获取客户端 HTTP 信息相反，Response 对象是用来控制发送给用户的信息，包括直接发送信息在浏览器中显示、重定向浏览器到另一个 URL 以及设置 cookie 的值。在 ASP.NET 中一般不用 Response 对象发送信息给浏览器，可以用其它方法重定向浏览器到另一个 URL，因此在 ASP.Net 中使用 Response 对象的机会越来越少了，这里只对 Response 对象做简单介绍，设置 cookie 方法在另一节介绍。

11.2.1 用 Response 对象发送信息在浏览器中显示

(1) 在浏览器中显示数据，例如：（在 ASP.Net 不建议这样使用。）

```
<%@ Page language="c#" %>
<html>
<body>
  <%
```

```
    Response.Write("<font Size=7>");
    Response.Write("Response 对象使用");
    Response.Write("</font");
    Response.Write("<br>");
    %>
</body>
</html>
```

(2) 显示一个文件

```
<%@ Page language="c#" %>
<html>
<body>
    <%
    System.IO.FileStream fs=new System.IO.FileStream("d:\\asp\\g1.txt",FileMode.Open);
    IntPtr FileHandle=fs.Handle;
    Response.WriteFile(FileHand,0,fs.Length);
    Fs.Close();
    %>
</body>
</html>
```

11.2.2 用 Response 对象重定向浏览器

用 Response 对象重定向浏览器到新浪网主页的例子如下：

```
<html>
<script language="c#" runat=server>
    void EnterBtn_Click(Object src,EventArgs e)
    {
        Response.Redirect("http://www.sina.com.cn");
    }
</script>
<body>
    <form runat=server>
    <asp:Label runat=server>单击按钮打开新浪网主页</asp:Label>
    <br>
    <asp:button text="打开新浪网" Onclick="EnterBtn_Click" runat=server/>
    </form>
</body>
</html>
```

这里实现的功能完全可以用 HyperLink 控件实现，请读者试一试。但是如果根据条件用语句实现转向其它网页，使用此语句还是必要的，例如，有些用户企图不经过登录直接访问其它网页，在其它网页的 Page_Load 方法中要进行判断，如果未登录，可用上述方法直接转向登录界面。

11.3 Cookie 对象

用户用浏览器访问一个网站，由于采用的 http 的特性，Web 服务器并不能知道是哪一个用户正在访问，但一些网站，希望能够知道访问者的一些信息，例如是不是第一次访问，访问者上次访问时是否有未做完的工作，这次是否为其继续工作提供方便等等。用浏览器访问一个网站，可以在此网站的网页之间跳转，当从第一个网页转到第二个网页时，第一个网页中建立的所有变量和对象都将不存在。有时希望在这些被访问的网页中建立联系，例如一个网上商店，访问者可能从不同的网页中选取不同的商品，那么用什么办法记录该访问者选取的商品，也就是一般所说的购物筐如何实现。用 Cookie 对象可以解决以上问题。

11.3.1 用 Cookie 对象记录访问的次数

```
<html>
  <script language="c#" runat=server>
    void Page_Load(Object src,EventArgs e)
    {
      if(!Page.IsPostBack)//如果用户单击刷新按钮，访问次数不加 1
      {
        int Num=1;
        HttpCookie myCookie=Request.Cookies["VistNum"];
        if(myCookie!=null)
        {
          myCookie.Value=myCookie.Values+1;
          Num=myCookie.Values;
        }
        else
        {
          myCookie=new HttpCookie("VistNum");
          myCookie.Values=1;
          Response.Cookie.Add(myCookie);
        }
        Label1.Text="您是第"+Convert.ToString(Num)+"次访问本站";
      }
    }
  </script>
<body>
  <form runat=server>
    <asp:Label id="label1" runat=server></asp:Label>
  </form>
</body>
</html>
```

当然，浏览器的 Cookies 必须设置为允许使用。

11.3.2 网上商店购物筐实现

网上商店网站一般有多个网页，用户可以浏览这些网页，从每个网页中选择商品，网上商店网站要记录这些要购买的商品，一般把这个功能叫做购物筐，下边的例子介绍购物筐的实现方法。例子中有两个网页，每个网页有一个 CheckBoxList 控件，可以选不同商品，每个网页都有两个按钮，一个按钮的标题是：把选中商品放入购物筐，另一个按钮的标题是：结算。

(1) 第一个网页文件 e11_3_2A.aspx 如下：

```
<html>
<script language="c#" runat=server>
  void button1_Click(Object src,EventArgs e)
  {
      string s;
      HttpCookie myCookie;
      for(int i=0;i<2;i++)
      {
        if(checkBoxList1.Items[i].Selected)
        {
          s=checkBoxList1.Items[i].Text;
          checkBoxList1.Checked=false;//已记录，清除所做选择
          myCookie=Request.Cookies[s];
          if(myCookie!=null)
          {
            myCookie.Value=myCookie.Values+1;
          }
          else
          {
            myCookie=new HttpCookie(s);
            myCookie.Values=1;
            Response.Cookie.Add(myCookie);
          }
        }
    }
  }
  void button2_Click(Object src,EventArgs e)
  {
    string s="您定购了如下商品：<br>";
    HttpCookieCollextion myCookie myCookieS=Request.Cookies;
    for(i=0;i<myCookieS.Length;i++)
    {
      s+=myCookieS[i].Name+"："+myCookieS[i].Value.ToString+"<br>"
    }
    label1.Text=s;
  }
```

```
</script>
<body>
  <form runat=server>
  <asp:CheckBoxList id=checkBoxList1 runat=server>
    <asp:ListItem Text="香蕉"/>
    <asp:ListItem Text="苹果"/>
  </asp:CheckBoxList>
  <br>
  <asp:button text="把选中商品放入购物筐" Onclick="button1_Click" runat=server/>
  <br>
  <asp:button text="结算" Onclick="button2_Click" runat=server/>
  <br>
  <asp:HyperLink NaviGateUrl="e11_3_2B.aspx" runat=server/>选择花卉</asp:HyperLink>
  <br>
  <asp:Label id="label1" Text="" runat=server></asp:Label>
  </form>
</body>
</html>
```

(2) 第二个网页文件 e11_3_2B.aspx 如下：

```
<html>
<script language="c#" runat=server>
  void button1_Click(Object src,EventArgs e)
  {
      string s;
      HttpCookie myCookie;
      for(int i=0;i<2;i++)
      {
        if(checkBoxList1.Items[i].Selected)
        {
          s=checkBoxList1.Items[i].Text;
          checkBoxList1.Checked=false;//已记录，清除所做选择
          myCookie=Request.Cookies[s];
          if(myCookie!=null)
          {
            myCookie.Value=myCookie.Values+1;
          }
          else
          {
            myCookie=new HttpCookie(s);
            myCookie.Values=1;
            Response.Cookie.Add(myCookie);
          }
        }
    }
```

```
    }
    void button2_Click(Object src,EventArgs e)
    {
      string s="您定购了如下商品：<br>";
      HttpCookieCollextion myCookie myCookieS=Request.Cookies;
      for(i=0;i<myCookieS.Length;i++)
      {
        s+=myCookieS[i].Name+"："+myCookieS[i].Value.ToString+"<br>"
      }
      label1.Text=s;
    }
</script>
<body>
    <form runat=server>
    <asp:CheckBoxList id=checkBoxList1 runat=server>
      <asp:ListItem Text="菊花"/>
      <asp:ListItem Text="茉莉"/>
    </asp:CheckBoxList>
    <br>
    <asp:button text="把选中商品放入购物筐" Onclick="button1_Click" runat=server/>
    <br>
    <asp:button text="结算" Onclick="button2_Click" runat=server/>
    <br>
    <asp:HyperLink NaviGateUrl="e11_3_2A.aspx" runat=server/>选择水果</asp:HyperLink>
    <br>
    <asp:Label id="label1" Text="" runat=server></asp:Label>
    </form>
</body>
</html>
```

(3) 两个个文件都存到宿主目录中，在浏览器中输入地址：http://Localhost/e11_3_2A.aspx，选中某种水果，转到第二个网页 e11_3_2B.aspx，选中某种花卉，单击结算按钮，应显示所选的所有商品。当然，本例只是说明问题，由许多不尽合理之处。读者可以采用数据库，用 DataGraid 控件商品，增加一列，由两个按钮，标题分别是：放到购物筐和从购物筐取出。还应时刻显示购物筐的内容。

Cookies 集合设置 cookie 的值。若指定的 cookie 不存在，则创建它。若存在，则设置新的值并且将旧值删去。

语法 Response.Cookies(cookie)[(key)|.attribute]=value

这里的 cookie 是指定 cookie 的名称。而如果指定了 key，则该 cookie 就是一个字典。Attribute 指定 cookie 自身的有关信息。Attribute 参数可以是下列之一 ：

Domain 若被指定，则 cookie 将被发送到对该域的请求中去。

Expires 指定 cookie 的过期日期。为了在会话结束后将 cookie 存储在客户端磁盘上，必须设置该日期。若此项属性的设置未超过当前日期，则在任务结束后 cookie 将到期。

HasKeys 指定 cookie 是否包含关键字。

Path 若被指定，则 cookie 将只发送到对该路径的请求中。如果未设置该属性，则使用应用程序的路径。

11.4 Application 对象

Application 对象生存期和 Web 应用程序生存期一样长，生存期从 Web 应用程序网页被访问开始，HttpApplication 类对象 Application 被自动创建，直到没有一个网页被访问时结束，Application 对象被自动撤销。因此 Application 对象中的变量也有相同生存期，并且变量可以被 Web 应用程序中的所有网页访问。因此，可以在 Application 对象中建立一些全局的公用变量，由于存储在 Application 对象中的数值可以被应用程序的所有网页读取，所以 Application 对象的属性也适合在应用程序的网页之间传递信息。Application 对象主要有以下用途：

