

# C#程序设计



## 第5章 方法

杨琦

西安交通大学  
计算机教学实验中心

<http://ctec.xjtu.edu.cn>

# 授课内容



- 5.1 方法概述
- 5.2 方法的定义
- 5.3 方法的调用
- 5.4 方法间的参数传递
- 5.5 方法与数组
- 5.6 局部变量和静态变量
- 5.7 委托
- 5.8 事件
- 程序设计举例
- 调试技术

## 5.1 方法概述

- 一个方法是把一些相关的语句组织在一起，用来执行一个任务的语句块。每一个 C# 程序至少有一个带有 Main 方法的类。

- 使用一个方法：

- 定义方法

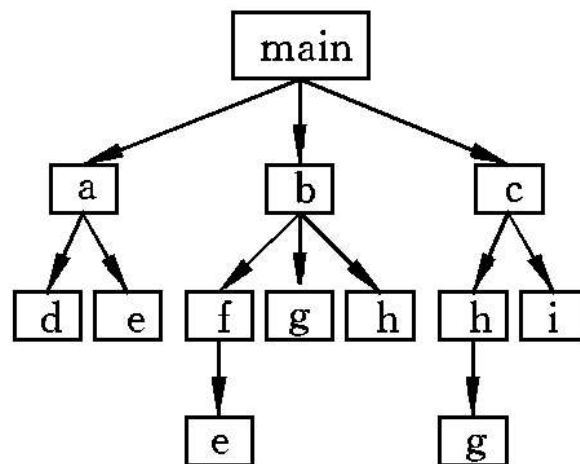
- 调用方法

- C#提供三种类型的方法：

- Main主方法

- 标准库方法

- 用户自定义方法



# 说明:



- (1) 一个源程序文件由一个或多个方法组成。
- (2) 一个 C# 程序由一个或多个源程序文件组成。
- (3) C# 程序的执行从 Main 方法开始。
- (4) 所有方法都是平行的，即在定义方法时是互相独立的。
- (5) 从用户使用的角度看，方法有两种：
  - ① 标准方法；② 用户自己定义的方法
- (6) 从方法的形式看，方法分两类：
  - ① 无参方法；② 有参方法

## 【例5-1】编写一个求阶乘 $n!$ 的方法



```
1.  class My{
2.      static int fac(int n)  {
3.          int result = 1;
4.          if (n < 0)    return -1;
5.          else if (n == 0)
6.              return 1;
7.          while (n > 1)  {
8.              result *= n;
9.              n--;
10.         }
11.         return result;
12.     }
```



## 【例5-1】编写一个求阶乘 $n!$ 的方法

```
1. static int Main()
2. {
3.     int n;
4.     Console.WriteLine("Please input a number n: ");
5.     n=Convert.ToInt32( Console.ReadLine() );
6.     Console.WriteLine( n + "! = " + fac(n) );
7.     return 0;
8. }
9. }
```

The screenshot shows a Windows command prompt window with the title bar "C:\WINDOWS\system32\cmd.exe". The window contains the following text: "Please input a number n to calculate n! :", "6", "6! = 720", and "请按任意键继续. . .". At the bottom, there is a search bar with the text "搜狗拼音 半:".



## 5.2 方法体

- [格式]:
  - 方法修饰符 数据类型 方法名([形式参数列表])
  - {
  - 变量、数组的定义语句;
  - 其它可执行部分
  - }

# 方法修饰符——C#封装



- 一个 **访问修饰符** 定义了一个类成员的范围和可见性。
- 分五类：
  - **public**: 所有对象都可以访问；
  - **private**: 对象本身在对象内部可以访问；
  - **protected**: 只有该类对象及其子类对象可以访问
  - **internal**: 同一个程序集的对象可以访问；
  - **protected internal**: 访问限于当前程序集或派生自包含类的类型。





## 2. 方法返回值声明

- 方法值类型，是通过方法体内部的return语句提供。
- **return** 表达式的值的**类型**应与方法说明中的**方法值类型**一致。
- 如果某一方法确实没有返回值，则使用说明符**void**。

例如：主方法

```
static int Main()  
{  
    ... ..  
    return 0;  
}
```



### 3. 形式参数声明

表示将从主调方法中接收哪些类型的信息

现代的方式:

例: `double grav(double m1, double m2, double distance)`

参数说明格式为:

