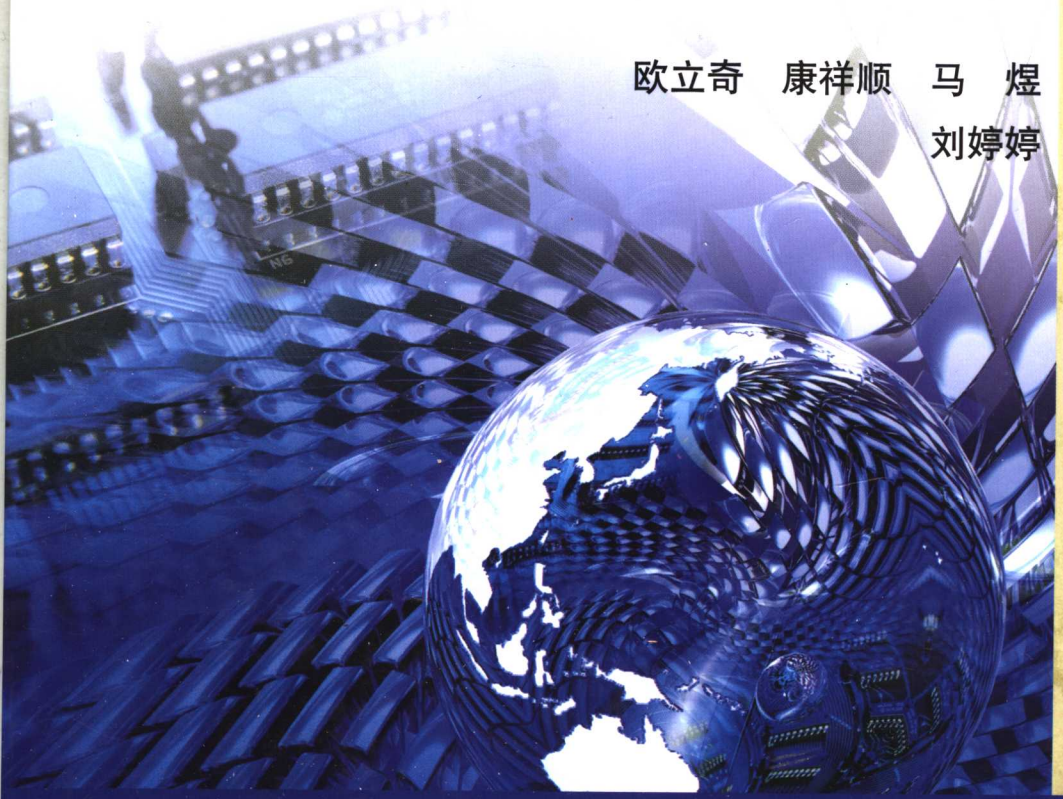
商业开发代码库系列

Visual C#.NET 案例开发集锦

欧立奇　康祥顺　马　煜　编著
刘婷婷　审校



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

商业开发代码库系列

Visual C#.NET案例开发集锦

欧立奇　康祥顺　马　煜　编著
刘婷婷　审校

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

内 容 简 介

本书主要通过具体的实例，介绍如何运用Visual C#.NET编程工具开发实际的应用程序，从基本应用到高级处理都有介绍，包括基础设计、图像处理、多媒体应用、系统文件处理、数据库基本处理、网络处理、网络与数据库高级应用、综合实例共8个章节。每个案例都严格按照读者的阅读习惯进行编排，由经验丰富的项目开发人员亲手编写，大部分案例都在项目开发中经过了实践的检验。本书将是指导你进入实战型程序设计师领域的一盏明灯。

本书适用于大中专院校学生、程序设计人员和C#编程爱好者。

图书在版编目（CIP）数据

Visual C#.NET案例开发集锦/欧立奇等编著.—北京：电子工业出版社，2005.10
（商业开发代码库系列）
ISBN 7-121-01722-9

Ⅰ. V… Ⅱ. 欧… Ⅲ. C语言—程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2005）第099779号

责任编辑：朱 巍
特约编辑：陈 虹 相里闵鹤
印 刷：北京天竺颖华印刷厂
出版发行：电子工业出版社
北京市海淀区万寿路173信箱 邮编：100036
北京市海淀区翠微东里甲2号 邮编：100036
经 销：各地新华书店
开 本：787×1092 1/16 印张：32.375 字数：810千字
印 次：2005年10月第1次印刷
定 价：47.00元（含光碟1张）