- 存储记录在线人数或访问网站总人数的变量。
- 存储网站共用最新消息，供所有网页更新。
- 记录网站中个网页同一条广告被点击的次数或时间。
- 存储供所有网页使用的数据库数据。
- 不同用之间通讯，例如多用户聊天室，多用户游戏等

本节首先介绍 Application 对象的用法，然后介绍记录访问网站总人数的实现方法。

11.4.1 Application 对象属性

虽然 Application 对象没有内置的属性，但我们可以使用以下句法设置用户定义的属性也可称为集合：Application("属性/集合名称")=值，例如，Application("MyVar")="Hello"。用以下语句取出数据：string s= Application("MyVar")。

11.4.2 方法

Application 对象有两个方法，它们都是用于处理多个用户对存储在 Application 中的数据进行写入的的同步问题。由于存储在 Application 对象中的数值可以被应用程序的所有网页读取，因此一个用户在修改这个变量时，不允许其它用户修改，这两个方法就是解决这个问题的。

- L ock 方 法

  Lock 方法阻止其他客户修改存储在 Application 对象中的变量，以确保在同一时刻仅

有一个客户可修改和存取 Application 变量。如果用户没有明确调用 Unlock 方法，则服务器将在 .asp 文件结束或超时后即解除对 Application 对象的锁定。

- Unlock 方法

和 Lock 方法相反，Unlock 方法允许其他客户修改 Application 对象的属性。下例介绍一个计数器变量的使用方法。

```
Application.Lock;
Application["counter"]=(Int32)Application["counter"]+1;
Application.UnLock;
```

11.4.3 事件

- Application_OnStart 事件

第一个浏览器访问 Web 应用程序网页时，产生的事件。

- Application_OnEnd 事件

没有浏览器访问时 Web 应用程序网页时，产生的事件。

Application_OnStart 和 Application_OnEnd 事件的处理过程必须写在 global.asax 文件之中。

11.4.4 例子:显示访问网站总人数

(1) 建立一个主页文件 Default.aspx 如下：

```
<html>
<script language="c#" runat=server>
    void Page_Load(Object src,EventArgs e)
    {
        if(!Page.IsPostBack)//如果用户单击刷新按钮，计数器不加 1
        {
            int num;
            Application.Lock;
            Application["counter"]=(Int32)Application["counter"]+1;
            num=(Int)Application["counter"];
            Application.UnLock;
            label1.Text=Convert.ToString(num);
        }
    }
</script>
<body>
    <form runat=server>
    <asp:Label id="label1" Text="" runat=server></asp:Label>
    <br>
    <asp:HyperLink id="hLink1" NavigaterUrl="other.aspx" Target="_blank" runat=server>
    单击此处转到 e1.aspx，计数器不加 1。
    </asp:HyperLink >
```

```
    </form>
</body>
</html>
```
(2) 建立 other.aspx 网页文件如下：
```
<html>
<script language="c#" runat=server>
    void Page_Load(Object src,EventArgs e)
    {
            Int num=(Int)Application["counter"];
            label1.Text=Convert.ToString(num);
        }
    }
</script>
<body>
  <form runat=server>
  <asp:Label id="label1" Text="" runat=server></asp:Label>
  <br>
  <asp:HyperLink id="hLink1" NavigaterUrl="default.aspx" runat=server>
  单击此处转到 dault.aspx，计数器不加 1。
  </asp:HyperLink >
  </form>
</body>
</html>
```
(3) 建立 global.asax 文件如下：
```
<script language="c#" runat=server>
    void Application_OnStart(Object src,EventArgs e)
    {
        Application.Add("counter",0);
    }
</script>
```
(4) 三个文件都存到宿主目录中，在浏览器重输入 URL 地址：http://Localhost/，查看显示的计数器数值，单击刷新按钮，查看显示的计数器数值是否改变，转到 Other.aspx 网页，在转回 dault.aspx 网页，查看显示的计数器数值是否改变。关闭所有网页，在打开 default.aspx 网页，显示的计数器值从 0 开始，这是因为没有网页访问网站时，Application 对象被自动撤销。在打开新网页，产生 Application_OnStart 事件，将 counter 值为 0。为了解决此问题，可以建立一个文件，记录访问网站总人数，初值为 0，Application_OnStart 事件函数中，从文件取出已访问网站总人数，赋值给 counter，Application_OnEnd 事件函数中，将 counter 存到文件中。

(5) 用记事本创建文件 counter_File.txt，其中内容为字符 0。存文件到宿主目录中。

(6) 修改 global.asax 文件如下：
```
<script language="c#" runat=server>
    void Application_OnStart(Object src,EventArgs e)
    {//用 Server 对象对象得到 counter_File 文件绝对路径
      string s=Serve.MapPath(\counter_File.txt);
```

```
        Application.Add("counterFile",s);//保存供 Application_OnEnd 事件函数使用
        System.IO.FileStream fs=new System.IO.FileStream("s", FileMode.OpenOrCreate);
        System.IO.StreamReader r=new System.IO.StreamReader(fs);
        s=r.ReadLine();
        r.Close();
        Application.Add("counter",Convert.ToInt(s));
    }
    void Application_OnEnd(Object src,EventArgs e)
    {//此时 Server 对象已不存在，无法用 Server 对象得到 counter_File 文件绝对路径
        string s= (string)Application("counterFile");//取出保存的计数文件的全路径地址
        System.IO.FileStream fs=new System.IO.FileStream("s", FileMode.OpenOrCreate);
        System.IO.StreamWrite w=new System.IO.StreamWrite(fs);
        int num=(int)Application("counterFile");
        w.Write(num.ToString());
        w.Close();
    }
</script>
```

(7) 再一次访问 dault.aspx 网页，看是否已解决以上提出的问题。这里还有一个问题，如果用用如下 URL 访问网页：http://Localhost/Other.aspx，这样计数器就不能计数，解决的方法见 Session 对象一节。

11.5 Session 对象

前边提到，用浏览器访问一个网站，当在网站的网页之间跳转时，希望在这些被访问的网页中建立联系，例如一个网上商店的购物筐的实现，这些可以用 Cookie 实现。用 Session 对象也可以解决以上问题。

当浏览器开始访问网站的某网页时，Web 服务器将自动创建一个 Session 对象，在 Session 对象中可以建立一些变量，这个 Session 对象和 Session 对象中的变量只能被这个访问者使用，其它访问者不能使用。当用户在网站的网页之间跳转时，Session 对象和存储在 Session 对象中的变量不会被清除，这些变量始终存在。当浏览器离开本网站或超过一定时间和网站没有联系，Session 对象被撤销，同时存储在 Session 中的变量也不存在了。

在 ASP 中，Session 对象的功能本质上是用 Cookie 实现的，如果用户将浏览器上面的 Cookies 设置为禁用，那么 Session 就不能工作。但在 ASP.NET 中我们有解决方法，在 config.web 文件中，我们将＜sessionstate cookieless="false" /＞设置为 true 就可以了，也就说，不使用 Cookies 也可以使用 Session。

11.5.1 属性

- SessionIDSessionID 属性返回用户的会话标识。在创建会话时，服务器会为每一个会话生成一个单独的标识。会话标识以长整形数据类型返回。在很多情况下 SessionID 可以用于 WEB 页面注册统计。
- TimeOut Timeout 属性以分钟为单位为该应用程序的 Session 对象指定超时时限。如果

用户在该超时时限之内不刷新或请求网页，则该会话将终止。

11.5.2 方法

Session 对象仅有一个方法，就是 Abandon，Abandon 方法删除所有存储在 Session 对象中的对象并释放这些对象的源。如果您未明确地调用 Abandon 方法，一旦会话超时，服务器将删除这些对象。当服务器处理完当前页时，下面示例将释放会话状态。 < % Session.Abandon %>