`<类型><参数1>, <类型><参数2>, ..., <类型><参数n>`

例: `int [] array, int count`

**static** 关键字把类成员定义为静态的。当声明一个类成员为静态时, 意味着无论有多少个类的对象被创建, 只会有一个该静态成员的副本。



## 【例5-2】求任意两个整数的最大数

- 输入和输出
- Enter two integers:
- 12
- 18
- The maxium number is 18

## 【例5-2】求任意两个整数的最大数



```
1.  using System;
2.  class My{
3.      static int max(int x, int y)  {
4.          return x > y ? x : y;
5.      }
6.      static int Main(string[] args)  {
7.          int a, b;
8.          Console.WriteLine("Enter two integers: ");
9.          a = Convert.ToInt32(Console.ReadLine());
10.         b = Convert.ToInt32(Console.ReadLine());
11.         Console.WriteLine("The maxium number is {0}", max(a, b));
12.         return 0;
13.     }
14. }
```



## 5.3 实例方法与静态方法

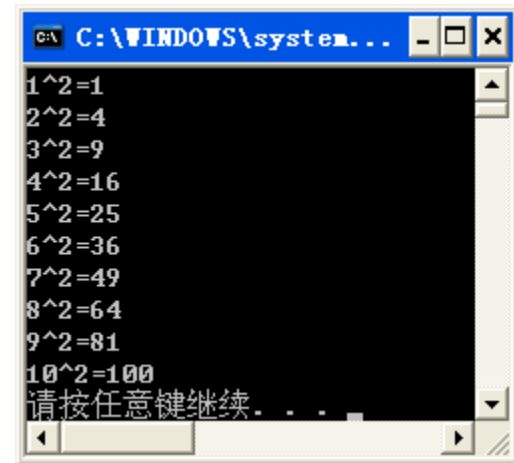
- 格式1：方法名([实际参数列表])
- 格式2：对象名.方法名([实际参数列表])
- 格式3：类名.方法名([实际参数列表])

## 5.3 实例方法与静态方法

- 例题：计算平方的方法

```
static int pf(int n) {  
    return n * n;  
}
```

```
static void Main(string[] args) {  
    int i,j;  
    for (i = 1; i <= 10; i++) {  
        j = pf(i) ;  
        Console.WriteLine("{0}^2={1}",i,j);  
    }  
}
```



```
C:\WINDOWS\system...  
1^2=1  
2^2=4  
3^2=9  
4^2=16  
5^2=25  
6^2=36  
7^2=49  
8^2=64  
9^2=81  
10^2=100  
请按任意键继续...
```

## 5.4 方法的调用





## 5.4 方法间的参数传递

- 参数传递是指实参传给形参的方式
- C#中的参数传递可分成三种：
  - 值参数
  - 引用参数
  - 输出参数
- 1. 值参数
  - 实参把值复制一份传给形参，形参接收了实参的值后与实参已不再存在任何联系。





## 【例5-3】交换两个变量值方法

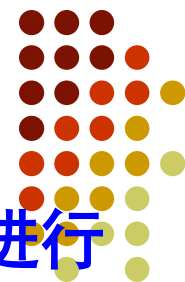
- 输入和输出
- Before exchange:  $a=2, b=3$
- Before exchange:  $a=2, b=3$



## 【例5-3】交换两个变量值方法

```
1.  class My{
2.      static void swap(int x, int y) {
3.          int tmp;
4.          tmp = x;
5.          x = y;
6.          y = tmp;
7.      }
8.      static int Main(string[] args) {
9.          int a = 2, b = 3;
10.         Console.WriteLine("Before exchange:  a={0} ,b={1} ", a, b);
11.         swap(a, b);
12.         Console.WriteLine("After exchange:  a={0} ,b={1} ", a, b);
13.         return 0;
14.     }
```

## 2. 引用参数



- 使用“引用参数”方式传递参数时，在方法中对形参进行修改也就修改了对应的实参。
- 使用格式：关键字ref。
  - 定义方法时，在形式参数的类型说明符前加上ref
  - 调用方法时，在实际参数之前加上ref。

## 【例5-4】利用引用变量编写交换方法



```
1.  class My {  
2.      static void swap(ref int x, ref int y)  {  
3.          int tmp = x;    x = y;    y = tmp;  
4.      }  
5.      static int Main()  {  
6.          int a = 2, b = 3;  
7.          Console.WriteLine("Before exchange:  a= " + a + ",  b= " + b);  
8.          swap(ref a, ref b);  
9.          Console.WriteLine("After  exchange:  a= " + a + ",  b= " + b);  
10.         return 0;  
11.     }  
12. }
```