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换，若书店售缺，请与本社发行部联系。联系电话：010-68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

前　言

你需要阅读本书吗

微软推出的第三代网络平台.NET，激发了新的程序开发方式。为求重复使用、易维护，在新的架构中，.NET强调语言归语言、系统归系统，并在平台上推出新的.NET Framework，通过一致的Common Language Runtime执行应用程序。

现今对于.NET平台，各自看法不同。有些人说.NET是微软的下一代Visual Studio开发环境，有些人说它是一个新的程序语言（C#），还有些人说它是以XML和SOAP为基础的资料交换与传递信息的机制。其实，上述三者都是.NET想扮演的角色，而且还不止于此。

C#是一种最新的、面向对象的编程语言。它使得程序员可以快速地编写各种基于Microsoft.NET平台的应用程序，Microsoft.NET提供了一系列的工具和服务来最大程度地开发利用计算与通信领域。

正是C#面向对象的卓越设计，使它成为构建各类组件的理想之选——无论是高级的商业对象还是系统级的应用程序。使用简单的C#语言结构，这些组件可以方便地转化为XML 网络服务，从而使它们可以由任何语言在任何操作系统上通过Internet进行调用。

最重要的是，C#使得C++程序员可以高效地开发程序，而绝不损失C/C++原有的强大功能。因为这种继承关系，C#与C/C++具有极大的相似性，熟悉类似语言的开发者可以很快地转向C#。

本书涵盖的内容及编排格式

本书共8章，每章都是由既相互独立，又有一定逻辑关系的实例组成。每一个实例都分为5个部分编写。

- **案例运行效果与操作**：让读者对本案例有一个直观的印象，以便进一步了解如何解决这个问题。
- **制作要点**：让读者明白，要解决这个问题，需要用到哪些知识点。很多情况下，一个案例介绍完毕后，读者虽然可以把程序执行通过，但并不知道程序的内部机理和知识涵盖。本书作者力图让读者知其然更要知其所以然，达到举一反三的目的。
- **步骤详解**：告诉读者如何一步步实现案例，以便于读者自己动手实现案例效果，达到加深理解、锻炼自我的目的。
- **代码添加与解释**：告诉读者如何实现此案例的功能，让读者理解代码的运用及解决问题的思路；同时，还告诉读者代码的功能和解决的问题。
- **程序员代码与解释**：附带了所有的该案例的源代码，只要将这些源代码复制到自己的程序中，就能够实现功能，以方便读者进行验证；同时对于程序中应用到的难以理解的函数或者对象，都添加了详细的解释，以便于读者弄清楚问题的根源。

这些案例的源代码，既可以用来试验某个编程的概念，也可以应用到自己的程序中，实现某种特定的功能，达到迅速解决问题的效果。

通过阅读本书，你将学习到以下几个方面的知识：

- 第1章“C# Windows编程基础”。通过对一些具体实际案例的剖析，简单介绍了C#的重要功能和属性。目的是为加深读者对C#直观的理解。
- 第2章“图形图像处理”。介绍图形处理中的GDI技术。
- 第3章“多媒体应用”。通过对一些我们常见的游戏分析（贪食蛇、联五子等）介绍C#中复杂Windows程序的方法。
- 第4章“系统文件处理”。介绍如何在程序中控制系统硬件和外围设备，以及调用Windows的底层控件。
- 第5章“数据库应用”。介绍如何在C#程序中开发数据库的方法和技巧，重点是ADO.NET的使用方法。
- 第6章“C#网络开发”。介绍了利用网络编程的技巧和方法，以及常见的网络协议的应用。
- 第7章“C# Web编程应用”。介绍了在VS.NET环境下，利用ASP.NET编程的技巧和方法。
- 第8章“C#综合实例编程应用”。通过对目前比较流行的水晶报表、Web Service、.NET Remoting等技术实例的解析，深入细致地诠释了.NET技术的内涵和开发方法。

本书配套光碟的使用

为了方便读者阅读、测试和使用案例，本书附带了每个案例的源代码。源代码严格按照章节编排，每个案例目录下附带了所有用到的资源文件，如图片、数据表等。每个案例都是作者在WinXP + VS 2003下进行严格测试后再组织起来的。可能存在的问题是：案例文件打开后无法找到引用的文件、资源文件。这主要有两个方面的原因，一是作者保存案例文件的位置与你的不同，二是作者的机器名与你的设置不一致。如果出现这两个方面的问题，你可以遵照实现步骤自己制作案例程序，然后把光碟中的源代码复制到相应的程序部分，或者打开程序中的对象检查器，重新根据自己的机器设置相关属性。