11.5.3 事件

Session 对象有两个事件可用于在 Session 对象启动和释放是运行过程。

- Session_OnStart 事件在服务器创建新会话时发生，当用户第一次浏览网页时，发生 Session_OnStart 事件。服务器在执行请求的页之前先处理该脚本。Session_OnStart 事件是设置会话期变量的最佳时机，因为在访问任何页之前都会先设置它们。尽管在 Session_OnStart 事件包含 Redirect 或 End 方法调用的情况下 Session 对象仍会保持，然而服务器将停止处理 Global.asa 文件并触发 Session_OnStart 事件的文件中的脚本。为了确保用户在打开某个特定的 W eb 页 时始终启动一个会话，就可以在 S ession_OnStart 事 件中调用 R edirect 方 法。当用户进入应用程序时，服务器将为用户创建一个会话并处理 S ession_OnStart 事 件脚本。您可以将脚本包含在该事件中以便检查用户打开的页是不是启动页，如果不是，就指示用户调用 R esponse.Redirect 方 法启动网页。程序如下 :< SCRIPT RUNAT=Server Language=VBScript> Sub Session_OnStart startPage = "/MyApp/StartHere.asp" currentPage = Request.ServerVariables("SCRIPT_NAME") if strcomp(currentPage,startPage,1) then Response.Redirect(startPage) end if End Sub < /SCRIPT> 上述程序只能在支持 cookie 的浏览器中运行。因为不支持 cookie 的浏览器不能返回 SessionID cookie，所以，每当用户请求 Web 页时，服务器都会创建一个新会话。这样，对于每个请求服务器都将处理 Session_OnStart 脚本并将用户重定向到启动页中。
- Session_OnEnd 事件在会话被放弃或超时发生。如果用户在指定时间内没有请求或刷新应用程序中的任何页，会话将自动结束。这段时间的默认值是 2 0 分 钟。可以通过在 I nternet 服 务管理器中设置“应用程序选项”属性页中的“会话超时”属性改变应用程序的默认超时限制设置。应依据您的 W eb 应 用程序的要求和服务器的内存空间来设置此值。例如，如果您希望浏览您的 W eb 应 用程序的用户在每一页仅停留几分钟，就应该缩短会话的默认超时值。过长的会话超时值将导致打开的会话过多而耗尽您的服务器的内存资源。对于一个特定的会话，如果您想设置一个小于默认超时值的超时值，可以设置 S ession 对象的 T imeout 属 性。例如，下面这段脚本将超时值设置为 5 分 钟。 < % Session.Timeout = 5 %> 当然你也可以设置一个大于默认设置的超时值，Session.Timeout 属性决定超时值。你还可以通过 Session 对象的 Abandon 方法显式结束一个会话。例如，在表格中提供一个“退出”按钮，将按钮的 ACTION 参数设置为包含下列命令的 .asp 文件的 URL。 < % Session.Abandon %>
-

11.5.4 用 Session 对象实现网上商店购物筐

本例要求和用 Cookie 实现网上商店购物筐完全一样，只是用 Session 对象实现，具体代码如下：

(1) 第一个网页文件 e11_5_4A.aspx 如下：

```
<html>
<script language="c#" runat=server>
  void button1_Click(Object src,EventArgs e)
  {
       string s;
       for(int i=0;i<2;i++)
       {
         if(checkBoxList1.Items[i].Selected)
         {
           s=checkBoxList1.Items[i].Text;
           checkBoxList1.Checked=false;//已记录，清除所做选择
           if(Session[s]!=null)
           {
              Session[s]=Session[s]+1;
           }
           else
           {
              Session[s]=1;
           }
         }
    }
  }
  void button2_Click(Object src,EventArgs e)
  {
    string s="您定购了如下商品：<br>";
    string keyName;
    for(i=0;i<Session.Count;i++)
    {
      keyName=Session.Keys[i];
      s+=keyName+"："+Session[keyName].ToString+"<br>"
    }
    label1.Text=s;
  }
</script>
<body>
  <form runat=server>
  <asp:CheckBoxList id=checkBoxList1 runat=server>
    <asp:ListItem Text="香蕉"/>
```

```
    <asp:ListItem Text="苹果"/>
  </asp:CheckBoxList>
  <br>
  <asp:button text="把选中商品放入购物筐" Onclick="button1_Click" runat=server/>
  <br>
  <asp:button text="结算" Onclick="button2_Click" runat=server/>
  <br>
  <asp:HyperLink NaviGateUrl="e11_5_4B.aspx" runat=server/>选择花卉</asp:HyperLink>
  <br>
  <asp:Label id="label1" Text="" runat=server></asp:Label>
  </form>
</body>
</html>
```

(2) 第二个网页文件 e11_5_4B.aspx 如下：

```
<html>
<script language="c#" runat=server>
  void button1_Click(Object src,EventArgs e)
  {
      string s;
      for(int i=0;i<2;i++)
      {
        if(checkBoxList1.Items[i].Selected)
        {
          s=checkBoxList1.Items[i].Text;
          checkBoxList1.Checked=false;//已记录，清除所做选择
          if(Session[s]!=null)
          {
            Session[s]=Session[s]+1;
          }
          else
          {
            Session[s]=1;
          }
        }
    }
  }
  void button2_Click(Object src,EventArgs e)
  {
    string s="您定购了如下商品：<br>";
    string keyName;
    for(i=0;i<Session.Count;i++)
    {
      keyName=Session.Keys[i];
      s+=keyName+"："+Session[keyName].ToString+"<br>"
```

```
        }
        label1.Text=s;
    }
</script>
<body>
    <form runat=server>
    <asp:CheckBoxList id=checkBoxList1 runat=server>
        <asp:ListItem Text="菊花"/>
        <asp:ListItem Text="茉莉"/>
    </asp:CheckBoxList>
    <br>
    <asp:button text="把选中商品放入购物筐" Onclick="button1_Click" runat=server/>
    <br>
    <asp:button text="结算" Onclick="button2_Click" runat=server/>
    <br>
    <asp:HyperLink NaviGateUrl="e11_5_4A.aspx" runat=server/>选择水果</asp:HyperLink>
    <br>
    <asp:Label id="label1" Text="" runat=server></asp:Label>
    </form>
</body>
</html>
```

(3) 两个个文件都存到宿主目录中，在浏览器中输入地址：http://Localhost/e11_3_2A.aspx，选中某种水果，转到第二个网页 e11_3_2B.aspx，选中某种花卉，单击结算按钮，应显示所选的所有商品。象前边所说，本例不尽合理，读者可以采用数据库，用 DataGraid 控件商品，增加一列，由两个按钮，标题分别是：放到购物筐和从购物筐取出。还应时刻显示购物筐的内容。

11.6 Server 对象

Server 对象提供对服务器上的资源进行访问的方法和属性，主要包括：得到服务器的计算机名称，设置脚本程序的失效时间，将 HTML 的特殊标记转变为 ASCII 字符，得到文件的真实路径等等，本节将逐一介绍这些方法。

11.6.1 属性 MachineName 和 ScriptTimeout

(1) 属性 MachineName

该属性用来获取当前运行 Web 应用程序的 Web 服务器的计算机名称，使用方法如下：string s=Server.MachineName;这个计算机名称可以用如下办法查到：打开”控制面板”，选中”系统”中的”计算机名”，应和用 Server 对象的属性 MachineName 获得计算机名称一致。

(2) 属性 ScriptTimeout

Web 应用程序由于运行在计算机网络中，由于网络的原因，一些程序可能无法完成，一直在等待，这将极大消耗 Web 服务器的资源，为了避免这种情况，可以设置程序运行的

最长时间，即设置属性 ScriptTimeout，在脚本程序运行超过属性 ScriptTimeout 指定时间之后即作超时处理，也就停止程序运行。如以下代码指定服务器处理脚本程序在 100 秒后超时：Server.ScriptTimeout=100，其默认值为 90 秒。

11.6.2 HtmlEncode 方法

HTML 标记语言中，有些 ASCII 字符被作为标记，例如字符串：
中的<和>都是标记，如需要显示这些字符，必须作特殊处理，例如为了在浏览器中正确显示如下字符串：”
是换行标记”，字符串必须写为如下形式：

```
<asp:Label id="label1" Text="%3cbr%3c 是换行标记" runat=server></asp:Label>;
```

也可以用 Server 对象的属性 HtmlEncode 方法，用法如下：

```
<asp:Label id="label1" runat=server>Server.HtmlEncode(”<br>是换行标记”)</asp:Label>;
```

11.6.3 URLEncode 方法

URL 是 Uniform Resource Location(统一资源定位器)的简称，URL 用来定位一个网页的。在 URL 中，有些 ASCII 字符具有特殊含义，必须做特殊处理。例如 http://www.sina.com/中的字符/，用 Server 对象 URLEncode 方法处理，

```
string s=”http://www.sina.com/”;
```

11.6.4 MapPath 方法

网页中网页文件的路径一般是以宿主目录为根目录，不同的系统中，宿主目录所在的实际目录并不相同，而且网页也可能在虚拟目录中。因此网页文件的路径并不是网页文件的实际路径。而在用 File 类处理文件时，则要求文件的地址必须是实际的全路径，Server 对象的 MapPath 方法提供这两种路径的转换方法，例如，f1.aspx 文件存在宿主目录下的 Test 目录下，用 Server 对象得到 f1.aspx 文件绝对路径方法如下：

```
string s=Serve.MapPath(\Test\f1.aspx);//这里\表示以宿主目录
```

也可以用如下语句：

```
string s=Serve.MapPath(Test\f1.aspx);//表示单前网页所在的目录的子目录 Test
```

习题

(1) 如何实现记录访问网站的在线人数。(提示：增加一个 Application 对象变量作为计数器，Application_Start 事件函数中计数器为 0，Session_Start 事件函数中计数器加 1，Session_End 事件函数中计数器减 1，每个网页的 Page_Load 事件函数中用 Label 控件显示计数器值。)
(2) 用 Application 对象建立一个 2 人聊天室。如果是多人聊天室，又如何实现。
(3) 用户不经过主页，直接访问网站的某网页，将不能时访问者总数加 1，如何防止。
(4) 将书中的例子用 Visual Studio.Net 实现。

第十二章　可扩展标记语言

12.1 HTML 及其缺点

Internet 提供了全球范围的网络互连与通信功能，Web 技术的发展更是一日千里，其丰富的信息资源给人们的学习和生活带来了极大的便利。特别是应运而生的 HTML（超文本置标语言），以简单易学、灵活通用的特性，使人们发布、检索、交流信息都变得非常简单，从而使 Web 成了最大的环球信息资源库。然而，电子商务、电子出版、远程教育等基于 Web 的新兴领域的全面兴起使得传统的 Web 资源更加复杂化、多样化，数据量的日趋庞大对网络的传输能力也提出更高的要求，人们对 Web 服务功能的需求也达到更高的标准。而传统的 HTML 由于自身特点的限制，不能满足这些要求。HTML 主要有如下不足：

