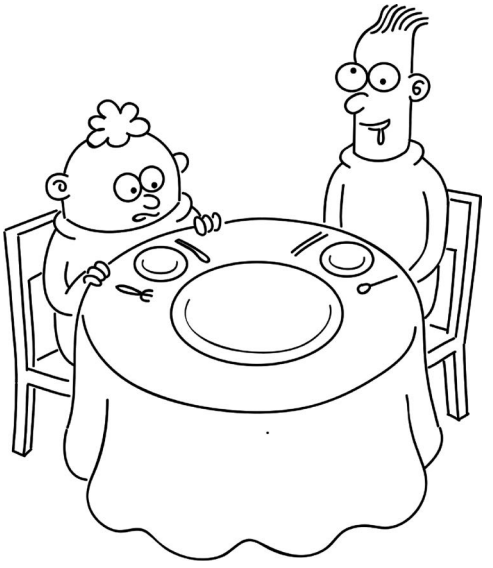
第 3 章

C＃语言数据类型



在声明变量或常量时会用到数据类型，本章介绍 C＃语言数据类型。

C＃语言的数据类型分为：

(1) 值类型。

(2) 引用类型。

3.1 值类型

值类型直接存储其值，而引用类型存储对值的引用，与其他语言相比，C＃语言中的值类似于 Java 语言中的简单类型(整数类型、浮点类型等)。

值类型的变量总是包含该类型的值，值类型的值不可能为 null。

例如，int(整数)类型是值类型，那么执行 int x = 100；语句时，如图 3-1 所示，系统为变量 x 分配内存空间(假设变量 x 内存地址为 0x61ff08)。

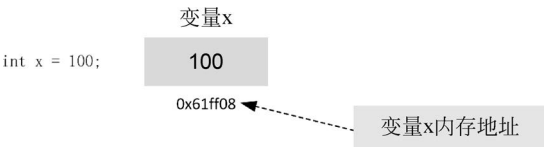


图 3-1　值类型

C#语言中内置的值类型包括如下几种：

(1) 整数类型；

(2) 浮点类型；

(3) 字符类型；

(4) 布尔类型。

这些数据类型也称为“简单类型”，此外，值类型还有结构类型(struct)和枚举类型(enum)，有关结构类型和枚举类型将在后面详细介绍。

3.1.1　整数类型

C#语言中整数类型包括 sbyte、short、int、long、byte、ushort、uint、ulong，它们之间的区别如表 3-1 所示。

表 3-1　整数类型

整数类型	大　小	取值范围	后　缀
sbyte	8 位带符号整数	−128～127	
byte	无符号 8 位整数	0～255	
short	有符号 16 位整数	−32 768～32 767	
ushort	无符号 16 位整数	0～65 535	
int	带符号 32 位整数	−2 147 483 648～2 147 483 647	
uint	无符号 32 位整数	0～4 294 967 295	U 或 u
long	64 位带符号整数	−9 223 372 036 854 775 808～9 223 372 036 854 775 807	L 或 l
ulong	无符号 64 位整数	0～18 446 744 073 709 551 615	UL 或 ul
nint	带符号 32 位或 64 位整数	取决于(在运行时计算的)平台	
nuint	无符号 32 位或 64 位整数	取决于(在运行时计算的)平台	

C#语言的整数类型默认是 int 类型，例如 10 表示为 int 类型整数 10，而不是 short 或 byte 类型；而 10L(或 10l)表示 long 类型的整数 10，就是在 10 后面加上 l(小写英文字母)或 L(大写英文字母)，则类型为 ulong 的整数，类似的后缀还有 UL、Ul、uL、ul、LU、Lu、lU 或 lu。

另外，整数类型常量还可以使用二进制数、八进制数和十六进制数表示，它们的表示方式分别如下。

(1) 二进制数：以 0b 或 0B 为前缀，注意 0 是阿拉伯数字，不要误认为是英文字母 o。

(2) 八进制数：以 0 为前缀，注意 0 是阿拉伯数字。

(3) 十六进制数：以 0x 或 0X 为前缀，注意 0 是阿拉伯数字。

使用整数类型示例代码如下：

```
//3.1.1 整数类型
using System;
namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {

            byte myNum1 = 128;                  // 编译错误
            byte myNum2 = 125;
            short myNum3 = 5000;
            int myNum4 = 5000;
            long myNum5 = 10L;                  // 声明 long 类型变量
            long myNum6 = 10l;                  // 声明 long 类型变量
            ulong myNum7 = 10UL;                // 声明 ulong 类型变量

            int decimalInt = 10;
            byte binaryInt1 = 0b1010;
            short binaryInt2 = 0B11100;
            long octalInt = 012;
            byte hexadecimalInt = 0xA;
        }
    }
}
```

