

第 2 章 C# 基础语法

内容简介

无规矩不成方圆，所有的编程语言都有自己的语法规则，用来说明如何使用这种语言来编写程序，C# 也不例外。为了让程序能够正确运行并减少错误的产生，就必须遵守这些语法规则。

本章将介绍 C# 最基本的语法知识，为后面的程序开发奠定基础。主要包括变量、常量、数据类型、类型转换以及运算符等内容。

例：控制台应用程序：将华氏温度转化为摄氏温度

其中，c 表示摄氏温度，f 表示华氏温度，其值从键盘输入。

$$c = \frac{3 \times (f - 32)}{9}$$

```

static void Main(string[] args)
{
    float c, f;
    string s;
    s = Console.ReadLine(); // 从键盘输入
    f = float.Parse(s);
    c = 5 * (f - 32) / 9;
    Console.WriteLine(" 华氏 {0} 度 = 摄氏 {1} 度 ", s, c.ToString());
    Console.ReadLine();
}

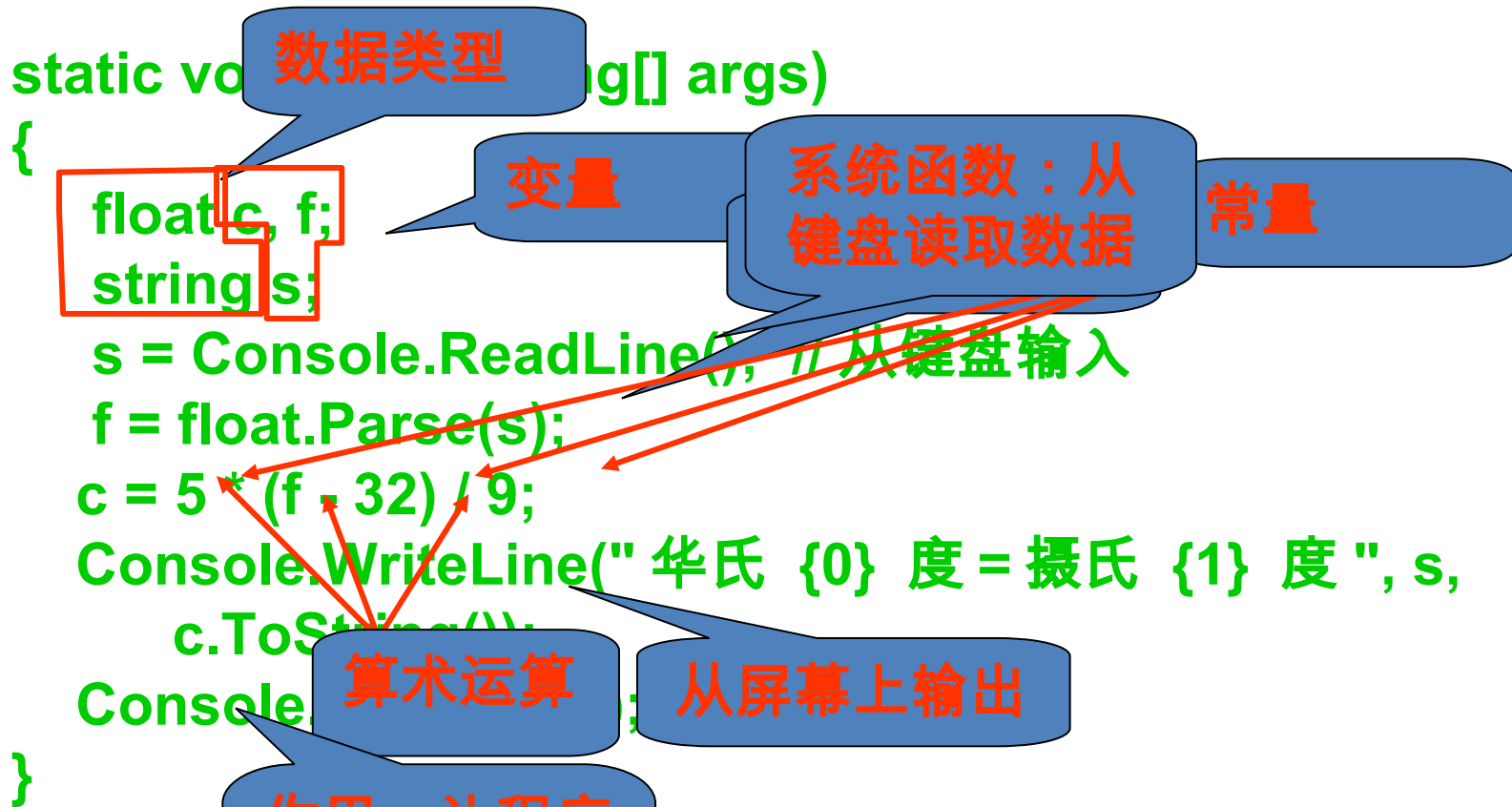
```

Parse

vt. 从语法上描述或分析 (词句等)

数据类型 .Parse(String s);





—— 正确理解数据类型、变量、常量数据类型转换、基本运算等概念是编写 C# 程序的关键

2.1 变量

无论编写任何应用程序，数据都必须以某种方式表示。变量和常量在 C# 中经常用到，熟练掌握变量和常量，使代码更容易维护，更具有可读性。这一节我们就主要来学习变量的相关知识。