本书是多人智慧的结晶，除封面书名的作者外，参与本书整理资料和制作的人员还有沈静博、刘洋、李永杭、常浩、牛永杰、段韬、吴建军、周欣、潘飞、胥虎军、黄艳华、郭茜、钟欣等。这些同志虽然从事着繁重的项目开发工作，但是仍然在工作之余，兢兢业业、严谨认真地编写，保证了本书的质量和进度。由于作者水平有限，尽管做了严格的审核和测试，书中难免仍有错误，敬请广大读者批评指正，在此深表感谢！

目　录

第1章

C# Windows编程基础

本章内容

- 循环语言的学习——砝码程序验证
- 随机数的案例——洗牌程序
- 实例剖析C#继承机制
- 递归与全局变量的案例1——打靶程序
- 递归与全局变量的案例2——二叉树
- 引用类型的举例——消去字符串空格
- 委托的使用方法实例——加减大小比较
- 用C#索引器实现文本文件的倒序输入
- C#文本文件操作实例——杨辉三角形的写入与读出

案例1　循环语言的学习——砝码程序验证

实例演示与说明

在本例中，我们将解决一个砝码程序验证的问题——4个砝码，每个重量都是整克，总重量是40克，天平可以称出1～40克的物体，问这4个砝码各是多少克？在通常的情况下是比较难计算的。通过本例，我们将学习循环语言的使用方法。案例运行效果如图1-1所示。

```
F:\study\c#\21\11>csc fana.cs
Microsoft (R) Visual C# .NET 编译器版本 7.10.3052.4
用于 Microsoft (R) .NET Framework 版本 1.1.4322
版权所有 (C) Microsoft Corporation 2001-2002。保留所有权利。

F:\study\c#\21\11>fana
1       3       9       27
1       9       27      9
1       9       3       27
1       9       27      3
1       27      3       9
1       27      9       3
3       1       9       27
3       1       27      9
3       9       1       27
3       9       27      1
3       27      1       9
3       27      9       1
9       1       3       27
9       1       27      3
9       3       1       27
9       3       27      1
9       27      1       3
9       27      3       1
27      1       3       9
27      1       9       3
27      3       1       9
27      3       9       1
27      9       1       3
27      9       3       1
DONE
```

图1-1 砝码程序案例

要点说明

1. 本例中我们使用for循环语句。一个循环的3个关键部件有：

（1）循环条件——当求值为true时，循环体重复执行。

（2）循环初始化——初始化循环时，参与循环条件的变量被初始化为适当值。此过程只在循环开始前发生一次。

（3）循环更新——更新循环的条件变量。在每次循环时都要重复进行更新。

下面是一个给出了循环的3个关键部件的示例：

```
01:  int index;
02:  index = 0;              // 循环初始化
03:  while(index<5)          // 循环条件
04:  {
05:      Console.WriteLine(index);
06:      index++;            // 循环更新
07:  }
```

示例是一个while循环，我们再写一个for循环与之对照：

```
int index;
for(index=0; index<5;index++)
    Console.WriteLine(index);
```

上面的for循环语法把循环初始化，循环条件，循环更新放在一行语句中。

实现步骤

1. 新建一个C#控制台应用程序项目，文件名为fama.cs；
2. 添加相应代码实现；
3. 执行程序。

代码添加与解释

1. 分析程序，先假设4个砝码的重量分别是a，b，c，d；那么它们重量范围为1～40克，这样就调用循环语句，设能够称的克数为r，r必然是从1～40循环的；还有一点要注意的是a和b有可能分别放在天平左边或右边（如a是1克，c是3克，a和一个重量r为2克的东西放在左边，c放右边），这样称出来的结果就是r=c-a，所以我们再设置4个变量i，j，k，l，设置他们的3种可能左、中、右，也就是－1，0，1；再设置循环语句i从－1循环到1，建立验证函数Check代码如下：

```
static bool Check(int a,int b,int c,int d)
    {
        int i,j,k,l;
        i = j = k = l = 0;
        for(int r = 1;r <= 40;r++)
        {
            for(i = -1;i <= 1;i++)
                for(j = -1;j <= 1;j++)
                    for(k = -1;k <= 1;k++)
                        for(l = -1;l <= 1;l++)
                        {
                            //如果可以称r重量，则直接跳出去称r+1的重量
                            if(a*i + b*j + c*k + d*l == r)
                                goto NextLoop;
                        }
```

```
            //如果不能称r重量，则说明失败，返回假
            return false;
        NextLoop:
            continue;
        }
        return true;
```