提示 在程序代码中，尽量不用小写英文字母 l，因为它容易与数值 1 混淆，特别是在C#语言中表示 long 类型整数时很少使用小写英文字母 l，而是使用大写的英文字母 L。例如，10L 要比 10l 可读性更好。

微课视频

3.1.2 浮点类型

浮点类型主要用来存储小数数值，C#语言浮点类型为：①单精度浮点(float)；②双精度浮点(double)，它们的区别是占用内存空间不同。浮点类型说明如表 3-2 所示。

表 3-2 浮点类型

浮点类型	大致范围	精度	大小
float	$\pm 1.5\times10^{-45}\sim\pm3.4\times10^{38}$	6～9 位数字	4 字节
double	$\pm 5.0\times10^{-324}\sim\pm1.7\times10^{308}$	15～17 位数字	8 字节
decimal	$\pm 1.0\times10^{-28}\sim\pm7.9228\times10^{28}$	28～29 位	16 字节

C#语言的浮点类型默认是 double 类型，例如 0.0 表示 double 类型常量，而不是 float 类型。如果想要表示 float 类型，则需要在数值后面加 f 或 F，如果想要明确指定为 double 类型可以在数值后面加上 d 或 D。

另外，浮点数据可以使用小数表示，也可以使用科学记数法表示，科学记数法中使用大写或小写的 e 表示 10 的指数，如 e2 表示 10^2。

浮点类型示例代码如下：

```
//3.1.2 浮点类型
using System;
namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            float float1 = 0.0f;                    // 数值后加 f 表示 float 类型
            float float2 = 2F;                      //数值后加 F 也表示 float 类型
            double float3 = 2.1543276e2;            // 科学记数法表示浮点数
            double float4 = 2.1543276e-2;           // 科学记数法表示浮点数
            double double1 = 0.0;                   // 0.0 默认是 double 类型
            double double2 = 0.0d;                  // 数值后加 d 表示 double 类型
            double double3 = 0.0D;                  // 数值后加 D 表示 double 类型
        }
    }
}
```

3.1.3　字符类型

字符类型表示单个字符，C#语言中 char 声明字符类型，C#语言中的字符常量必须包裹在单引号(')中。

C#语言字符类型采用 Unicode UTF-16 编码，占 2 字节(16 位)，因而可用十六进制(无符号的)编码形式表示，它们的表现形式是\un，其中 n 为 16 位十六进制数，所以字符'B'也可以用 Unicode 编码'\u0042'表示。

示例代码如下：

```
//3.1.3 字符类型
using System;
namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            char letter1 = 'B';
            char letter2 = '\u0042';                // Unicode 编码表示的字符
            char letter3 = '斯';                    // 汉字字符
            int letter4 = 65;
            int int1 = 'B' + 3;                     // 字符类型可以进行数学计算

            Console.WriteLine("letter1:" + letter1);
```

```
            Console.WriteLine("letter2:" + letter2);
            Console.WriteLine("letter3:" + letter3);
            Console.WriteLine("letter4:" + (char)letter4);                    ①
            Console.WriteLine("int1:" + int1);                                ②
        }
    }
}
```

上述示例代码运行结果如下：

```
letter1:B
letter2:B
letter3:斯
letter4:A
int1:69
```

从运行结果可见，其中变量 letter1 和 letter2 保存了相同的字符，上述代码第①行中(char)letter4 是将整数类型转换为字符类型，这样在打印输出时可以得到字符 A，而不是字符 A 对应的编码，因此代码第②行输出的是编码 69。

提示 字符类型虽然表示单个字符，但它也可以作为数值类型使用，可以与 int 等数值类型进行数学计算或相互转换。这是因为字符类型在计算机中保存的是 Unicode 编码，双字节 Unicode 编码的存储范围为\u0000～\uFFFF，所以字符类型取值范围为 $0 \sim 2^{16}-1$。Unicode 编码可以表示各种字符，包括中文等非 ASCII 字符。

3.2 类型转换

学习了数据类型后，读者可能会思考一个问题，数据类型之间是否可以转换呢？数据类型的转换情况比较复杂，在互相兼容的数据类型(例如：简单类型)之间可以互相转换，一般分为两种转换方式。