2.1.1 变量概述

2.1.2 声明变量

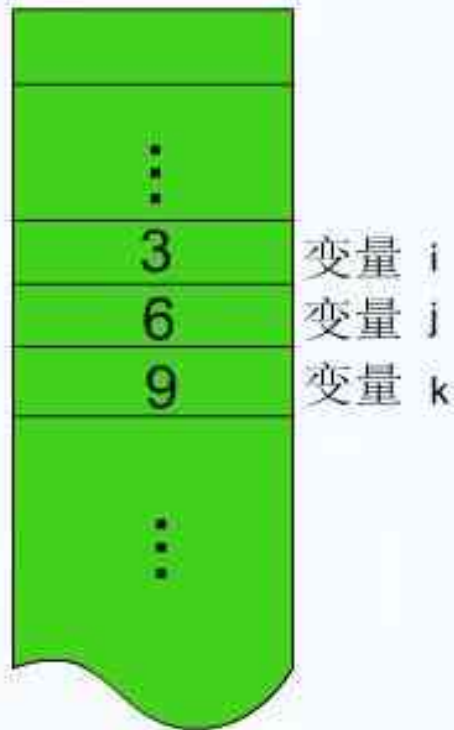
2.1.3 初始化变量

2.1.4 变量分类

2.1.1 变量概述

在 C# 中，变量就是存取信息的基本单元，它有两个基本特征，即**变量名**（标识变量的名称）和**变量值**（变量存储的数据）。对于变量，必须明确变量的命名、类型、声明以及作用域。

内存用户数据区



变量，内存中的一块存储区域，会对应一个唯一的内存地址。

内存地址不好理解也不好记忆。需要变量名。

2.1.2 声明变量

用户可以通过指定数据类型和标识符来声明变量。声明变量的语法：

```
DataType identifier;
```

或

```
DataType identifier = value;
```

DataType 指变量的类型，可以是 int、string、bool、char 或 double 类型等等。

identifier 标识符或者变量的名称。

value 指变量的值。多个同类型的变量可以同时定义，中间要使用逗号分隔。

定义一个变量名称，需要遵循相应的规则，如下所示：

- 变量名称必须以字母开头
- 变量名称只能由字母、数字和下划线组成，而不能包含空格、标点符号、运算符等其它符号
- 变量名称不能与 C# 中的关键字名称相同，如 using、static、namespace 和 class 等等
- 起名要有实际意义的名称，容易辨别数据类型

下面我们给出了一些合法和非法的变量名称的例子：

```
int j,k,m,n;           // 合法，声明同类型的多个变量
string stuName;        // 合法
decimal totalPrice;    // 合法
string stu.Age;        // 不合法，含有非法字符“.”
double pro price;      // 不合法，变量名称含有空格
int my-age, my_name;
char @use;             // 合法，与关键字 use 相同，加上前缀 @
```

2.1.3 初始化变量

初始化变量就是给变量指定一个明确的初始值，初始化变量有 2 种方式：一种是先声明、后赋值；一种是声明时直接赋值。如下所示：

```
int score;  
score = 100;  
或者  
string userName = "李延亮";
```

声明变量时有些类型的变量有默认值（初始值），有些变量是没有初始值的。以下 6 类中的变量属于初始已赋值的变量：

- 静态变量
- 初始已赋值的实例变量
- 数组元素
- 值参数
- 引用参数
- 在 catch 子句或 foreach 语句中声明的变量

以下 3 类中的变量属于初始未赋值的变量：

- 初始未赋值结构变量的实例变量
- 输出参数
- 局部变量，在 catch 子句或 foreach 语句中声明的那些除外

2.1.3 变量分类

在 C# 中，定义了 7 种不同类别的变量。它们是：静态变量、实例变量、数组元素、值参数、引用参数、输出参数和局部变量。

1. 静态变量

静态变量使用 `static` 修饰符表示，它在程序编译通过后存在，在关联的应用程序域终止时终止。静态变量使用时，在同一个类中可以直接调用，不同类中直接使用“类名.变量名”调用即可。

例：声明一个 `int` 类型的静态变量 `stuAge`，在 `Main()` 方法中将 `stuAge` 的值加 1，然后直接在控制台中输出。代码：

```
static int stuAge;  
public static void Main(string[] args)  
{  
    stuAge=stuAge + 1;  
    Console.WriteLine(stuAge);  
    Console.ReadLine();  
}
```

上例中 `stuAge` 没有赋初值，是否可用？运行结果如何？

2.1.3 变量分类

2. 实例变量

实例变量是指没有使用 static 关键字声明的变量。它和静态变量完全相反，又可以称为非静态变量或对象变量。

```
class Program
{
    public string articleTitle;
}
```

静态变量和实例变量的区别：

	静态变量	实例变量
内存分配	类装载时就开始分配内存	被实例化后才会分配内存
生存周期	即应用程序的存在周期	取决于实例化类的存在周期
调用方式	类名.变量名	实例名.变量名
共享方式	全局变量，被所有的类的实例对象共享	局部变量，不共享
取值	同类的所有实例的同一静态变量都是同一个值	同一个类的不同实例的同一非静态变量可以是不同的值

```
class Program
```

```
{
```

```
    public class student
```

```
    {    public string stuName;
```

```
        public static int stuAge;
```

```
        public student()
```

```
        {    stuAge= 20;    }
```

```
    }
```

```
    static void Main(string[] args)
```

```
    {
```

```
        student testOne = new student();
```

```
        student testTwo = new student();
```

```
        testOne.stuName = " 张三 ";
```

```
        testTwo.stuName = " 李四 ";
```

```
        student.stuAge++;
```

```
        Console.WriteLine(testOne.stuName);
```

```
        Console.WriteLine(testTwo.stuName);
```

```
        Console.WriteLine(student.stuAge);
```

```
    }
```

```
}
```

输出 ?