- HTML 的标记都是预先定义的，用户不能自定义有意义的标记，可扩展性差。
- HTML 的显示方式内嵌在数据中，这样在创建文本时，要同时考虑输出格式，如果因为需求不同而需要对同样的内容进行不同风格的显示时，要从头创建一个全新的文档，重复工作量很大。不能对数据按照不同的需求进行多样化显示等个性化服务。
- HTML 缺乏对数据结构的描述，对于应用程序理解文档内容、抽取语义信息都有诸多不便。不能进行智能化的语义搜索。不能对不同平台、不同格式的数据源进行数据集成和数据转化等。
- HTML 语言不能描述矢量图形、数学公式、化学符号等特殊对象。

12.2 SGML(标准通用置标语言)

SGML(Standard Generalized Markup Language)是一种通用的文档结构描述置标语言，为语法置标提供了异常强大的工具，同时具有极好的扩展性，因此在数据分类和索引中非常有用。但 SGML 复杂度太高，不适合网络的日常应用，加上开发成本高、不被主流浏览器所支持等原因，使得 SGML 在 Web 上的推广受到阻碍。

12.3 XML(可扩展置标语言)

XML(eXtensible Markup Language)是由 W3C 于 1998 年 2 月发布的一种标准。它同样是 SGML 的一个简化子集，它将 SGML 的丰富功能与 HTML 的易用性结合到 Web 的应用中，以一种开放的自我描述方式定义了数据结构，在描述数据内容的同时能突出对结构的描述，从而体现出数据之间的关系。XML 的优点如下：

- XML 简单易用，功能强大。
- XML 允许各个组织、个人建立适合自己需要的标记集合，并且这些标记可以用通用的工具显示。例如定义数学、化学、音乐等专用标记。
- XML 的最大优点在于它的数据存储格式不受显示格式的制约。一般来说，一篇文档包括三个要素：数据、结构以及显示方式。XML 把文档的三要素独立开来，分

别处理。首先把显示格式从数据内容中独立出来，保存在样式表文件（Style Sheet）中，这样如果需要改变文档的显示方式，只要修改样式表文件就行了。XML 的自我描述性质能够很好地表现许多复杂的数据关系，使得基于 XML 的应用程序可以在 XML 文件中准确高效地搜索相关的数据内容，忽略其他不相关部分。

- XML 还有其他许多优点，比如它有利于不同系统之间的信息交流，完全可以充当网际语言，并有希望成为数据和文档交换的标准机制。

由于以上优点，XML 必将在商务的自动化处理，信息发布，智能化的 Web 应用程序和数据集成等领域被广泛被使用。

12.4 XML 的文档格式

首先介绍 XML 文档内容的基本单元——元素，它的语法格式如下：

〈标签〉文本内容〈/标签〉

元素是由起始标签、元素内容和结束标签组成。用户把要描述的数据对象放在起始标签和结束标签之间。例如：

<姓名>王平</姓名>

无论文本内容有多长或者多么复杂，XML 元素中还可以再嵌套别的元素，这样使相关信息构成等级结构。下面的例子中，在<学生>的元素中包括了所有学生的信息，每个学生都由<学生>元素来描述，而<学生>元素中又嵌套了<编号>、<姓名>、<性别>和<年龄>元素。完整 XML 文件 student.xml 内容如下，例 1：

```
<?xml version="1.0" encoding="GB2312"?>
<?xml-stylesheet type="text/xsl" href="student1.xsl"?>
<学生>
    <编号>001</编号>
    <姓名>张三</姓名>
    <性别>男</性别>
    <年龄>20</年龄>
</学生>
```

除了元素，XML 文档中能出现的有效对象是：声明指令、注释、根元素、子元素和属性。

- 声明指令

  声明指令给 XML 解析器提供信息，使其能够正确解释文档内容，它的起始标识是“<?”，结束标识是“?>”。常见的 XML 声明就是一个处理指令：

  <?xml version="1.0" encoding="GB2312"?>

  该处理指令指明 XML 使用的版本号和文档的编码方式是"GB2312"。又如：

  <?xml-stylesheet type="text/xsl" href="student1.xsl"?>

  使用 student1.xsl 样式表文件显示本 XML 文档。

- 注释

  注释是 XML 文件中用作解释的字符数据，XML 处理器不对它们进行任何处理。注释是用“<!--”和“ --> ”引起来的，可以出现在 XML 元素间的任何地方，但是不可以嵌套：

  <!--这是一个注释-->

- 根元素和子元素

  如果一个元素从文件头的序言部分之后开始一直到文件尾，包含了文件中所有的数据信息，我们称之为根元素。XML 元素是可以嵌套的，那么被嵌套在内的元素称

为子元素。在前面的例子中，<编号>就是<学生>的子元素。

- 属性

  属性给元素提供进一步的说明信息，它必须出现在起始标签中。属性以名称/取值对出现，属性名不能重复，名称与取值之间用等号“=”分隔，并用引号把取值引起来。例如：

  <工资 currency=“US$”> 25000 </工资>

  上例中的属性说明了薪水的货币单位是美元。

- XML 文档的基本结构

  XML 文档的基本结构由序言部分和一个根元素组成。序言包括了 XML 声明和 DTD（或者是 XML Schema），DTD（Document Type Define，文档定义类型）和 XML Schema 都是用来描述 XML 文档结构的，也就是描述元素和属性是如何联系在一起的。例如，在例 1 的文档前面加上如下的序言部分，就构成了一个完整的 XML 文档：

  <?xml version="1.0" encoding="GB2312"?>

  <?xml-stylesheet type="text/xsl" href="student1.xsl"?>

  <!DOCTYPE employees SYSTEM“employees.dtd”>

  一个 XML 文档中有且仅有一个根元素，其他所有的元素都是它的子元素，在例 1 中，<学生>就是根元素。

- 格式良好的”（Well-Formed）XML 文档

  一个 XML 文档首先应当是“格式良好的”（Well-Formed），该规定的正式定义位于：http://www.w3.org/TR/REC-xml。“格式良好的”XML 文档除了要满足根元素唯一的特性之外，还包括：

  (1) 起始标签和结束标签应当匹配：结束标签是必不可少的；

  (2) 大小写应一致：XML 对字母的大小写是敏感的，<employee>和<Employee>是完全不同的两个标签，所以结束标签在匹配时一定要注意大小写一致；

  (3) 元素应当正确嵌套：子元素应当完全包括在父辈元素中，下面的例子就是嵌套错误：

  <A>

  <B>

  </A>

  </B>

  正确的嵌套方式如下：

  <A>

  <B>

  </B>

  </A>

  (4) 属性必须包括在引号中；元素中的属性是不允许重复的。

12.5 用 XSL 文件显示 XML 文档

由于 XML 文档只是定义数据的结构，并不包含显示的格式。如要按指定格式显示这些数据，还要使用 CSS 文件或 XSL 文件定义显示格式。这里使用三个 XSL 文件按不同显示格式显示同一个 XML 文件。首先定义一个 student1.xsl 文件显示上边 XML 文档，文件如下：

<?xml version="1.0" encoding="GB2312"?>

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <xsl:for-each select="学生">
            <xsl:value-of select="编号"/>，
            <xsl:value-of select="姓名"/>，
            <xsl:value-of select="性别"/>，
            <xsl:value-of select="年龄"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

将 student1.xsl 文件和文件 student1.xml 存到同一文件夹，双击 student1.xml 文件，打开 IE5.0，显示效果如下：

001， 张三， 男， 20

可以定义不同的 xsl 文件，以不同的显示方式显示 student1.xml 文件。例如 student2.xsl 文件如下：

```
<?xml version="1.0" encoding="GB2312"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <html>
            <head>
                <meta http-equiv="Content-Type" content="text/html; charset=gb2312"/>
                <title>演示 2</title>
            </head>
            <body>
                <xsl:for-each select="学生">
                    <table border="1" cellpadding="0" cellspacing="0" style="border-collapse: collapse" bordercolor="#111111" width="100%" id="AutoNumber1">
                        <tr>
                            <td width="50%">编号</td>
                            <td width="50%">
                                <xsl:value-of select="编号"/>
                            </td>
                        </tr>
                        <tr>
                            <td width="50%">姓名</td>
                            <td width="50%">
                                <xsl:value-of select="姓名"/>
                            </td>
                        </tr>
                        <tr>
                            <td width="50%">性别</td>
                            <td width="50%">
                                <xsl:value-of select="性别"/>
                            </td>
```

```
                    </tr>
                    <tr>
                        <td width="50%">年龄</td>
                        <td width="50%">
                            <xsl:value-of select="年龄"/>
                        </td>
                    </tr>
                </table>
            </xsl:for-each>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

例如用 student3.xsl 以不同的显示方式显示 student1.xml 文件。文件如下：

```
<?xml version="1.0" encoding="GB2312"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
        <html>
            <head>
                <meta http-equiv="Content-Type" content="text/html; charset=gb2312"/>
                <title>演示 3</title>
            </head>
            <body>
                <p align="center">学生信息</p>
                <xsl:for-each select="学生">
                    <form method="POST" action="--WEBBOT-SELF--">
                        <table border="0" cellpadding="0" cellspacing="0"
style="border-collapse: collapse" bordercolor="#111111" width="100%" id="AutoNumber1">
                            <tr>
                                <td width="45%" align="right">编号：</td>
                                <td width="5%" align="center">  </td>
                                <td width="50%">
                                    <input type="text" name="编号" size="20">
                                        <xsl:attribute name="value">
                                            <xsl:value-of select="编号"/>
                                        </xsl:attribute>
                                    </input>
                                </td>
                            </tr>
                            <tr>
                                <td width="45%" align="right">姓名：</td>
                                <td width="5%" align="center">  </td>
                                <td width="50%">
                                    <input type="text" name="姓名" size="20">
```