## 5.6 数组的参数传递

- 方法之间可以传递多个值
- `bubble_up(array, 5);`      // 方法调用

输入和输出

原数组是:

**503 87 512 61 908 170 897 275 653 426**

对数组前**5**项进行排序后的结果是:

**61 87 503 512 908 170 897 275 653 426**

对整个数组排序后的结果是:

**61 87 170 275 426 503 512 653 897 908**



## 例5-6 编写对整型数组排序的方法

```
1.  using System;
2.  class My{
3.      static void bubble_up(int[] list, int count) {
4.          for (int i = 0; i < count; i = i + 1)
5.              for (int j = count - 1; j > i; j = j - 1)
6.                  if (list[j - 1] > list[j])
7.                      {
8.                          int tmp = list[j - 1];
9.                          list[j - 1] = list[j];
10.                         list[j] = tmp;
11.                     }
12.      }
13.
```



## 例5-6 编写对整型数组排序的方法

```
1. static int Main(string[] args) {  
2.     int i;  
3.     int[] array = new int[10] {  
4.         503, 87, 512, 61, 908, 170, 897, 275, 653, 426 };  
  
5.     Console.WriteLine("原数组是:");  
6.     for (i = 0; i < 10; i++)  
7.         Console.Write("{0} ", array[i]);  
8.     Console.WriteLine();  
9.     bubble_up(array, 5);
```

## 例5-6 编写对整型数组排序的方法



```
1. Console.WriteLine("对数组前5项进行排序后的结果是:");
2.   for (i = 0; i < 10; i++)
3.       Console.Write("{0} ", array[i]);
4.   Console.WriteLine();
5.   bubble_up(array, 10);
6.   Console.WriteLine("对整个数组排序后的结果是:");
7.   for (i = 0; i < 10; i++)
8.       Console.Write("{0} ", array[i]);
9.   Console.WriteLine();
10.  return 0;
11. }
12. }
```





## 例5-7 打印1000~10000的回文数

- 回文数例子：
  - 123454321      463364      9889
- 不是回文数的例子：
  - 1234567890      9988      71234
- 算法分析：
  - 穷举判断1000-10000中每个数是否是回文数
  - 任何4位整数变成反序整数：
    - 原数除以10取余数
    - 新数乘10加上本次余数
    - 原数整除10，直到原数为0停止循环
  - 比较新数与原数是否相等，若相等，原数是回文数



## 例5-7 打印1000~10000的回文数

- 输入和输出

- 1001 1111 1221 1331 1441 1551 1661 1771 1881 1991
- 2002 2112 2222 2332 2442 2552 2662 2772 2882 2992
- 3003 3113 3223 3333 3443 3553 3663 3773 3883 3993
- 4004 4114 4224 4334 4444 4554 4664 4774 4884 4994
- 5005 5115 5225 5335 5445 5555 5665 5775 5885 5995
- 6006 6116 6226 6336 6446 6556 6666 6776 6886 6996
- 7007 7117 7227 7337 7447 7557 7667 7777 7887 7997
- 8008 8118 8228 8338 8448 8558 8668 8778 8888 8998
- 9009 9119 9229 9339 9449 9559 9669 9779 9889 9999

## 例5-7 打印1000~10000的回文数



```
1.  static int Main()
2.      {
3.          for (int i = 1000; i < 10000; i++)
4.          {
5.              if (IsPalindrome(i))
6.                  Console.WriteLine("{0}\t", i);
7.          }
8.          Console.WriteLine();
9.          return 0;
10.     }
```

## 例5-6 打印回文数



```
1. static bool lspalindrome(int n)
2. {
3.     int k, m = 0;
4.     k = n;
5.     while (k!=0)
6.     {
7.         m = m * 10 + k % 10;
8.         k = k / 10;
9.     }
10.    return (m == n);
11. }
```



## 【例5-8】数组清零

- 输入和输出
- 数组清零前的结果是：  
503 87 512 61 908 170 897 275 653 426
- 输入数组元素：  
5
- 对数组清零后的结果是：  
0 0 0 0 0 170 897 275 653 426
-

## 【例5-8】数组清零



```
1.  class My{
2.      static void clear_array(double[] ptr, int n)  {
3.          for (int i = 0; i < n; i++)
4.          {
5.              ptr[i] = 0.0;
6.          }
7.      }
8.
```