2.1.3 变量分类

3. 数组元素

数组元素是指将数组作为成员参数的元素，它在数组创建时开始存在，在没有对该数组实例的引用时停止存在。

例：在名称为 Program.cs 的文件中添加一个方法，将 int 类型的数组作为该方法的参数。

```
public void ArrayInt(int[] stuAges)
{
    stuAges[0]=10;
    stuAges[1]=20;
}
```

上述代码中数组变量 stuages 中的每一个元素都称为数组元素，然后在 ArrayInt() 方法中分别为数组元素赋值。

2.1.3 变量分类

4. 值参数

用户声明方法时可以在方法中传入参数。值参数就是在方法中传入参数，该参数不使用任何修饰符，与一般参数无异。**参数是按值传递。**

例：打开名称为 Program.cs 的文件，添加一个 GetTotalSum() 方法，在该方法中声明两个 int 类型的值参数 firstNum 和 secondNum，该方法返回两个参数的和。

```
public static int GetTotalNum(int firstNum, int secondNum)
{
    firstNum++;
    secondNum--;
    return firstNum + secondNum;
}
```

方法成员为值参数分配了一个新的存储位置。仅仅把“**值**”传进去，被调用的方法中所做的修改不会影响原调用方法中的参数值。

值参数

```
static void Main(string[] args)
{
    int a = 2;
    int b = 3;
    Console.WriteLine( plus( a, b) );
    Console.WriteLine(a.ToString()+" & "+ b.ToString());
    Console.ReadLine();
}
```

```
public static int plus (int first,int second)
{
    first = -1;
    second = -1;
    return first + second;
}
```

-2
2 & 3

2.1.3 变量分类

5. 引用参数

引用参数是指使用 **ref** 修饰符声明的参数。

注意：在方法成员中，引用参数仍然使用其基础变量的存储位置。方法中对参数的修改会影响到原调用方法中的参数的值。

例：在名称为 Program.cs 文件中添加一个新的方法 RefGetTotalSum() 方法，在该方法中声明两个 int 类型的引用参数 firstNum 和 secondNum，该方法返回两个参数的和。

```
public static int RefGetTotalSum(ref int firstNum, ref int secondNum)
{
    firstNum++;
    secondNum--;
    return firstNum+secondNum;
}
```

使用方法：在方法**定义**时和**调用**时都要使用 ref 修饰符；使用 ref 修饰的参数必须在调用的方法中赋值，不能为常量。

引用参数

```
class Program
{
    public static int mySum( ref int first, ref int second)
    {
        first++;
        second++;
        return first + second;
    }

    static void Main(string[] args)
    {
        int a, b,c;
        a = 23;
        b = 55;
        c = mySum(ref a, ref b);
        Console.WriteLine("{0},{1},{2}",a,b,c);

        Console.ReadKey();
    }
}
```

运行结果？

2.1.3 变量分类

6. 输出参数

输出参数是指使用 **out** 修饰符声明的参数。可以直接输出，其使用效果和有返回值 `return` 的效果是一样的。调用时也需要 **out** 修饰。

例：在名称为 `Program.cs` 文件中添加一个新的方法 `OutGetTotalSum()` 方法，在该方法中声明 3 个 `int` 类型的参数 `firstNum`、`secondNum` 和 `totalSum`，其中 `totalSum` 为输出参数，表示输出 `firstNum` 和 `secondNum` 的结果。

```
public static void OutGetTotalSum(int
firstNum, int secondNum, out int
totalSum)
{
    firstNum++;
    secondNum--;
    totalSum = firstNum + secondNum;
}
```

```
static void Main(string[] args)
{
    int a, b, c;
    a = 23;
    b = 55;
    c = 9;
    OutGetTotalSum(a, b, out c);
    Console.WriteLine("{0},{1},{2}", a, b, c);

    Console.ReadKey();
}
```

一个方法输出多个结果？
(上例中，和、差、积都要输出)

2.1.3 变量分类

7. 局部变量

在子程序中定义的变量就是局部变量，又称内部变量。它在某一段时间内存在，它的生存期从声明该变量开始，一直到它所在的区域结束为止。

```
public static void Main(string[] args)
{
    int i,j;
    i=2;    // 局部变量不具有初值，必须赋值
    j=2*i;
}
```

从形式上看，局部变量和类的成员变量相似，但在使用上区别很大：

- 1、系统不会自动为局部变量赋初值。
- 2、局部变量没有访问权限修饰符，不能用 `public`、`private` 和 `protected` 来修饰。这是因为它只能在定义它的方法内部使用。
- 3、局部变量不能用 `static` 修饰，没有“静态局部变量”。

2.2 常量

用户在进行圆的面积计算的时候，需要使用到圆周率 π ，可以把 π 声明为静态变量，如 `static float pi=3.14f`。这种情况下用户可以在其它地方随意更改圆周率的值。但是，我们知道，一般情况下，圆周率是不能随意更改的。那应该怎么办呢？为了解决这个问题，我们使用另一种方式来表示数据，这一节我们就来学习下常量。

2.2.1 常量概述

2.2.2 声明常量

2.2.1 常量概述

常量是指在使用过程中不会发生变化的量。

应用程序中使用常量至少有 3 个好处：

1. 常量用易于理解的清楚的名称替代了含义不明确的数字或字符串，使程序更易于阅读。
2. 常量使程序更易于修改。
3. 常量更容易避免程序出现错误。如果要把另一个值赋给程序中的一个常量，而该常量已经有了一个值，编译器就会报告错误。

2.2.2 声明常量

在 C# 中，使用 `const` 关键字（修饰符）定义的量就叫常量，也可以称作静态常量。

例如，我们更改本小节开始时提出的问题，重新声明并初始化圆周率 π 的值：

```
const float pi = 3.14f;
```

如果使用一个 `const` 关键字需要同时声明多个常量，那么常量之间需要使用逗号分隔。如下所示：

```
const int stuid=10,score=100,age=20;
```

使用 `const` 关键字定义常量时，需要注意以下几点：

- `const` 默认是静态的，不能和 `static` 同时使用
- `const` 必须在字段声明时就初始化，而不是先声明再赋值
- `const` 只能定义字段和局部变量
- `const` 只能应用在值类型和 `string` 类型上，其他引用类型常量只能定义为 `null`。否则会引发错误提示