2. 主函数属性设定，只要a, b, c, d之和等于40，就可以继续进行。执行主函数代码如下：

```
static void Main( )
    {
        int a,b,c,d;
        for(a = 1;a < 40;a++)
            for(b = 1;b < 40;b++)
                for(c = 1;c < 40;c++)
                    for(d = 1;d < 40;d++)
                    {
                        if(a + b + c + d == 40)
                        {
                            if(Check(a,b,c,d))
                            {
                                Console.WriteLine("{0}\t{1}\t{2}\t{3}",a,b,c,d);
                            }
                        }
                    }
        Console.WriteLine("DONE");
        Console.ReadLine( );
    }
```

程序源代码与解释

```
using System;
public class myApp
{
    static void Main( )
    {
        int a,b,c,d;
        for(a = 1;a < 40;a++)
            for(b = 1;b < 40;b++)
                for(c = 1;c < 40;c++)
                    for(d = 1;d < 40;d++)
                    {
                        if(a + b + c + d == 40)
                        {
                            if(Check(a,b,c,d))
                            {
```

```
                                Console.WriteLine("{0}\t{1}\t{2}\t{3}",a,b,c,d);
                            }
                        }
                    }
        Console.WriteLine("DONE");
        Console.ReadLine( );
    }
    static bool Check(int a,int b,int c,int d)
    {
        int i,j,k,l;
        i = j = k = l = 0;
        for(int r = 1;r <= 40;r++)
        {
            for(i = -1;i <= 1;i++)
                for(j = -1;j <= 1;j++)
                    for(k = -1;k <= 1;k++)
                        for(l = -1;l <= 1;l++)
                        {
                            //如果可以称r重量，则直接跳出去称r+1的重量
                            if(a*i + b*j + c*k + d*l == r)
                                goto NextLoop;
                        }
            //如果不能称r重量，则说明失败，返回假
            return false;
        NextLoop:
            continue;
        }
        return true;
    }
}
```

案例2　随机数的案例——洗牌程序

实例演示与说明

在本例中，我们将解决一个洗牌程序的问题——52张牌（这里不算大小王），假设有一个人随机的洗牌，这就会打乱牌的顺序。我们在这个程序中将产生一个数组模拟牌，随机数模拟发牌的顺序。通过本例，我们将学习随机数的创建与数组的使用方法。案例运行效果如图1-2所示。

图1-2 洗牌程序案例

要点说明

1. C#数组介绍

数组是具有相同类型的一组数据。当访问数组中的数据时，可以通过下标来指明。C#中数组元素可以为任何数据类型，数组下标从0开始，即第一个元素对应的下标为0，以后逐个递增。数组可以是一维也可以是多维。

（1）包含6个元素的一维整数数组：

```
int[] mf1=new int[6];    //注意初始化数组的范围，或者指定初值；
```

（2）包含6个元素的一维整数数组，初值为1，2，3，4，5，6：

```
int[] mf2=new int[6]{1,2,3,4,5,6};
```

（3）一维字符串数组，如果提供了初始值设定项，则还可以省略new运算符：

```
string[] mf3={"c","c++","c#"};
```

（4）一维对象数组：

```
Object[] mf4 = new Object[5] { 26, 27, 28, 29, 30 };
```

（5）二维整数数组，初值mf5[0,0]=1,mf5[0,1]=2,mf5[1,0]=3,mf5[1,1]=4：

```
int[,] mf5=new int[,]{{1,2},{3,4}};
```

（6）6*6的二维整型数组：

```
int[,] mf6=new mf[6,6];
```

在声明一个数组的时候，方括号必须跟在类型后面，而不能跟在变量名后面。

```
int[] table;          //不能写成int table[]
```

在C#中你可以不指定数组的大小，这样就可以指定任意长度的数组。

```
int[] numbers;        //它的长度是任意的
```

当然，你也可以指定它的大小。

```
int[10] numbers;      //指定了一个长度为10的数组
```

2. 随机数的使用

计算机不可能产生完全随机的数字，所谓的随机数发生器都是通过一定的算法对事先选定的随机种子做复杂的运算，用产生的结果来近似地模拟完全随机数，这种随机数被称做伪随机数。伪随机数是以相同的概率从一组有限的数字中选取的。所选数字并不具有完全的