(1) 隐式类型转换，也称为自动类型转换；

(2) 显式类型转换，也称为强制类型转换。

微课视频

3.2.1 隐式类型转换

隐式类型转换可以安全地实现转换，不需要采取其他手段，总的原则是小范围数据类型可以自动转换为大范围数据类型，所以如下的转换是自动的：

sbyte→short→int→long→float→double

此外，char 类型比较特殊，char 类型自动转换为 int、long、float 和 double 类型，但 sbyte 类型或 short 类型不能隐式转换为 char 类型，而且 char 类型也不能隐式转换为 sbyte 类型或 short 类型。

隐式类型转换不仅发生在赋值过程中，在进行数学计算时也会发生，在计算中往往是先

将数据类型转换为同一类型，然后再进行计算，计算规则如表 3-3 所示。

表 3-3　计算规则

操作数 1 类型	操作数 2 类型	转换后的类型
sbyte、short、char	int	int
sbyte、short、char、int	long	long
sbyte、short、char、int、long	float	float
sbyte、short、char、int、long、float	double	double

示例代码如下：

```
//3.2.1 隐式类型转换
using System;
namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int decimalInt = 10;
            byte byteInt = 0b1010;
            short shortInt = byteInt;                          // byte 类型转换为 short 类型
            Console.WriteLine(shortInt.GetType().Name);        // 打印 Int16                    ①
            long longInt = shortInt;                           // short 类型转换为 long 类型
            Console.WriteLine(longInt.GetType().Name);         // 打印 Int64

            char charNum = 'C';
            decimalInt = charNum;                              // char 类型转换为 int 类型
            Console.WriteLine(decimalInt.GetType().Name);      // 打印 Int32
            float floatNum = longInt;                          // long 类型转换为 float 类型
            Console.WriteLine(floatNum.GetType().Name);        // 打印 Single
            double doubleNum = floatNum;                       // float 类型转换为 double 类型
            Console.WriteLine(doubleNum.GetType().Name);       // 打印 Double

            //表达式计算后类型是 double
            double res = floatNum * floatNum + doubleNum / charNum;
            Console.WriteLine(res.GetType().Name);             // 打印 Double
        }
    }

}
```

上述代码第①行 GetType()方法可以获得某个数值的类型，GetType().Name 可以获得该类型的名称，它采用.NET 类型的别名表示，其他代码此处不再赘述。

上述示例代码运行结果如下：

```
Int16
Int64
Int32
```

```
Single
Double
Double
```

微课视频

3.2.2 显式类型转换

隐式类型转换的相反操作是显式类型转换，显式类型转换是在变量或常量之前加上“(目标类型)”实现。

示例代码如下：

```
//3.2.2 显式类型转换
using System;
namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int decimalInt = 10;
            byte byteInt = (byte)decimalInt; // 将 int 类型强制转换为 byte 类型
            double float1 = 2.1543276e2;
            long longInt = (long)float1;      //将 float 类型强制转换为 long 类型
            Console.WriteLine(longInt);       //输出 215,小数部分被截掉
            float float2 = (float)float1;     //将 double 类型强制转换为 float 类型

            long long1 = 999999999999L;
            int int1 = (int)long1;            // 将 long 类型强制转换为 int 类型,精度丢失 ①
            Console.WriteLine(int1);          // - 727379969
        }

    }
}
```

上述示例代码运行结果如下：

```
215
- 727379969
```

上述代码第①行在运行强制类型转换时，发生了数据精度丢失的问题，这是因为 long1 变量太大，当取值范围大的数值转换为取值范围小的数值时，大取值范围数值的高位被截掉，这样就会导致数据精度丢失。

微课视频

3.2.3 类型转换方法

对于兼容的数据类型可以进行隐式类型转换或显式类型转换，而对于不兼容数据类型(例如：字符串类型转换为 int 类型等)可以使用 System.Convert 类的相应方法进行转换，方法主要有以下几种。