2.3 数据类型

2.3.1 数据类型分类

2.3.2 值数据类型

2.3.3 引用类型

2.3.1 数据类型分类

在 C# 中，数据类型主要分为 3 类：

- 值类型 它的变量直接包含数据
- 引用类型 它的变量直接存储其数据的访问地址
- 指针类型 只能作用在不安全的代码中，和 C、C++ 语言中的指针类似。而且 C# 中很少使用指针类型，因此，不做详细介绍

2.3.2 值数据类型

所有的值类型都源自 System.ValueType 家族，每个值类型对象都有个独立的内存区域保存自己的值，只要在代码中修改它，就会在它的内存区域中保存这个值。值类型主要包括基本数据类型、结构类型和枚举类型等。

1. 基本数据类型

基本数据类型主要包括整数类型 int、实数类型、布尔类型 bool、字符类型 char 等。下表是常见的整数类型。

数据类型	说明	取值范围	对应于 System 程序集中的名称
sbyte	有符号 8 位整数	-128-127	System.SByte
byte	无符号 8 位整数	0-255	System.Byte
short	有符号 16 位整数	-32 768-32 767	System.Int16
ushort	无符号 16 位整数	0-65 535	System.UInt16
int	有符号 32 位整数	-2 147 483 648-2 147 483 647	System.Int32
uint	无符号 32 位整数	0-42994967295	System.UInt32
long	有符号 64 位整数	-2^{63} - 2^{63}	System.Int64
ulong	无符号 64 位整数	$0-2^{64}$	System.UInt64
char	无符号 16 位整数	0~65 535	System.Char

2.3.2 值数据类型

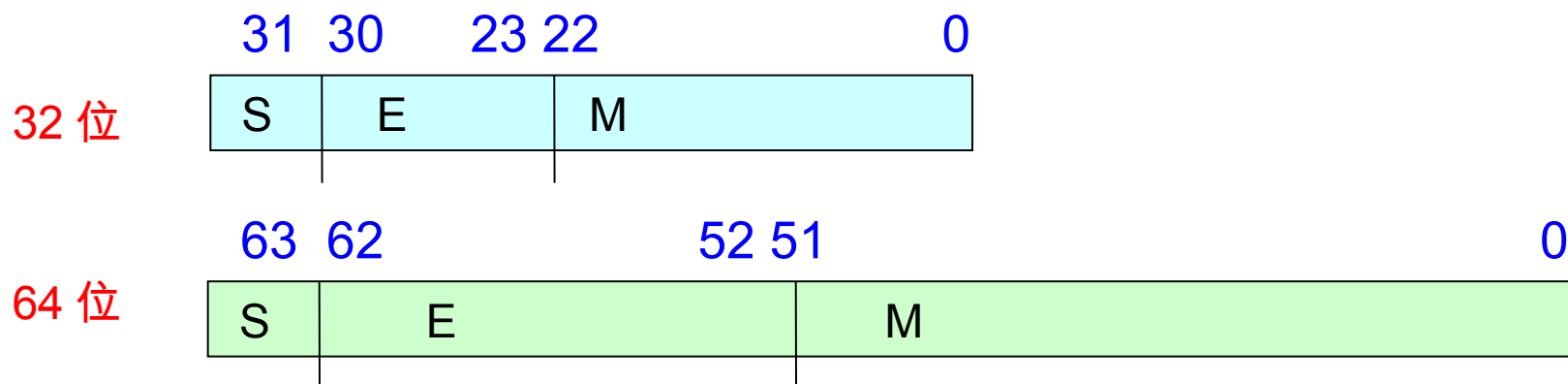
常见的实数类型如表所示：

数据类型	说明	取值范围
float	32 位单精度实数	1.5×10^{-45} - 3.4×10^{38}
double	64 位双精度实数	5.0×10^{-324} - 1.7×10^{308}
decimal	128 位十进制实数	1.0×10^{-28} - 9.9×10^{28}

表中 float 和 double 类型又可以称作浮点型。使用这三种类型的数值时，数字后面需写上 f、(d)、m。

```
float f = 100.4f;  
double d = 23.33;  
decimal m = 100.23m;
```

- 按照 IEEE754 的标准，32 位浮点数和 64 位浮点数的格式为：



S——尾数符号，0 正 1 负；

M——尾数，纯小数表示，小数点在尾数域的最前面。采用原码表示。

E——阶码，阶符采用隐含方式，即采用“移码”方法来表示正负指数。

布尔类型 (bool)：表示布尔逻辑量。它的取值只能是 true 或者 false 。

例如，在应用程序中直接声明并初始化一个 bool 类型的变量，编写如下的代码：

```
bool isParent = true;
```

字符类型 (char)：由所有的 Unicode 字符集组成，这种字符集的特点是一个字符用两个字节来存储，因此每个字符占用 16 位，类似于一个 16 位无符号整数。

2.3.2 值数据类型

2. 结构类型

结构 (struct) 也是一种值类型，通常用来封装一组相关的变量。结构是轻量级的类，使用时不可以使用 new，它可以包含常量、字段、方法、属性和嵌套类型等等。有关结构类型的知识将在第 8 章中进行介绍。

一般情况下很少使用结构，很多人不建议使用。

2.3.2 值数据类型

3. 枚举类型

枚举主要使用 **enum** 关键字来定义，常常用于声明一组指定的常数。关于枚举的知识，在第 8 章会进行详细介绍。

```
class Program
{
    public enum agroup
    {
        monday = 1,
        tuesday = 2,
        Wednesday, sunday = 7
    }

    static void Main(string[] args)
    {
        agroup aday = agroup.monday;
        Console.WriteLine(aday.ToString() );
        Console.ReadLine();
    }
}
```


默认情况下，枚举中的第一个变量被赋值为 0，其他的变量的值按定义的顺序来递增 (0,1,2,3...)，因此以下两个代码定义是等价的：

```
enum TrafficLight
{
    Green,
    Yellow,
    Red
}
```

```
enum TrafficLight
{
    Green = 0,
    Yellow = 1,
    Red = 2
}
```