```
                                    <xsl:attribute name="value">
                                        <xsl:value-of select="姓名"/>
                                    </xsl:attribute>
                                </input>
                            </td>
                        </tr>
                        <tr>
                            <td width="45%" align="right">性别：</td>
                            <td width="5%" align="center">  </td>
                            <td width="50%">
                                <input type="text" name="性别" size="20">
                                    <xsl:attribute name="value">
                                        <xsl:value-of select="性别"/>
                                    </xsl:attribute>
                                </input>
                            </td>
                        </tr>
                        <tr>
                            <td width="45%" align="right">年龄：</td>
                            <td width="5%" align="center">  </td>
                            <td width="50%">
                                <input type="text" name="年龄" size="20">
                                    <xsl:attribute name="value">
                                        <xsl:value-of select="年龄"/>
                                    </xsl:attribute>
                                </input>
                            </td>
                        </tr>
                    </table>
                    <p align="center">
                        <input type="submit" value="Submit" name="B1"/>
                        <input type="reset" value="Reset" name="B2"/>
                    </p>
                </form>
            </xsl:for-each>
            <p align="center">  </p>
        </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

读者可以试一下。

12.6 .NET 对 XML 的支持

首先，创建一个 XML 文档，文件名为 MyXMLFile.xml，内容如下：

```
<?xml version="1.0" encoding="GB2312" ?>
<!--这是一个注释-->
<bookstore>
    <book  出版社="电子工业出版社">
        <书名>SQL 实用全书</书名>
        <作者>Rafe Colburn</作者>
        <出版日期>2001 年 6 月</出版日期>
        <价格>34.00</价格>
    </book>
    <book  出版社="清华大学出版社">
        <书名>C#高级编程</书名>
        <作者>Simon Robinson</作者>
        <出版日期>2002 年 6 月</出版日期>
        <价格>128.00</价格>
    </book>
    <book  出版社="人民邮电出版社">
        <书名>ASP.NET 从入门到精通</书名>
        <作者>Chris Payne</作者>
        <出版日期>2002 年 1 月</出版日期>
        <价格>41.00</价格>
    </book>
    <book  出版社="中国青年出版社">
        <书名>精通 C#与 ASP.NET 程序设计</书名>
        <作者>孙三才</作者>
        <出版日期>2003 年 6 月</出版日期>
        <价格>39.00</价格>
    </book>
    <book  出版社="电子工业出版社">
        <书名>ASP.NET 实用全书</书名>
        <作者>张三</作者>
        <出版日期>2004 年 6 月</出版日期>
        <价格>55.00</价格>
    </book>
</bookstore>
```

请读者用 IE 浏览器(5.0 以上版本)浏览 MyXMLFile.xml 文件，单击标记前的减号(或加号)，看一下效果。

网页文件 c8-1-1A.aspx 用来读出每本书的书名、作者、出版日期、价格等数据。

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<html>
```

```
<title>读 XML 文件</title>
<script runat=server>
public void Page_Load(Object sender, EventArgs e)
{
    string FileNameString="d:\\asp\\bookExample\\MyXMLFile.xml";
    XmlTextReader dr= new XmlTextReader(FileNameString);
    while(dr.Read())
      if(dr.NodeType==XmlNodeType.Text)
          ListBox1.Items.Add(dr.Value);
}
</script>
<body>
<h2>读 XML 文件</h2>
<form runat=server>
    <asp:ListBox id="ListBox1" runat="server"/>
</form>
</body>
</html>
```

当用 XmlTextReader 读(dr.Read())Xml 文档时，每次读出一个节点的数据。

一个 Xml 文档的元素，可以分为两大类，第一类是文本，第二类是标记。文本是 Xml 文档的数据，在两个标记之间的文本被称为一个文本节点，例如，<书名>SQL 实用全书</书名>中的”SQL 实用全书”是一个文本节点。这个节点的类型是：Xml.XmlNodeType.Text。

第二类 Xml 文档的元素是标记，它可以分为以下几大类：注释标记、声明标记、开始标记，结束标记，每类都被称为一个 Xml 文档的标记节点，例如，<!--这是一个注释-->是注释标记，注释标记的节点类型为：Xml.XmlNodeType.Comment，注释的内容为 dr.Value。<?xml version="1.0" encoding="GB2312" ?>是声明标记，其中包括两个声明：xml version="1.0"和 encoding="GB2312"，等号前内容的被称为声明的名称(dr.Name)，等号后内容的被称为声明的值(dr.Value)。声明标记的节点类型为：Xml.XmlNodeType.XmlDeclartion。<book 出版社="电子工业出版社">是开始标记，book 被称为标记名称(dr.Name)，出版社被称为属性(dr.AttributeName)，"电子工业出版社"被称为属性的值(Value)。开始标记的节点类型为：Xml.XmlNodeType.Element。</book>是结束标记，book 被称为标记名称(dr.Name)。结束标记的节点类型为：Xml.XmlNodeType.EndElement。

本网页的 Page_Load 方法中，用 dr.Read()读 Xml 文档，每次读出一个节点的数据，用语句 if(dr.NodeType==XmlNodeType.Text)判断是否是文本节点，如果是文本节点，则把文本内容加到 ListBox1。如果希望只显示书名，则判断语句可以改为：if(dr.NodeType==XmlNodeType.Text && dr.Name==”书名”)。

网页文件 c8-1-1B.aspx 用来读出标记 book 的属性，具体内容如下：

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<html>
<title>读 XML 文件</title>
<script runat=server>
public void Page_Load(Object sender, EventArgs e)
{
```

```
    string FileNameString="d:\\asp\\bookExample\\MyXMLFile.xml";
    XmlTextReader dr= new XmlTextReader(FileNameString);
    while(dr.Read())
      if(dr.NodeType==XmlNodeType.Element)
          for(int i=0;i<dr.AttributeCount;i++)
            ListBox1.Items.Add(dr.GetAttribute(i));
}
</script>
<body>
<h2>读 XML 文件</h2>
<form runat=server>
    <asp:ListBox id="ListBox1" runat="server"/>
</form>
</body>
</html>
```

如果，显示注释，改为下列语句：

```
public void Page_Load(Object sender, EventArgs e)
{
    string FileNameString="d:\\asp\\bookExample\\MyXMLFile.xml";
    XmlTextReader dr= new XmlTextReader(FileNameString);
    while(dr.Read())
      if(dr.NodeType==XmlNodeType.Comment)
          ListBox1.Items.Add(dr.Value);
}
```

如果，显示声明，改为下列语句：(见 c8-1-1c.aspx)

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<html>
<title>读 XML 文件</title>
<script runat=server>
public void Page_Load(Object sender, EventArgs e)
{
    string FileNameString="d:\\asp\\bookExample\\MyXMLFile.xml";
    XmlTextReader dr= new XmlTextReader(FileNameString);
    while(dr.Read())
      if(dr.NodeType==XmlNodeType.XmlDeclaration)
          ListBox1.Items.Add(dr.Name+" "+dr.Value);
}
</script>
<body>
<h2>读 XML 文件</h2>
<form runat=server>
    <asp:ListBox id="ListBox1" runat="server"/>
</form>
```

```
</body>
</html>
```

下例用 DataGrid 控件显示 MyXMLFile.xml，(见 c8-1-1D.aspx)

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Data" %>
<html>
<title>读 XML 文件</title>

<script runat=server>
public void Page_Load(Object sender, EventArgs e)
{
    string FileNameString="d:\\asp\\bookExample\\MyXMLFile.xml";
    DataSet ds = new DataSet();
    ds.ReadXml(FileNameString);
    DataGrid1.DataSource=ds;
    DataGrid1.DataMember="book";
    DataGrid1.DataBind();
}
</script>


<body>
<h2>读 XML 文件</h2>

<form runat=server>
    <asp:DataGrid id="DataGrid1" runat="server"/>
</form>

</body>
</html>
```

12.7 ADO.NET 和 XML

仔细察看 MyXMLFile.xml 文件，它和数据库的表有对应关系，标记<bookstore>之间的内容可以看作一个数据库的表，标记<book>之间的内容可以看作一个数据库的表的一个记录，标记<书名>、<作者>、<出版日期>、<价格>可以看作一个数据库的表的字段，这些标记之间的文本可以看作这些字段的数据。我们知道，一个字段还有一些其他属性，例如，字段的数据类型，为了表示这些属性，可以使用 DTD（Document Type Define，文档定义类型）和 XML Schema 来描述 XML 文档的数据结构，也就是描述元素和属性是如何联系在一起的。微软的.NET 系统支持用 XML Schema 来描述 XML 文档的数据结构，下例介绍如何使用 XML Schema，见文件 C8-1-1F.aspx。

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
```

```
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.IO" %>

<html>
<title>DataGrid</title>

<script runat=server>
DataSet ds;
public void Page_Load(Object sender, EventArgs e)
{
      string txtConn="DATABASE=Northwind;SERVER=localhost;UID=sa;PWD=;";
      string txtCommand="SELECT employeeid, firstname, lastname FROM Employees";
      SqlConnection conn = new SqlConnection(txtConn);
      SqlDataAdapter da = new SqlDataAdapter(txtCommand, conn);

      ds = new DataSet();
      da.Fill(ds, "MyTable");