(1) Convert.ToBoolean(x)：将参数 x 转换为布尔类型。

(2) Convert.ToDouble(x)：将参数 x 转换为 double 类型。

(3) Convert.ToString(x)：将参数 x 转换为字符串类型。

(4) Convert.ToInt32(x)：将参数 x 转换为 int 类型。

(5) Convert.ToInt64(x)：将参数 x 转换为 long 类型。

示例代码如下：

```
//3.2.3 类型转换方法

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int myInt = 10;
            double myDouble = 5.25;
            bool myBool = true;

            Console.WriteLine(Convert.ToString(myInt)); //将 int 类型数据转换为字符串类型
            Console.WriteLine(Convert.ToDouble(myInt)); //将 int 类型数据转换为 double 类型
            Console.WriteLine(Convert.ToInt32(myDouble)); //将 double 类型数据转换为 int
                                                          //类型  ①
            Console.WriteLine(Convert.ToString(myBool)); //将布尔类型数据转换为字符串
                                                         //类型
        }
    }
}
```

注意上述代码第①行将 double 类型数据转换为 int 类型数据时，小数部分会被截掉，上述示例代码运行结果如下：

```
10
10
5
True
```

3.3　引用类型

引用类型的变量又称为对象，可存储对实际数据的引用。例如 string(字符串)是引用类型，那么执行 string name="Ben"；语句时，如图 3-2 所示，系统在为“Ben”字符串分配内存空间，假设内存地址为 0x61ff08，而变量 name 是引用类型，它保存了 Ben 字符串的内存地址，需要注意的是，引用类型变量 name 也有自己的内存地址，假设其为 0x61ff09。

引用类型数据包括如下几种：

(1) 类(class)；

(2) 接口(interface)；

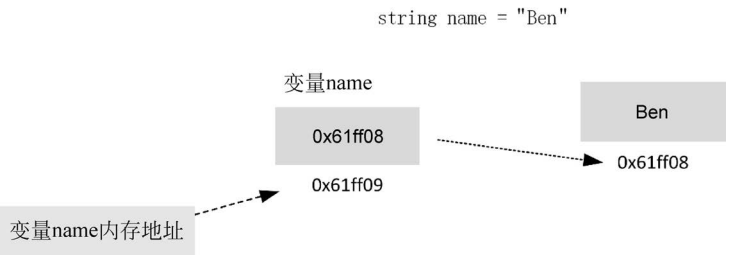


图 3-2　引用类型

(3) 委托(delegate)；

(4) 字符串(string)；

(5) 数组。

这几种类型将在以后的学习中介绍。

微课视频

3.4　装箱和拆箱

C#语言中任何值类型数据都可以转换为 object(对象)类型，它是所有对象的根类，在C#语言中所有类都直接或间接继承 object 类型，object 都是引用类型。这种将值类型数据转换为 object 类型数据就是装箱(boxing)，反之称为拆箱(unboxing)。

示例代码如下：

```
//3.4 装箱和拆箱

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int i = 789;                    //声明整数类型变量 i
            object obj1 = (object)i;        //装箱                                    ①
            object obj2 = i;                //自动装箱                                ②

            int in12 = (int)obj2;           //拆箱                                    ③
            short int2 = (short)obj2;       //拆箱失败,抛出异常 InvalidCastException  ④
        }
    }
}
```

上述代码第①行将值类型变量 i 装箱，这个过程事实上非常复杂，它们在内存中存放的形式发生了很大的变化。装箱过程可以自动进行，代码第②行是自动装箱。

代码第③行是拆箱过程，这需要显式指定拆箱的数据类型，拆箱的数据类型要与保存的值类型一致，否则会发生异常，见代码第④行。

上述代码运行结果此处不再赘述，读者自己尝试一下。

3.5　可空类型

默认情况下，所有的数据类型都是非空类型(Non-Null)，声明的变量都是不能接收空值(null)的，但是有时变量确实有可能为空值，因此C#语言声明变量时，还可以将变量指定为可空类型(Nullable)。

3.5.1　可空类型概念

微课视频

非空类型设计能够有效防止空指针异常，空指针异常引起的原因是在试图调用一个空对象的方法或字段等成员时，会抛出空指针异常。在C#语言中可以将一个变量声明为非空类型，那么它就永远不会接收空值，否则会发生编译错误。示例代码如下：

```
//3.5.1 可空类型概念

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int n1 = 10;                    //声明 int 类型变量 n1                    ①
            n1 = null;                      //发生编译错误                            ②

            int? n2 = 10;                   //声明 int?类型变量 n2,它是 int 的可空类型 ③
            n2 = null;
            Nullable<double> d1 = 10.8;     //另一种声明可空类型方式                  ④
        }
    }
}
```