如果 enum 中的部分成员显式定义了值，而部分没有，那么没有定义值的成员会按照上一个成员的值来递增赋值。

```
class Program
{
    public enum Agroup
    {
        monday = 1,
        tuesday = 2,
        Wednesday, sunday = 7
    }

    static void Main(string[] args)
    {
        string s ;
        s = ((Agroup)3).ToString();
        Console.WriteLine(s);
        Console.WriteLine(agroup.Wednesday);
        Console.WriteLine((agroup)3);

        Console.ReadLine();
    }
}
```

三行相同

2.3.3 引用类型

所有的引用类型都源自 System.Object 家族，它存储的是对值的引用，**两个不同的引用变量指向同一个内存中的物理地址**，引用类型变量的赋值只复制对象的引用，而不复制对象本身。

1. 数组

数组主要是对同一种数据类型的数据进行批量处理。在 C# 中，数组需要初始化才能使用。数组的知识在第 4 章进行详细的介绍。

```
public static void Main(string[] args){
    int[] values = {3,4};
    increase(values);
    Console.WriteLine( values[0]+" "+values[1]);
    Console.Read();
}
```

```
public static void increase ( int[] numbers ){
    for (int n = 0; n < numbers.Length; n++)
    {
        numbers[n] += 1;
        Console.WriteLine("numbers[" + n + "]= " +
numbers[n]);
    }
}
```

values 变了 35

2.3.3 引用类型

2. 字符串

string (字符串) 类型表示零或更多 Unicode 字符组成的序列。在第 9 章中要详细介绍。

例：在应用程序中定义一个方法 StrChange()，该方法传入一个 string 类型的参数，然后在 Main() 方法中声明并初始化一个 string 类型的变量 str，将 str 传入 StrChange() 方法，最后输出字符串的值。

```
static void StrChange(string str)
{
    str = "hello";
}
public static void Main(string[] args)
{
    string str = "123"; // 声明一个字符串
    StrChange(str);    // 调用方法
    Console.WriteLine(str); // 输出字符串
    Console.ReadLine();
}
```

输出结果 ???

运行上面的代码，控制台输出的结果为：123。我们知道：string 类型属于引用类型，而引用类型变量的赋值只复制对象的引用，而不复制对象本身。那么为什么输出的结果是 123 而不是 hello 呢？

特例：string 是引用类型，但它的很多操作是与值类型看齐的，所以如果确实要传地址，那就要加 ref

```
static void StrChange(string str)           // 是值传递
```

```
static void StrChange(ref string str)       // 是引用传递
```

例：修改上例为引用传递

2.3.3 引用类型

3. 类

类 (class) 是抽象的概念，确定对象拥有的特征 (属性) 和行为 (方法)。它可以包含成字段、方法、索引器和构造函数等等。

4. 接口

接口 (Interface) 可以简单的理解为一种约定，用来定义、规范和约束某种行为。接口里面的成员全部都是未实现的，它只包含方法、属性、索引器和事件四种成员。

2.4 运算符与表达式

了解过 C# 的变量和数据类型之后，我们还会遇到它们之间的计算问题。这就需要使用到运算符和表达式。本小节我们会详细介绍它们。

2.4.1 运算符

2.4.2 运算符优先级

2.4.3 表达式

2.4.1 运算符

运算符指明了进行运算的类型，在表达式中用于描述涉及一个或多个操作符的运算。

在 C# 中，根据运算符所需操作的个数，可以分为 3 类：

- 一元运算符 只带有一个操作数。例如 !a 或 a++
- 二元运算符 使用两个操作数。例如 x+y、x/z 或 x%y
- 三元运算符 带有 3 个操作数。三元运算符只有一个 (?:)，用于组成条件表达式

根据运算符执行的操作类型，主要分为：

- 算术运算符
- 比较运算符
- 条件运算符
- 赋值运算符
- 逻辑运算符
- 特殊运算符

2.4.1 运算符

1. 算数运算符

算术运算符就是进行算术运算的操作符，实现了基本算术运算的功能。在 C# 中，常见的算术运算符如表所示。

运算符	说明	示例
+	加法运算符	a+b
-	减法运算符	a-b
*	乘法运算符	a*b
/	除法运算符	a/b
%	求余运算符	a%b
++	两种情况	a++, ++a
--	两种情况	a--, --a

```
float a, b, c, d;
```

```
a = 2;
```

```
b = 5;
```

```
c = a / b;
```

```
Console.WriteLine(c);
```

```
d = a % b;
```

```
Console.WriteLine(d);
```

```
int a , b , c ;
```

```
a = 3 ;
```

```
b = a ++ ;
```

```
Console.WriteLine("{0},{1}",a,b);
```

```
c = ++a ;
```

```
Console.WriteLine("{0},{1}",a,c);
```

```
int a , b , c ;
```

```
a = 3;
```

```
b = a -- ;
```

```
Console.WriteLine("{0},{1}",a,b);
```

```
c = --a ; Console.WriteLine("{0},  
{1}",a,c);
```

2.4.1 运算符

2. 比较运算符

比较运算符又可以称为**关系运算符**。它通常用来比较两个对象，并且返回一个**布尔值**。在 C# 中，常见的比较运算符如表 5 所示。

运算符	说明	示例	结果
>	大于	10>12	结果为 false
<	小于	10<12	结果为 true
==	等于	10==12	结果为 false
!=	不等于	10!=12	结果为 true
>=	大于等于	10>=12	结果为 false
<=	小于等于	10<=12	结果为 true

```
string name1 = “张三”, name2 = “李四”;  
if ( name1 == name2 )  
{  
    // .....  
}  
else  
{  
    // .....  
}
```