## 【例5-8】数组清零



```
1. static int Main() {  
2.     int i,n, count = 10;  
3.     double[] array = new double[10] {503, 87, 512, 61, 908,  
4. 170, 897, 275,653, 426};  
5.     Console.WriteLine("数组清零前的结果是: ");  
6.     for (i = 0; i < count; i++)  
7.         Console.Write(array[i] + " ");  
8.     Console.WriteLine();  
9.     Console.WriteLine("输入数组元素: ");  
10.    n = Convert.ToInt32(Console.ReadLine());  
11.    clear_array(array, n);
```

## 【例5-8】数组清零



```
1. Console.WriteLine("对数组清零后的结果是: ");
2.     for (i = 0; i < count; i++)
3.         Console.Write(array[i] + " ");
4.     Console.WriteLine();
5.     return 0;
6. }
7. }
```





## 【例5-9】 实现矩阵相乘运算

1. 算法说明：
2. 矩阵A ( $L \times N$ ) 和矩阵B ( $N \times M$ ) 相乘。
3. A是L行、N列；B是N行、M列。
4. 要求:A的列数和B的行数必须相同。

$$C_{ij} = \sum_{k=1}^M A_{ik} \times B_{kj} \quad , i = 1, 2, \dots, L; j = 1, 2, \dots, N$$

## 算法说明（续）



$$A = \begin{vmatrix} 1 & 0 & 3 \\ 2 & 1 & 1 \end{vmatrix} \quad B = \begin{vmatrix} 3 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \end{vmatrix}$$

$$\begin{aligned} A \times B &= \begin{vmatrix} \begin{matrix} r11 \\ 1*3+0*2+3*1 \end{matrix} & \begin{matrix} r12 \\ 1*1+0*0+3*2 \end{matrix} & \begin{matrix} r13 \\ 1*2+0*1+3*1 \end{matrix} \\ \begin{matrix} 2*3+1*2+1*1 \\ r21 \end{matrix} & \begin{matrix} 2*1+1*0+1*2 \\ r22 \end{matrix} & \begin{matrix} 2*2+1*1+1*1 \\ r23 \end{matrix} \end{vmatrix} \\ &= C \end{aligned}$$

# 矩阵乘法算法



- 用两重循环实现对  $C_{ij}$  的求值:
- for( $i=0$ ;  $i<L$ ;  $i=i+1$ )
- for( $j=0$ ;  $j<N$ ;  $j=j+1$ )
- 求  $C_{ij}$ ;
- 其中 “求  $C_{ij}$ ” 又可以细化为:
- $C_{ij} = 0$ ;
- for( $k=0$ ;  $k<M$ ;  $k=k+1$ )
- $C_{ij} = C_{ij} + A_{ik} \times B_{kj}$



## 【例5-9】 实现矩阵相乘运算

- 输入和输出
- The result is c=
- 32 15 -9
- 43 27 24
- -1 -21 77
- 29 33 -5
-



## 【例5-9】 实现矩阵相乘运算

```
1.  using System;
2.  class My{
3.      // 方法matrix_multi(): 计算两个矩阵的乘积
4.      static void matrix_multi(double[] a, double[] b, double[] c,
5.  int l, int m, int n)
6.      {
7.          int i, j, k;
8.          for (i = 0; i < l; i++)
9.              for (j = 0; j < n; j++)    {
10.                  c[i * n + j] = 0;
11.                  for (k = 0; k < m; k++)
12.                      c[i * n + j] = c[i * n + j] + a[i * m + k] * b[k * n + j];
13.              }
14.      }
```



## 【例5-9】 实现矩阵相乘运算

```
1. // 测试上述矩阵相乘方法的主程序
2. static int Main(string[] args) {
3.     double[] a = new double[20] {1.0, 3.0,-2.0, 0.0, 4.0,
4.                                   -2.0,-1.0, 5.0,-7.0, 2.0,  0.0, 8.0, 4.0, 1.0,-5.0,
5.                                   3.0,-3.0, 2.0,-4.0, 1.0
6.     };
```