      // Display the data
      grid.DataSource = ds.Tables["MyTable"];
      grid.DataBind();
      grid.Visible = true;
}

public void SvaeXmlWithSchema(Object sender, EventArgs e)
{
          ds.WriteXml(Server.MapPath("XmlFile1.xml"), XmlWriteMode.WriteSchema);
}

public void SvaeXmlNoSchema(Object sender, EventArgs e)
{
          StreamWriter sw = new StreamWriter(Server.MapPath("XmlFile2.xml"));
          sw.Write(ds.GetXml());
          sw.Close();
}
</script>


<body bgcolor="ivory" style="font-family:arial;font-size:9pt">

<!-- ASP.NET topbar -->
<h2>将数据库表存为带 XML 架构和不带 XML 架构 XML 文件</h2>

<form runat=server>
```

```
        <asp:linkbutton runat="server" id="SvaeXml1" text="将数据库表存为带 XML 架构 XML 文件" onclick="SvaeXmlWithSchema" />
        <br>
        <asp:linkbutton runat="server" id="SvaeXml2" text="将数据库表存为不带 XML 架构 XML 文件" onclick="SvaeXmlNoSchema" />

        <asp:DataGrid id="grid" runat="server" visible="false"
        AutoGenerateColumns="true"
        CssClass="Shadow" BackColor="white"
        CellPadding="2" CellSpacing="2" GridLines="none"
        BorderStyle="solid" BorderColor="black" BorderWidth="1"
        font-size="x-small" font-names="verdana">

        <AlternatingItemStyle BackColor="palegoldenrod" />
        <ItemStyle BackColor="beige" />
        <HeaderStyle ForeColor="white" BackColor="brown" Font-Bold="true" />
          </asp:DataGrid>

    </form>

    </body>
    </html>
```

单击两个按钮，可以创建带 XML 架构和不带 XML 架构 XML 文件，文件名为"XmlFile1.xml"和"XmlFile1.xml"。用浏览器察看这两个 XML 文件，可以看到它们的区别。如创建了有架构的 XML 文件，可以修改该文件，例如，修改字段类型。用网页文件 C8-1-1G 可以重新打开带 XML 架构或不带 XML 架构 XML 文件。

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>
<%@ Import Namespace="System.Data" %>
<html>
<title>读 XML 文件</title>

<script runat=server>
public void Page_Load(Object sender, EventArgs e)
{
    string FileNameString="d:\\asp\\bookExample\\XmlFile.xml";
    DataSet ds = new DataSet();
    ds.ReadXml(FileNameString);
    DataGrid1.DataSource=ds;
    DataGrid1.DataMember="MyTable";
    DataGrid1.DataBind();
}
</script>
```

```
<body>
<h2>读带 XML 架构和不带 XML 架构 XML 文件</h2>

<form runat=server>
    <asp:DataGrid id="DataGrid1" runat="server"/>
</form>

</body>
</html>
```

12.8 使用 Visual Studio.Net 建立和显示 XML 文档

(1) 创建一个 Web 应用程序框架，项目名为 UseXml。
(2) 在窗体中放置控件 DataGrid，其属性 Name=DataGrid1。
(3) 放工具箱的 2 个 Button 控件到窗体，修改属性 Text 分别为：存为带 XML 架构的 XML 文件，读带 XML 架构的 XML 文件。
(4) 新建一个 XML 文件。单击菜单项"项目/添加新项"，弹出标题为添加新项的窗口，在窗口中选中 XML 文件，文件名为 MyXMLFile.xml，单击打开按钮，增加一个 XML 文件
(5) 在文件添加如下内容：

```
<?xml version="1.0" encoding="GB2312" ?>
<!--这是一个注释-->
<bookstore>
    <book 出版社="电子工业出版社">
        <书名>SQL 实用全书</书名>
        <作者>Rafe Colburn</作者>
        <出版日期>2001 年 6 月</出版日期>
        <价格>34.00</价格>
    </book>
    <book 出版社="清华大学出版社">
        <书名>C#高级编程</书名>
        <作者>Simon Robinson</作者>
        <出版日期>2002 年 6 月</出版日期>
        <价格>128.00</价格>
    </book>
    <book 出版社="人民邮电出版社">
        <书名>ASP.NET 从入门到精通</书名>
        <作者>Chris Payne</作者>
        <出版日期>2002 年 1 月</出版日期>
        <价格>41.00</价格>
    </book>
    <book 出版社="中国青年出版社">
        <书名>精通 C#与 ASP.NET 程序设计</书名>
        <作者>孙三才</作者>
        <出版日期>2003 年 6 月</出版日期>
```

```
        <价格>39.00</价格>
    </book>
    <book 出版社="电子工业出版社">
        <书名>ASP.NET 实用全书</书名>
        <作者>张三</作者>
        <出版日期>2004 年 6 月</出版日期>
        <价格>55.00</价格>
    </book>
</bookstore>
```

(6) 单击 MyXMLFile.xml 窗口下的数据，可以看到用表格显示的 XML 文件。

(7) 为 Page_Load 事件函数增加语句：

```
private void Page_Load(object sender, System.EventArgs e)
{
    string FileNameString="MyXMLFile.xml";
    DataSet ds = new DataSet();
    ds.ReadXml(Server.MapPath(FileNameString));
    DataGrid1.DataSource=ds;
    DataGrid1.DataMember="book";
    DataGrid1.DataBind();
    // 在此处放置用户代码以初始化页面
}
```

(8) 运行，可以看到用表格显示的 XML 文件。

(9) 打开 MyXMLFile.xml 文件，单击菜单项"XML/创建架构"，将创建 MyXMLFile.xsd 文件，打开此文件，可以修改每个字段的属性。

(10)为单击存为带 XML 架构的 XML 文件按钮事件(Click)函数增加语句(双击 Click 事件)：

(11)为单击读带 XML 架构的 XML 文件按钮事件(Click)函数增加语句(双击 Click 事件)：

第十三章　Web 服务

Micosoft.Net 平台架构中的分布式系统主要包括两部分：用 ASP.Net 技术构建服务器端动态网页，以及 Web 服务(Web Service 或 XML Web Service)。前边章节已详细介绍了构建服务器端动态网页的方法，本节将介绍 Web 服务的基本概念和构建方法。

13.1 Web 服务的概念和用途

Web 中无论是静态网页还是动态网页，数据都被嵌入到网页中，网页的服务对象都是人，用户可以很容易阅读这些网页。但如果一个程序使用这种方式获得数据，会是十分困难的，程序必须从网页中把数据分离，才能加以利用。而用一个程序在 Web 中获得数据有时又是十分必要的，例如：一个气象台总站希望通过 Internet 网获得各个基层气象台的各种资料，在网上以统一的网页对外发布。气象台总站希望各个基层气象台提供一个 Internet 网的服务，能根据总站的要求，自动提供相应的资料。类似的例子很多，例如一个很大的单位的总部和下属单位之间信息系统的整合，一个综合网站希望自动获得其它网站提供的信息等等。这种需求实际上就是 Web 服务。

为实现这种功能有很多困难，各个基层气象台使用的系统可能完全不同，即使使用相同的操作系统，也可能使用不同数据库系统，数据库中定义的字段可能不同，数据库应用程序可能使用不同的语言编制，即使这些完全相同，还可能数据的表示方式不相同，数据格式，数据的位数等等。为解决这些问题，已提出了许多方案，例如：微软的分布式控件对象模型(DCOM)、对象管理组织（OMG）的公用对象请求代理程序体系结构(CORBA)、SUN 公司的远程方法调用(RMI)等等，但这些方法都不能很好的解决以上问题。

Micosoft.Net 的 Web 服务为实现这种功能提供了完整的解决方案。Web 服务使用 Http 协议在 Internet 网上传输数据和消息，用 XML 扩展标记语言描述数据，用 SOAP 表示消息，SOAP 是一个简单的、重量轻的基于 XML 的协议，用于交换 Web 上的结构化的和模式化的信息。用 Micosoft.Net 的 Web 服务实现气象台总站所需功能的大概思路是这样的，每个基层气象台在自己的系统中提供一个 Internet 网远程调用函数，该函数用 Http 协议接受用 SOAP 表示的调用，并把函数的返回值用 XML 扩展标记语言描述，用 SOAP 表示后，用 Http 协议返回给调用者。气象台总站只要使用 Http 和 SOAP 协议逐一调用这些 Web 远程函数，就可以获得各个基层气象台的资料了。由于这些协议都是被广泛接受的协议，能被不同的系统所接受，也就解决了以上所提出的问题。

有以上叙述可知，Web 服务不追求代码的可移植性，而是提供一个可行的解决方案来增强数据和系统的互操作性。有许多 Web 服务的定义，比较简单又比较容易理解的描述是：Web 服务是一个可通过 Http、SOAP 和 XML 协议进行访问的 Web 远程函数库。

刚才讨论的问题只是 Web 服务的几个应用，还有许多其它用途，例如：。

- **应用程序集成**

你可以使用 Web 服务以一种集成的方式整合表面上看上去完全不同的现有应用程序。例如许多公司的每个部门都有定制的软件，产生一系列有用但是孤立的数据和业务逻辑。为了管理上的方便，非常有必要把这些应用程序功能集合到一起。利用 Web 服务，就有可能把现有的应用程序中的数据和功能以 Web 服务方式提供给其它部门。然后可以创建一个集