2.4.1 运算符

3. 赋值运算符

赋值运算符就是将右边操作数的值赋值给左边的操作数。

运算符	说明	示例	示例含义
=	等于赋值	a = 12	a = 12
+=	加法赋值	a += b	a = a + b
-=	减法赋值	a -= b	a = a - b
*=	乘法赋值	a *= b	a = a * b
/=	除法赋值	a /= b	a = a / b
%=	求余赋值	a %= b	a = a % b

```
int  num1, num2 , num3 ;  
num1 = num2 = num3 =  
10;  
num1 += 5 ;  
num2 -= 5 ;  
num3 /= 5 ;
```

运算符	说明	示例	示例含义
<<=	左移赋值	a<<=b	a = a<<b , a 是被移位数 , b 是移位量。左移一位相当于乘 2
>>=	右移赋值	a>>=b	a = a>>b
&=	AND 位操作赋值	a&=b	a = a&b , a 和 b 按位进行与运算
^=	XOR 位操作赋值	a^=b	a = a^b
=	OR 位操作赋值	a = b	a = a b

位左移运算将整个数按位左移若干位，左移后空出的部分 0。比如 01100101 左移 3 位的结果是 00101000。

位右移运算将整个数按位右移若干位，右移后空出的部分填 0。比如：8 位的 byte 型变量 Byte a=0x65(既 (二进制的 01100101)) 将其右移 3 位：a>>3 的结果是 0x0c(二进制 00001100)。0x 前缀代表 16 进制数

```
public static void Main(string[] args) {
    uint a = 71, b = 115;
    uint c;
    Console.WriteLine(" a={0} , 其二进制形式为 : {1}", a, Convert.ToString(a, 2));
    Console.WriteLine("b={0} , 其二进制形式为 : {1}", b, Convert.ToString(b, 2));
    Console.WriteLine(" 依次计算 : a 左移 2 位 , b 右移 1 位 , a&b , a|b");
    c = a << 2;
    WaitShow(c);    // 输出 a 左移 2 位
    c = b >> 1;
    WaitShow(c);    // b 右移 1 位
    c = a & b;
    WaitShow(c);    // a&b
    c = a | b;
    WaitShow(c);    // a|b
    Console.ReadLine();
}
```

```
public static void WaitShow(uint i){
    Console.ReadLine();
    Console.WriteLine(Convert.ToString(i, 2));    // 把 i 转换成 2 进制的字符串
}
```

2.4.1 运算符

4. 逻辑运算符

&&、&、^、! || 以及| 都被称为逻辑运算符或逻辑操作符，用逻辑运算符把运算对象连接起来符合 C# 语法的式子称为逻辑表达式。逻辑运算符的运算结果可以用逻辑运算的真值来表示，真值表：

a	b	a&&b 或 a&b	a b 或 a b	!a	a^b
false	false	false	false	true	false
false	true	false	true	true	true
true	false	false	true	false	true
true	True	true	true	false	false


```
public static void Main(string[] args)
{
    string psw = Console.ReadLine();
    if ( !(psw.Contains("666") ) & (psw.Length>4) )    // psw 的条件
        Console.WriteLine(" 正确 ! ");
    else
        Console.WriteLine(" 不对 ! ");

    Console.ReadLine();
}
```

```
static void Main(string[] args)
{
    int i=23, j=58;
    if ( ( 24==i++ ) & ( 59==j++ ) )
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
    // Console.WriteLine("{0}, {1}",i,j);

    Console.ReadKey();
}
```

- 1、上面程序运行结果是 Yes 还是 No ？
- 2、把注释打开，执行后看 i、j 的值
- 3、把 & 改成 && ，是否有变化？

& 与 && | 与 ||

位操作：&，按位与

逻辑运算：相同的是：& 和 && 都表示“并且”；

不同的是：& 需要计算并判断前后两个操作数；而 && 遇到第一个操作数 false 时就不处理第二个数。“**短路运算**”

| 和 ||：同理

2.4.1 运算符

5. 条件运算符

?: 被称作条件运算符，它是 C# 语言中唯一的一个三元运算符。

如果表达式的值为真，则运算结果为 atrue；否则为 bfalse。

表达式 ? atrue : bfalse ;

```
public static void Main(string[] args)
{
    float ticketPrice;
    string ticketType = Console.ReadLine();
    ticketPrice = ( ticketType == " 经济 ") ? 500 :
1000;
    Console.WriteLine(ticketPrice);

    Console.ReadLine();
}
```

```
string s1, s2;  
s1 = Console.ReadLine();  
s2 = s1 == "yes" ? "you said yes " : "you said no ! ";  
Console.WriteLine(s2);
```

2.4.1 运算符

6. 特殊运算符

运算符	说明	结果
;	标点运算符	用于结束每条 C# 语句
,		将多个命令放在一行
()		强制改变执行的顺序
{ }		代码片段分组
sizeof	SizeOf 运算符	用于确定值的长度 (字节数)
typeof	类运算符	获取某个类型的 System.Type 对象
is		检测运行时对象的类型是否和某个给定的类型相同
as		用于兼容类型之间的转换
new	New 运算符	在堆栈上创建对象并调用构造函数
>>	移位运算符	移位向右移动
<<		移位向左移动

is 检查一个对象是否属于指定的类型，并返回一个 Boolean 值：true 或者 false。注意，is 操作符不会抛出异常。

```
class Program
{
    public class ClassA
    {
    }

    static void Main(string[] args)
    {
        ClassA a = new ClassA();
        if (a is ClassA)
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");

        Console.ReadKey();
    }
}
```

表达式	结果 (int 类型)
sizeof(sbyte)	1
sizeof(byte)	1
sizeof(short)	2
sizeof(ushort)	2
sizeof(int)	4
sizeof(uint)	4
sizeof(long)	8
sizeof(ulong)	8
sizeof(char)	2 (Unicode)
sizeof(float)	4
sizeof(double)	8
sizeof(bool)	1


```
static void Main()
{
    Console.WriteLine("The size of short is {0}.", sizeof(short));
    Console.WriteLine("The size of int is {0}.", sizeof(int));
    Console.WriteLine("The size of long is {0}.", sizeof(long));
}
```