随机性，但是从实用的角度而言，其随机程度已足够。伪随机数的选择是从随机种子开始的，所以为了保证每次得到的伪随机数都足够地“随机”，随机种子的选择就显得非常重要。如果随机种子一样，那么同一个随机数发生器产生的随机数也会一样。一般地，我们使用同系统时间有关的参数作为随机种子，这也是.NET Framework中的随机数发生器默认采用的方法。

我们可以使用两种方式初始化一个随机数发生器：

（1）不指定随机种子，系统自动选取当前时间作为随机种子。

```
Random random = new Random( );
```

（2）可以指定一个int型参数作为随机种子。

```
Random random = new Random(10);
```

之后，我们就可以使用这个Random类的对象来产生随机数，这时候要用到Random.Next( )方法。这个方法使用相当灵活，你甚至可以指定产生的随机数的上下限。

除了Random.Next( )方法之外，Random类还提供了Random.NextDouble( )方法产生一个范围在0.0～1.0的随机的双精度浮点数：

```
double dResult;
dResult=random.NextDouble( );
```

另外一个与Random.NextDouble( )相似的方法是Random.Sample( )，它跟Random.NextDouble( )方法惟一的区别在于访问级别。

在本例中，我们将使用Random.NextDouble( )方法产生随机数。

实现步骤

1. 建一个C#控制台应用程序项目，文件名为sjs2.cs；
2. 添加相应代码实现；
3. 执行程序。

代码添加与解释

1. 分析程序，一共有52张牌，那么就要建立一个容量为52的数组，首先按照顺序发牌。这里注意a[0]存储1，a[1]存储2……a[51]存储52，代码如下：

```
//声明1个一维数组，没有初始化
        int[] a;
        int i,tmp,randtmp;
        //初始化两个一维数组
         a = new int[52];

        for(i=1;i<=52;i++)
        {
            a[i-1] = i;
        }
```

2. 然后产生随机函数，这里的办法是抽出第1张牌，然后把它随机插到某给位置上，然后把那个位置上的牌放在第1张牌的地方，抽出第2张牌，然后把它随机插到某个位置上，然后把那个位置上的牌放在第2张牌的地方……抽出第52张牌，然后把它随机插到某给位置上，然后把那个位置上的牌放在第52张牌的地方。这样就达到了随机发牌。代码如下：

```
//随机数产生函数
        Random r1 = new Random( );
        for(i=0;i<52;i++)
        {
        ////产生一个0～1的随机数然后乘以52就变成0～51的随机数了
            randtmp = (int)(52*r1.NextDouble( ));
            //把i位置上的牌和产生的随机的randtmp位置上的牌交换
            tmp=a[i];
            a[i]=a[randtmp];
            a[randtmp]=tmp;
        }
```

程序源代码与解释

```
using System;
public class myApp
{
    public static void Main( )
    {
        int randomum1,randomum2,flag=0;
        //声明1个一维数组，没有初始化
        int[] a;
        int i,tmp,randtmp;
        //初始化两个一维数组
         a = new int[52];
        for(i=1;i<=52;i++)
        {
            a[i-1] = i;
        }
        //随机数产生函数
        Random r1 = new Random( );
        for(i=0;i<52;i++)
        {
            ////产生一个0～1之间的随机数然后乘以52再取整就变成0～51的随机数了
            randtmp = (int)(52*r1.NextDouble( ));
            //把i位置上的牌和产生的随机的randtmp位置上的牌交换
            tmp=a[i];
```

```
            a[i]=a[randtmp];
            a[randtmp]=tmp;
        }
        for(i=0;i<52;i++)
            Console.Write("    {0}",a[i]);
    }
}
```

案例3　实例剖析C#继承机制

实例演示与说明

在本例中，我们将来剖析C#继承机制——先建立一个a类，然后建立一个b类，b类是从a类继承而来的。现在我们要分析其中的继承关系。通过本例，我们将学习C#继承机制的规则与使用方法。案例运行效果如图1-3所示。

图1-3 C#继承机制案例

要点说明

1. 继承基础知识

C#这种完全面向对象的程序设计语言提供了两个重要的特性——继承性inheritance和多态性polymorphism。

继承是面向对象程序设计的主要特征之一，它可以让你重用代码，也可以节省程序设计的时间。继承就是在类之间建立一种相交关系，使得新定义的派生类的实例可以继承已有的基类的特征和能力，而且可以加入新的特性或者是修改已有的特性建立起类的新层次。