成的应用程序，增强各部门之间的互操作性。

- **代码复用**

在软件开发行业，大部分开发者都依赖代码复用。过去开发者们为了利用他人已经实现了的代码，或者将代码段复制到自己的代码中，做一些改动以适应自己得需要，或者在服务器或个人计算机上安装一个控件库，让应用程序来访问这个库。这将使得代码有很多个版本，而这些版本间可能只有细微差别，却分散在各个地方。当代码最初的开发者决定对代码更新一下或者改正一下错误，要把这些改变告诉所有使用这些代码的开发者的时候，将是非常困难的。如果我们把代码放在一个中心位置存储，让所有人都访问这儿，这不是很好吗？这样原创者可以在做了一些增补或者修正之后，能够立即提供给所有使用它的人。用 Web 服务可以实现以上设想，远程调用 Web 服务中的方法，就象调用本地函数一样方便。

- **工作流程解决方案**

有些工作是非常复杂的，例如，货物的运输，可能要使用多种交通工具，火车、汽车、轮船等，商业上的一笔交易，都是一个非常复杂的流程，流程的每一个环节都由不同部门的不同的程序进行控制，如何建立这些控制程序之间的联系，是十分重要的。使用 Web 服务是一个很好的解决方案。通过 Web 服务，使各个流程控制程序建立联系，完全实现自动控制和管理。

- **新的销售方式**

现在软件的销售方式一般是用户把软件买回去，安装在自己的计算机中。有了 Web 服务，就可以提供软件的服务，按次收费。

- **由 Web 服务组成的自动化系统**

不远的将来，信息家电将要联接到 Internet 网上，PDA、手机，甚至各种嵌入式设备也要上网，这些设备和其它设备之间通过 Web 服务建立联系也是一种可行的方案。

13.2 建立 Web 服务

Web 服务仍采用客户/服务器模式(Cient/Server)。本节介绍在服务器端应做的工作，包括建立供客户端调用的 Web 服务方法，以及为了客户端使用 Web 服务方法，提供给客户端描述该 Web 服务的 WSDL 文档。

13.2.1 用记事本建立 Web 服务

建立一个 Web 服务文件和建立一个普通网页文件的步骤基本一样，下边是一个最简单的 Web 服务文件，其它程序访问其中的 Web 服务方法时，将返回参数 a 和 b 的和，具体程序代码如下：

```
<%@ WebService Language="C#" Class="MyClass"%>
using System;
using System.Web.Services;
public class MyClass:WebService
{    [WebMethod]
     public int MyWebMethod (int a,int b)
     {    return a+b;
     }
     //其它 WebMethod
```

}

在文件中，第一行的语句表示这是一个 Web 服务文件，使用 C#语言，Web 服务的类名是 MyClass。由于建立的 Web 服务类必须以 WebService 类为基类，所以必须引入命名空间 System.Web.Services，这个 Web 服务类必须是一个公有类。可供其它程序访问的方法叫 Web 服务方法，在其头部必须增加关键字[WebMethod]，表示这个方法是一个 Web 服务方法，这个方法必须是一个公有方法。

建立文件后，以 asmx 为扩展名存盘，存到网站的宿主目录中或其任意子目录中。使用 URL 定位这个 Web 服务文件。现在使用浏览器检验这个 Web 服务，如果把 Web 服务文件以 MyAdd.asmx 存到网站的宿主目录中，在浏览器中 URL 地址栏中输入如下地址：http://localhost/MyAdd.asmx，浏览器中显示如下：

点击 MyWebMethod，浏览器中显示如下：

在编辑框中输入两个加数分别为 10 和 20，然后点击 invote 按钮，在浏览器上显示如下内容，这是用 XML 标记表示的调用 Web 服务方法 MyWebMethod 返回的结果。

```
<?xml  version="1.0"  encoding="utf-8" ?>
 <string  xmlns="http://tempuri.org/">30</string>
```

13.2.2 用 Visual Studio.Net 建立 Web 服务

如果使用 Visual Studio.Net 建立这个 Web 服务文件，具体步骤如下：

(1) 打开 vs.net，新建项目(asp.net web 服务)，在位置中键入 http://localhost/webserver，其中 webserver 就是项目的名字。单击确定按钮，创建项目。

(2) 打开 Service1.asmx.cx 文件如下：

```
using  System;
using  System.Collections;
using  System.ComponentModel;
using  System.Data;
using  System.Diagnostics;
using  System.Web;
using  System.Web.Services;
namespace  webserver
{
///<summary>
///Service1 的摘要说明。
///</summary>
//(1)
public  class  Service1:System.Web.Services.WebService
{
    public  Service1()
    {
    //CODEGEN：该调用是 ASP.NET  Web 服务设计器所必需的
        InitializeComponent();
    }
```

```
    #region Component Designer generated code
    //Web 服务设计器所必需的
    private IContainer components = null;
    ///  <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()
    {
    }
    ///<summary>
    ///清理所有正在使用的资源。
    ///</summary>
    protected override void Dispose(bool disposing)
    {
        if(disposing&&components!=null)
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }
    #endregion
    //WEB 服务示例
    //HelloWorld()示例服务返回字符串 Hello  World
    //若要生成，请取消注释下列行，然后保存并生成项目
    //若要测试此 Web 服务，请按 F5 键
    //[WebMethod]
    //public string HelloWorld()
    //{
    //  return "Hello World";
    //}
  }
}
```

(3) 下面在//(1)处加入[WebService(Namespace="http://localhost/webserver/")]，这是因为 SOAP 是基于 http 协议上的，客户端无法知道 webservice 位于那个服务器上。在实际应用中，比如 http://www.ourfly.com 上放置这个 webservice，则 Namespace 改为 http://www.ourfly.com/webserver。

(4) 下面给这个 webservice 添加一个方法。微软帮我们写好了一个如下，以被注解掉。

```
//[WebMethod]
//public string HelloWorld()
//{
//return "Hello World";
//}
```

添加一个自己的方法。方法名称叫 show

```
[WebMethod]
public  string  show(string  yourname)
{
  return  “http://www.ourfly.com”+”欢迎”+yourname;
}
```

(5) 现在可以测试这个 Web 服务，按 F5 运行，点击 show，输入你的名字，然后点击 invote 按钮，在浏览器上显示如下内容，这是用 XML 标记表示的调用 Web 服务方法 Show 返回的结果。

```
<?xml  version="1.0"  encoding="utf-8"  ?>
<string  xmlns="http://tempuri.org/">http://www.ourfly.com 欢迎 yyg</string>
```

(6) 打开 bin 目录，Vs.net 已经将 proxy 做好了.webserver.dll。

(7) 请注意，这里运行只是一种测试，实际上应在其它计算机上生成一个调用此 Web 服务的程序，可以是 Windows 应用程序，也可以是控制台程序，或者是 ASP.Net 程序，即可以是 Micosoft.Net 系统程序，也可以是其它系统程序，例如 Java 程序，Linux 程序等等，下节将介绍这方面的知识。

13.2.3 服务描述语言(WSDL)

WSDL(Web Services Description Language)中文名称为 Web 服务描述语言。

Web 服务提供了一种服务，允许 Internet 上的计算机使用 http 和 SOAP 协议远程调用 Web 服务方法。大家都知道，为了使用一个函数，首先要看一下函数的使用说明。Web 服务方法也存在同样的问题，特别是 SOAP 协议，它采用 XML 标记语言描述 Web 服务中传递的消息，而 XML 标记语言是可以定义自己的标记的，但 SOAP 并没有提供一种通用的 XML 标记供 Web 服务使用，不同的 Web 服务中 SOAP 的 XML 标记定义可能不同。因此，为了使不同系统调用其它系统中的 Web 服务，必须对调用 Web 服务的方法及 Web 服务返回的数据的格式做详细说明即服务描述，而且这种描述也应采用被广泛接受的协议。

WSDL 就是 Web 服务描述语言。WSDL 是基于 XML 的，用 WSDL 生成一个 XML 文档，可以提供关于 Web 服务的操作信息，例如，抽象形式的服务接口信息、数据传输的具体访问协议和格式、供客户端使用该 Web 服务的细节等等。服务描述是一个使用 WSDL 语言的 XML 语法编写的 XML 文档，定义了 Web 服务能理解的 Web 服务消息格式。服务描述起一个协定的作用，用来定义一个 Web 服务的行为并且指示潜在的客户如何与之交互。由于在 micosoft.Net 中提供了一些工具，可以自动生成 WSDL 文档，这里就不介绍 WSDL 了，可以通过下边方法看到 micosoft.Net 自动生成的 WSDL 文档，例如查看上节生成的 Web 服务，在浏览器中 URL 地址中输入 http://localhost/MyAdd.asmx？WSDL，浏览器中显示该 Web 服务 WSDL 文档。

13.3 基于.Net 的 Web 服务客户端程序

Web 服务客户端程序是用来调用服务器端的 Web 服务方法，前边使用浏览器调用 Web 服务方法，只能算做一种测试，通过这种测试，可以验证 Web 服务方法的正确性，发现错误。作为客户端程序，无论在何处，采用那种操作系统，希望只要知道 Web 服务的所在网址，就可以调用其相关 Web 服务方法。Web 服务客户端程序一般应在 Web 网上的另一台计算机中，单做实验或学习，也可以和 Web 服务在同一台计算机中。本节介绍如何实现基于.Net 的 Web