2.4.2 运算符的优先级

优先级	类型	运算符
1	初级运算符	<code>a++</code> 、 <code>a--</code> 、 <code>()</code> 、 <code>[]</code> 、 <code>new</code> 、 <code>typeof</code> 、 <code>checked</code> 、 <code>unchecked</code>
2	一元运算符	<code>+(如 +a)</code> 、 <code>-(如 -a)</code> 、 <code>!(如 !a)</code> 、 <code>++a</code> 、 <code>--a</code> 和强制类型转换
3	乘除模运算符	<code>*</code> 、 <code>/</code> 、 <code>%</code>
4	加减运算符	<code>+(如 a+b)</code> 、 <code>-(如 a-b)</code>
5	移位运算符	<code><<</code> 、 <code>>></code>
6	比较和类型运算符	<code><</code> 、 <code>></code> 、 <code><=</code> 、 <code>>=</code> 、 <code>is(如 x is int)</code> 、 <code>as (如 x as int)</code>
7	等性比较运算符	<code>==</code> 、 <code>!=</code>
8	逻辑与运算符 (按位 AND)	<code>&</code>
9	逻辑异或运算符 (按位 XOR)	<code>^</code>
10	逻辑或运算符 (按位 OR)	<code> </code>
11	条件与运算符 (布尔 AND)	<code>&&</code>

$x = a + b + c;$

$y = a + b * c - (a / 2 \% c);$

$\text{bool } k = s1 == s2 \ \&\& \ b \% a \neq 0 ;$

$c = a++---b;$

checked 和 unchecked

checked 和 unchecked 关键字用来限定检查或不检查数学运算溢出；如果使用了 checked 发生数学运算溢出时会抛出 `OverflowException`；如果 unchecked 则不检查溢出，错了也不会报。

1. 一段编译没通过的代码

```
int a = int.MaxValue * 2;
```

以上代码段编译没有通过，在 VS 中会有一条红色的波浪线指出这段代码有问题：“The operation overflows at compile time in checked mode”。编译器会在编译时检查数学运算是否溢出，但是仅限于使用常量的运算。2 中的代码，编译器就报不出错误来了。

2. 一段编译通过但是不能得到正确结果的代码

```
int temp = int.MaxValue;  
int a = temp * 2;  
Console.Write(a);
```

这段代码可以正常执行，结果输出 -2。这说明在运行时 cpu 只管算，没有检查算术运算是否溢出。执行了，而结果是错误的，非常危险。如何避免这种危险呢？请看 3

3. 使用 checked 关键字，溢出时报警

```
int temp = int.MaxValue;  
try  
{  
    int a = checked(temp * 2);  
    Console.WriteLine(a);  
}  
catch (OverflowException)  
{  
    Console.WriteLine(" 溢出了 !");  
}
```

使用 checked 关键字修饰 temp*2 的计算结果，并使用 try catch 在发生溢出时做处理。以上代码将输出：“溢出了”

如果一段代码中有很多算术运算都需要做溢出检查，那会用很多 checked 修饰的表达式吗？请看 4

4. checked 关键字可以修饰一个语句块，请看下面代码

```
int temp = int.MaxValue;
try
{
    checked
    {
        int num = temp / 20;
        int a = temp * 2;
        int c = temp * 1000;
    }
}
catch (OverflowException)
{
    Console.WriteLine(" 溢出了 ! ");
}
```

以上程序输出结果和 3 一样

2.5 数据类型转换

上一节我们详细介绍了 C# 中常用的数据类型，编译器编译的时候需要确切的知道数据类型，正因为类型有明确的区分，所以需要数据类型转换。例如：用户声明了一个 double 类型的变量，想要将 double 类型转化为 int 数据类型，这时就需要进行数据类型转换。

2.5.1 隐式转换

2.5.2 显式转换

2.5.3 装箱和拆箱

2.5.1 隐式转换

简单来说，把取值范围小的类型转换为取值范围大的类型就叫做隐式转换。比如：int 类型可以隐式转换为 long、ulong、double、float 和 decimal 类型；long 类型可以转换为 float、double 和 decimal 类型；float 类型也可以转换为 double 类型等。

数据类型	说明	取值范围
int	有符号 32 位整数	-2 147 483 648-2 147 483 647
long	有符号 64 位整数	-2^{63} - 2^{63}
ulong	无符号 64 位整数	0- 2^{64}
float	32 位单精度实数	$1.5 * 10^{-45}$ - $3.4 * 10^{38}$
double	64 位双精度实数	$5.0 * 10^{-324}$ - $1.7 * 10^{308}$
decimal	128 位十进制实数	$1.0 * .9 * 10^{28}$

```
int count = 10;
long longnum = count;
Console.WriteLine(longnum);
Console.ReadLine();
```


2.5.2 显式转换

和隐式转换相反，把取值范围大的类型转换为取值范围小的类型就叫做**显式转换**，又称为**强制转换**。它不能保证数据的准确性。

进行类型转换时，在转换的类型前面加上“**()**”**转换操作符**。

例：声明并初始化两个 double 类型的学生成绩变量 score1 和 score2，声明 int 类型的变量 avgscore 表示学生的平均成绩。代码如下：

```
double stu1 = 90.5;
double stu2 = 90.7;
double avg = (stu1+stu2)/2;
int avg2 = (int) (stu1+stu2)/2;
```

```
double stu1 = 90.5;
double stu2 = 90.7;
int avg = (Int16) (stu1+stu2)/2;
```

包含了什么转换？

```
double stu1 = 90.5;
double stu2 = 90.7;
int avg = (Int64) (stu1+stu2)/2;
```

编译错误

2.5.2 显式转换

隐式转换和显式转换一般都应用在值数据类型之间，如果想要将字符串转换为 int 类型或 double 类型，可以使用 `parse()` 方法或使用 `Convert` 类。

`Convert` 类能够在各种基本类型之间相互转换，其常用的方法：

方法	说明
<code>ToInt32()</code>	转换为整型（int 类型）
<code>ToSingle()</code>	转换为单精度浮点型（float 类型）
<code>ToDouble()</code>	转换为双精度浮点型（double 类型）
<code>ToDecimal()</code>	转换为十进制实数（decimal 类型）
<code>ToString()</code>	转换为字符串类型（string 类型）
... ..	