现实世界中的许多实体之间不是相互孤立的，它们往往具有共同的特征，也存在内在的差别。人们可以采用层次结构来描述这些实体之间的相似之处和不同之处。

为了用软件语言对现实世界中的层次结构进行模型化，面向对象的程序设计技术引入了继承的概念。一个类从另一个类派生出来时，派生类从基类那里继承特性。派生类也可以作为其他类的基类。从一个基类派生出来的多层类形成了类的层次结构。

C#中，派生类只能从一个类中继承。这是因为，在C++中，人们在大多数情况下不需要一个从多个类中派生的类。从多个基类中派生一个类这往往会带来许多问题，从而抵消了这种灵活性带来的优势。

C#中，派生类从它的直接基类中继承成员：方法、域、属性、事件、索引指示器。除了构造函数和析构函数，派生类隐式地继承了直接基类的所有成员。

2. C#中的继承符合下列规则

（1）继承是可传递的。如果C从B中派生，B又从A中派生，那么C不仅继承了B中声明的成员，同样也继承了A中的成员。Object类作为所有类的基类。

（2）派生类应当是对基类的扩展。派生类可以添加新的成员，但不能除去已经继承的成员的定义。

（3）构造函数和析构函数不能被继承。除此以外的其他成员，不论对它们定义了怎样的访问方式，都能被继承。基类中成员的访问方式只能决定派生类能否访问它们。

（4）派生类如果定义了与继承而来的成员同名的新成员，就可以覆盖已继承的成员。但这并不意味着派生类删除了这些成员，只是不能再访问这些成员。

（5）类可以定义虚方法、虚属性及虚索引指示器，它的派生类能够重载这些成员，从而使得类可以展示出多态性。

（6）派生类只能从一个类中继承，可以通过接口实现多重继承。

3. 一个子类继承父类的例子：

```
using System;
public class ParentClass
{
    public ParentClass( )
    { Console.WriteLine("父类构造函数。"); }
    public void print( )
    { Console.WriteLine("I'm a Parent Class。") ; }
}
public class ChildClass : ParentClass
{
    public ChildClass( )
    { Console.WriteLine("子类构造函数。") ; }
}
class myApp
{
    public static void Main( )
    {
        ChildClass child = new ChildClass( ) ;
        //ParentClass child2 ;

        // ChildClass child2 ;
        // child2 = new ChildClass( ) ;
        //child.print( ) ;
         //child2.print( ) ;
    }
}
ChildClass child = new ChildClass( ) ;  //可以分成两个部分的
```

```
ChildClass child;                    //声明了一个对象child但没有初始化
child = new ChildClass( ) ;          //初始化对象child
```

虽然child对象是ChildClass类的，但ChildClass类是继承ParentClass类的。初始化对象child的时候就得先初始化基类ParentClass。但这并不意味ChildClass类继承ParentClass类的构造函数和析构函数。而是说ChildClass类可以调用它而已。

所以本题的结果是：

父类构造函数。

子类构造函数。

如果我们把ChildClass child = new ChildClass( );换成ParentClass child = new ChildClass( );这也是对的。前者是基类声明了一个对象child，然后你用派生类ChildClass将其初始化，这样是可以的。举例如下：

```
using System ;
public class schoolClass
{
    public schoolClass( )
    { Console.WriteLine("西北大学构造函数。"); }
    public  virtual void print( )
    { Console.WriteLine("我是西北大学的") ; }
}
public class partClass : schoolClass
{
    public partClass ( )
    { Console.WriteLine("西北大学物理系构造函数。") ; }
    public override void print( )
    { Console.WriteLine("我是西北大学物理系的") ; }
}
class myApp
{
    public static void Main( )
    {

        schoolClass jin1 ;

        jin1 = new partClass( ) ;
        jin1.print( );

    }
}
```

先声明了一个对象jin1，只是声明jin1是西北大学的。jin1 = new partClass( );使声明的jin1具备了西北大学物理系中的西北大学的性质（而jin1并不具备物理系独一无二的性质，因为jin1对象始终是西北大学类的对象，而不是物理系的对象），也就是将这个对象特化了。

partClass child = new schoolClass ( );这就是错的，因为你无法隐式把ParentClass类转