服务客户端程序。

13.3.1 Web 服务客户端程序代理类

Web 服务客户端程序是用 http 和 SOAP 协议来调用远程的 Web 服务方法，因此，Web 服务客户端程序必须把程序的调用及其参数转化为 SOAP 协议，传送到 Web 服务。但这个工作比较繁琐，程序员希望采用象普通编程语言调用一个方法那样调用 Web 方法。.NET Framework 的 SDK 提供了一个程序 WSDL.EXE，可以自动为 Web 服务客户端程序生成一个代理程序，该代理程序的功能是，Web 服务客户端程序用一般程序语言那样调用 Web 服务方法，代理程序负责转换为 SOAP 协议，发送到 Web 服务方法，由代理程序负责获得 Web 服务方法返回的数据，由于这些数据也用 SOPA 协议表示，也要由代理程序转换为转换为一般程序语言能够理解的形式，传送给 Web 服务客户端程序。下边介绍生成代理程序的具体方法。WSDL.EXE 必须在控制台界面下使用，使用的格式如下：

WSDL /l:C# /OUT:Hello.cs /protocol:soap http://LocalHost/Hello.asmx?WSDL

其中，/l 参数指定编制 Web 服务客户端程序使用的语言，可以是 vb、C#和 Jscript，默认值为 C#；/OUT 参数指定生成的代理类文件的路径和文件名，默认值和 Web 服务 ASMX 文件同名，扩展名为参数/l 指定的语言的扩展名；参数/protocol 指定调用 Web 服务方法使用的协议，可以是 HTTP-GET、HTTP-POST 和 SOAP 协议；http://后边是 Web 服务 ASMX 文件的 URL。WSDL 运行的结果是生成一个 Web 服务客户端程序代理类的源程序。有了源程序，还要编译源程序生成 dll 文件，格式如下：csc /t:librrary hello.cs。把生成的 hello.dll 文件存到 Web 服务客户端程序项目所在目录的子目录 bin 下，这个代理类就可以被项目的其它成代码使用了。

13.3.2 HTTP-GET、HTTP-POST 和 SOAP 协议

当构造一个 XML Web 服务时，它自动支持客户端使用 SOAP、HTTP-GET 和 HTTP-POST 协议通讯。HTTP-GET 和 HTTP-POST 支持使用 URL 编码的变量名/变量值对来传送消息，支持这两个协议的数据类型没有支持 SOAP 协议的数据类型丰富。SOAP 是一个简单的、重量轻的基于 XML 的协议，用于交换 Web 上的结构化的和模式化的信息。SOAP 的总体设计目标是使它保持尽可能的简单，并且提供最少的功能。这个协议定义了一个不包含应用程序或传输语义的消息框架。因此，这个协议是模块化的并且非常利于扩展。在 SOAP 中，使用 XML 把数据传送到 XML Web 服务或从 XML Web 服务取回消息，你可以使用支持丰富的数据类型集。

更多 SOAP 规格的信息，请看 W3C Web 站点（http://www.w3.org/TR/soap）。

13.3.3 使用代理类的 Web 服务客户端程序

(1) 控制台应用程序

```
using System;
class Welcome;
{
  static void Main()
```

```
  {
    string s;
    int x,y,z;
    Console.WriteLine("Please enter first number:");
    s=Console.ReadLine();
    x=Convert.ToInt(s);
    Console.WriteLine("Please enter second number:");
    s=Console.ReadLine();
    y=Convert.ToInt(s);
    Hollo h1=new Hollo();//代理类对象
    z=h1.hello(x,y);//调用Web服务方法
    Console.WriteLine("sum:={0}",z);
  }
}
```

(2) Windows 应用程序

(3) ASP.Net 应用程序

13.3.4 Visual Studio.Net 建立 Web 服务客户端程序

使用 Visual Studio.Net 很容易建立 Web 服务客户端程序，这个客户端程序不必一定和 Web 服务在同一台计算机中，可以在任意一台 Internet 网中的计算机中。下边是具体步骤：

(1) 打开 Visual Studio.Net，新建 windows 应用程序项目，命名为 AddServiceClient，在窗体中增加一个按钮用来调用 Web 服务的 Web 方法，三个文本框，两个用来输入两个加数，另一个用来显示调用 Web 服务的 Web 方法后返回的结果。

(2) 建立 Web 服务客户端程序一般要建立一个代理。选择菜单项"项目"|"添加 Web 引用"，在弹出的对话框中的地址栏中输入 Web 服务的 URL，例如 Web 服务所在的计算机的 IP 地址是 202.206.96.20,Web 服务的文件 Service1.asmx 在网站宿主目录下的子目录 webserver 中，地址为：http://202.206.96.20/webserver/Service1.asmx。按回车键，出现添加 Web 引用对话框，如图：单击添加引用按钮，在解决方案资源管理器中，可以看到一个新的引用，以及从 Web 服务端发到客户端的 DISCO 和 WSDL 文档。在解决方案资源管理器中，还可以看到新创建的类，这个类就是 Web 服务客户端程序的代理程序，该类的用途是把 Web 服务客户端程序调用 Web 服务方法转换为 SOAP 格式。

(3) 为按钮增加事件函数如下：

(4) 天出的对话框中再加入一个 system.web.webservices 的引用，在列表中有。在 form1.cs 里，加入

using　System.Web.Services;

using　webserver;

然后在

private　System.Windows.Forms.Button　button1;

```
private  System.Windows.Forms.TextBox  textBox1;
```
后面，插入
```
private  webserver.service1  Client
```
建立一个 service1 的实例。双击按钮，代码如下：
```
private  void  button1_Click(object  sender,  System.EventArgs  e)
{
Client  =new  Service1();
string  name;
name=Client.show("龙卷风.NET");
textBox1.Text=name;
}
```
按 F5，运行项目，点击按钮，文本框中显示
http://www.ourfly.com 欢迎龙卷风.NET

2． Asp.NET web 窗口的测试
方法与上面的一模一样，添加引用，建立 service1 的实例
在此不在细说。
3.在 VB 中测试
这个就要相对来说复杂一些
首先在 vb 中建立一个”标准 EXE”的项目。添加引用：Microsoft Soap Type library。
注意：如果没有安装 Microsoft Soap Toolkit,是没有这个类型库的。
可以在 http://www.ourfly.com 中下载。
添加一个 text
```
Private  Sub  Form_Load()
        Text1.Text  =  add()
End  Sub

Public  Function  Add()  As  String
Dim  objSoapClient  As  New  SoapClient
        objSoapClient.ClientProperty("ServerHTTPRequest")  =  True
Call        objSoapClient.mssoapinit("http://localhost/webserver/service1.asmx?WSDL",
"Service1",  "Service1Soap")
```
这句也可以
```
objSoapClient.mssoapinit("http://localhost/webserver/service1.asmx?WSDL")

        Add  =  objSoapClient.Show("龙卷风.NET")
End  Function
```

13.4 建立 Web 服务客户端程序一般方法

13.5 发布和发现 Web 服务

完成 Web 服务开发，如何发布该 Web 服务，通知客户使用，程序开发者如何发现并定位所需功能的 Web 服务，是这节要解决的问题。

13.5.1 Web 服务目录

和使用因特网上任何其他的资源一样，如果没有某些查找方法的话，是不可能够找到一个特定的 Web 服务的。Web 服务目录提供了一个网址，例如：http://uddi.microsoft.org/，可以让 Web 服务供应者在其上发布他们提供的 Web 服务的信息。这样的目录甚至可以是 Web 服务本身，可以编程访问并且提供搜索结果来响应 Web 服务客户端的查询。使用一个 Web 服务目录来定位一个提供 Web 服务的 URL，这是非常必要的。

UDDI（统一描述发现和集成规范）规格定义了一个标准方法来发布和发现 Web 服务的信息，也就是通过 UDDI 发现指定 Web 服务的服务描述，该描述是一个使用 WSDL 语言的 XML 语法编写的 XML 文档。与 UDDI 关联的 XML 模式定义了四个信息类型，能让开发者使用一个发布的 Web 服务。这些是：商业信息、服务信息、绑定信息和其他用于服务的规范的信息。

作为 UDDI 项目的核心控件，UDDI Business Registry（业务登记）允许 Web 服务开发者发布其 Web 服务的信息。Web 服务使用者可以使用 UDDI Business Registry 来定位发现 Web 服务描述文件。更多信息，请看 UDDI Web 站点（http://uddi.microsoft.com）。

13.5.2 Web 服务发现

程序设计者可以通过以下步骤发现所需的 Web 服务：

(1) 首先，访问 Web 服务目录网址，例如 http://uddi.microsoft.org/，查找所需 Web 服务，将返回一个所需 Web 服务 URL。
(2) 按照返回 URL，访问这个网址，例如：http:// 返回 Web 服务 URL/default.discro。.disco 文件，是包含连接到其他描述 XML Web 服务的资源的 XML 文件，能够编程发现一个 XML Web 服务。disco 是一个包含与其它发现文档、XSD 模式和服务描述连接的 XML 文档。换句话说，使用 ASP.NET 创建的 XML Web 服务自动地有提供一个产生发现文档的能力。
(3) 使用 Web 服务的 WSDL 建立一个 Web 服务客户端程序代理类。
(4) 建立 Web 服务客户端程序，使用代理类访问 Web 服务方法。

Web 服务发现是使用 Web 服务描述语言 WSDL 定位或发现一个或多个描述特定的 XML Web 服务的文件的操作。它让 XML Web 服务客户端得知一个 XML Web 服务是否存在并且到哪里找到这个 XML Web 服务的描述文件。
一个发布的.disco 文件，是包含连接到其他描述 XML Web 服务的资源的 XML 文件，能够编程发现一个 XML Web 服务。