例：

```
static void Main()
{
    string stringType = "12345";
    int intType = Int32.Parse(stringType);
    intType = Convert.ToInt32(stringType); // 与上句等效
    string s = Convert.ToString(intType); // Convert 有很多用法
    Console.WriteLine(intType);

    Console.ReadLine();
}
```

❑ Convert.ToInt32 (变量)

❑ int.Parse(string 类型变量)

对比如下：

◆ Convert.ToInt32 与 int.Parse 功能类似。

◆ Convert.ToInt32 将指定的值转换为 32 位带正负号的整数，可以转换的类型较多 (Char, Decimal, DateTime, Byte, Double, Int16, Int32, Sbyte, String,.....)

◆ Parse 只能转换**数字类型**的字符串，否则抛出异常。

◆ 与 int.parse() 类似： Int32.parse, Int64.parse, double.parse, float.parse , short.parse,

Convert 类 :

[参考资料](#)

显式数值转换表

源	目标
sbyte	byte 、 ushort 、 uint 、 ulong 或 char
byte	sbyte 或 char
short	sbyte 、 byte 、 ushort 、 uint 、 ulong 或 char
ushort	sbyte 、 byte 、 short 或 char
int	sbyte 、 byte 、 short 、 ushort 、 uint 、 ulong 或 char
uint	sbyte 、 byte 、 short 、 ushort 、 int 或 char
long	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 ulong 或 char
ulong	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 long 或 char
char	sbyte 、 byte 或 short
float	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 long 、 ulong 、 char 或 decimal
double	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 long 、 ulong 、 char 、 float 或 decimal

2.6 注释

在 C# 中，提供了 4 种注释类型：单行注释使用 `//`，代码块注释使用 `/* */`，文档注释使用 `///`。还有一种折叠注释 `#region`，可以将代码折叠。

```
class Program
{
    public static string name = “张三丰”; // 声明姓名变量
    ///<summary>
    /// SayHello() 方法：向某个打招呼
    ///</summary>
    public static string SayHello()
    {
        return name + “，你好，很高兴认识你！我是 Jack”;
    }
    public static void Main(string[] args)
    {
        /*
        * 从这里编写输出代码
        */
        Console.WriteLine(SayHello()); // 调用 SayHello() 方法，向控制台输出一句话
        Console.ReadLine(); // 使输出显示暂停
    }
}
```

#region 名称
一段可折叠的内容
#endregion

```
static void Main()
{
    #region defs
    string stringType = "12345";
    int intType = Int32.Parse(stringType);
    intType = Convert.ToInt32(stringType); // 与上句等效
    string s = Convert.ToString(intType); // Convert 有很多用法
    #endregion

    Console.WriteLine(s);

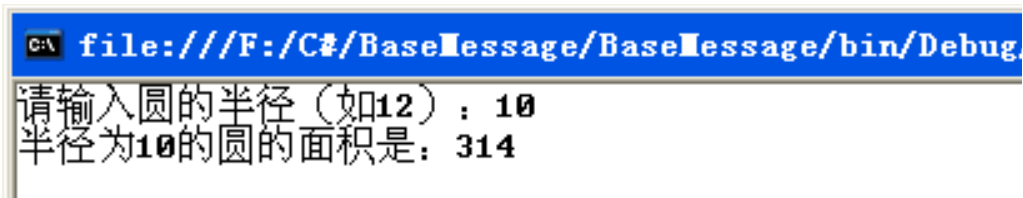
    Console.ReadLine();
}
```


例：求圆的面积

公式：圆的面积 = $\pi * R * R$ 。本小节就在程序中求出圆的面积。

在应用程序中声明两个变量 `area` 和 `radius`，分别表示圆的面积和半径。
声明常量 `Pi` 表示圆周率 π 。

```
public static void Main(string[] args)
{
    double area, radius;
    const double Pi = 3.14d;
    Console.Write(" 请输入圆的半径 ( 如 12 ) : ");
    radius = double.Parse(Console.ReadLine());
    area = Pi * radius * radius;
    Console.WriteLine(" 半径为 {0} 的圆的面积是 : {1}", radius, area);
    Console.ReadLine();
}
```



```
C:\ file:///F:/C#/BaseMessage/BaseMessage/bin/Debug
请输入圆的半径 (如12) : 10
半径为10的圆的面积是: 314
```

例：把整数转换为二进制数

```
static string ConvertIntToBinary (int n)  // n 是待转换的数
{
    string binary = string.Empty; // 保存二进制的字符串，初始为空
    int i = n;
    int m = 0;
    while (i > 1)
    {
        m = n % 2;    // 结果就是：n 的最右边一位的值
        i = n / 2;     // 结果就是：n 去掉最右边一位后的左边部分
        binary = m.ToString() + binary; // 添加到 binary 字符串中
        n = i;
    } // 循环结束后，i 或者为 1 或者为 0 （取决于 n 的高位）
    if (i > 0) binary = "1" + binary;
    return binary;
}

public static void Main(string[] args)
{ Console.WriteLine(ConvertIntToBinary(int.Parse(Console.ReadLine())));
  Console.ReadLine();
}
```