## 【例5-9】 实现矩阵相乘运算

```
1.  double[] b = new double[15]  {  
2.           4.0, 5.0,-1.0,           2.0,-2.0, 6.0,  
3.           7.0, 8.0, 1.0,           0.0, 3.0,-5.0,  
4.           9.0, 8.0,-6.0  
5.  };  
6.  double[] c = new double[12];  
7.  matrix_multi(a, b, c, 4, 5, 3);  
8.  Console.WriteLine("The result is c=");  
9.  for (int i = 0; i < 4; i++)    {  
10.      for (int j = 0; j < 3; j++)  
11.          Console.Write("{0}  ", c[i * 3 + j]);  
12.      Console.WriteLine();  
13.  }  
14.  return 0;
```



## 【例5-10】英文月份名称

- 输入和输出
- 请输入月份数值:
- 3
- 3月的英文名称是March



## 【例5-10】英文月份名称



```
1.  class My{
2.      static string[] month = new string[13]    {
3.          "Illegal month",      "January",      "February",
4.          "March",              "April","May", "June","July",
5.          "August","September","October",
6.          "November",  "December"    };

7.      static string month_name(int n)  {
8.          return (n >= 1 && n <= 12) ? month[n] : month[0];
9.      }
```



## 【例5-10】英文月份名称

```
1.  static int Main()
2.  {
3.      int n;
4.      Console.WriteLine("请输入月份数值:");
5.      n = Convert.ToInt32(Console.ReadLine());
6.      Console.WriteLine("{0}月的英文名称是{1}" ,n,
7.  month_name(n));
8.      return 0;
9.  }
10. }
```



## 5.7 委托

- C# 中的委托类似于 C 或 C++ 中的函数指针
- 使用委托包括以下步骤：
  - 声明委托。
  - 实例化委托。
  - 使用委托。
- 对于静态方法，委托对象封装要调用的方法。
- 对于实例方法，委托对象同时封装一个实例和该实例上的一个方法。
- 如果有一个委托对象和一组适当的参数，则可以用这些参数调用该委托。

## 【例5-10】通用数值积分方法（委托）



```
1.  class My{
2.      delegate double fun(double x);
3.      static double integral(double a, double b, fun f, int n)
4.      {
5.          double h = (b - a) / n;
6.          double sum = (f(a) + f(b)) / 2;
7.          for (int i = 1; i < n; i++)
8.              sum += f(a + i * h);
9.          sum *= h;
10.         return sum;
11.     }
12.
```

## 【例5-10】通用数值积分方法（委托）



```
1.  static int Main()
2.  {
3.      double x1, x2;
4.      Console.WriteLine("请输入积分区间:");
5.      x1 = Convert.ToDouble(Console.ReadLine());
6.      x2 = Convert.ToDouble(Console.ReadLine());
7.      Console.WriteLine("sin(x)结果是" + integral(x1, x2, Math.Sin, 1000));
8.      Console.WriteLine("cos(x)结果是" + integral(x1, x2, Math.Cos, 1000));
9.      Console.WriteLine("exp(x)结果是"
10. + integral(x1, x2, Math.Exp, 1000));
11.     return 0;
12. }
13. }
```



## 5.8 事件

- 事件是一种使对象或类能够提供通知的成员。
- C#中的事件处理步骤如下：
  - 定义事件。

[访问修饰符] **event** 委托名 事件名;

- 订阅该事件。

事件可用作 **+=** 和 **-=** 运算符左边的操作数

- 当事件发生时通知订阅者发生的事件。

要通知订阅某个事件的所有对象（即订阅者），需要引发该事件



## 【例5-11】事件的使用

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Text;
4.  namespace My {
5.      //创建一个事件类
6.      class MyEvent
7.      {
8.          //声明一个委托
9.          public delegate void Del();
10.         //声明一个事件
11.         public event Del Click;
```



## 【例5-11】事件的使用

```
1. //创建一个触发事件的方法
2. public void OnClick() {
3.     if (Click != null) {
4.         Console.WriteLine("引发事件: ");
5.         Click();
6.     }
7. }
8. //和事件关联的方法
9. public void ClickMethod() {
10.     Console.WriteLine("您触发了 Click 事件!");
11. }
12. }
```





## 【例5-11】事件的使用

```
1.  class Test
2.  {
3.      static int Main(string[] args)
4.      {
5.          MyEvent me = new MyEvent();
6.          //给对象预定事件
7.          me.Click += new MyEvent.Del(me.ClickMethod);
8.          me.OnClick();
9.          return 0;
10.     }
11. }
12. }
```

# 结 束 语



- 学好程序设计语言的唯一途径是

上机练习

- 你的编程能力与你在计算机上投入的时间成

正比