上述代码第①行是声明变量int类型变量n1，它是不能接收空值的，所以代码第②行会发生编译错误。

代码第③行声明变量n2，它是int? 类型，注意带有问号(?)说明int是可空类型，它是可以接收空值的。

代码第④行是采用另一种声明可空类型方式，Nullable<double>声明double为可空类型。

3.5.2　访问可空数值

微课视频

访问可空变量数值时，可以使用GetValueOrDefault()方法，如果数值非空，直接返回数值；如果数值为空，则返回该数据类型的默认值。

使用GetValueOrDefault()方法的示例代码如下：

```
//3.5.2 访问可空数值

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // 分配空值给变量 x
            Nullable<int> x = null;

            // 分配空值给变量 y
            bool? y = null;

            // 分配非空值给变量 z
            float? z = 10.9f;


            Console.WriteLine("Value of x: " + x.GetValueOrDefault());
            Console.WriteLine("Value of y: " + y.GetValueOrDefault());
            Console.WriteLine("Value of z: " + z.GetValueOrDefault());
        }
    }
}
```

上述示例代码运行结果如下：

```
Value of x: 0
Value of y: False
Value of z: 10.9
```

微课视频

3.5.3 合并操作符

可空值变量不能直接赋值给非空类型变量，所以如下代码会发生编译错误。

```
int? a = 10;
int b = a;              // 编译错误
Console.WriteLine(b);
```

那么如何解决这个问题？C#语言提供了合并操作符(??)，合并操作符语法格式如下：

表达式 1??表达式 2

如果“表达式 1”为非空，则返回“表达式 1”的结果；否则返回“表达式 2”结果。

示例代码如下：

```
//3.5.3 合并操作符

namespace HelloProj
{
```

```
    internal class Program
    {
        static void Main(string[] args)
        {
            int? a = 10;
            // int b = a;                    // 编译错误

            int b = a ?? 0;                  // 如果 a 为非空,返回 a,否则返回 0      ①
            Console.WriteLine("b = " + b);
        }
    }
}
```

上述代码第①行中使用了合并操作符(??),上述示例运行结果如下:

```
b = 10
```

3.6　字符串类型

由字符组成的一串字符序列,称为“字符串”,在前面的章节中也多次用到了字符串,本节重点介绍。

3.6.1　字符串表示方式

C#语言中普通字符串采用双引号“”包裹起来表示,C#语言字符串的数据类型是string,.NET 提供的别名是 System.String。使用字符串的示例代码如下:

```
//3.6.1 字符串表示方式

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            String s1 = "Hello World";
            String s2 = "\u0048\u0065\u006c\u006c\u006f\u0020\u0057\u006f\u0072\u006c\
u0064";
            String s3 = "世界你好";
            String s4 = "B";                  //"B"表示字符串 B,而不是字符 B
            String s5 = "";                   //空字符串

            Console.WriteLine("s1:" + s1);
            Console.WriteLine("s2:" + s2);
            Console.WriteLine("s3:" + s3);
            Console.WriteLine("s4:" + s4);
            Console.WriteLine("s5:" + s5);
        }
```

```
    }
}
```

上述代码执行结果如下：

```
s1:Hello World
s2:Hello World
s3:世界你好
s4:B
s5:
```

从运行结果可见，s1 和 s2 存储的都是 Hello World 字符串，其中 s2 采用 Unicode 编码表示。需要注意的是，s5 表示的是空字符串，也会在内存中占用空间，只是它的字符串内容为空，字符串长度为 0。

微课视频

3.6.2 转义符

如果想在字符串中包含一些特殊的字符，例如换行符、制表符等，则需要在普通字符串中转义，在其前面加上反斜杠(\)，这称为转义符。表 3-4 所示是常用的几个转义符。

表 3-4 常用的转义符

字符表示	Unicode 编码	说明
\t	\u0009	转义水平制表符
\n	\u000a	转义换行符
\r	\u000d	转义回车符
\"	\u0022	转义双引号
\'	\u0027	转义单引号
\\	\u005c	转义反斜杠

示例代码如下：

```
//3.6.2 转义符

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            String s1 = "\"世界\"你好!";                    // 转义双引号
            String s2 = "\'世界\'你好!";                    // 转义单引号
            String s3 = "Hello\t World";                    // 转义制表符
            String s4 = "Hello\\ World";                    // 转义反斜杠制表符
            String s5 = "Hello\n World";                    // 转义换行符

            Console.WriteLine("s1:" + s1);
            Console.WriteLine("s2:" + s2);
            Console.WriteLine("s3:" + s3);
            Console.WriteLine("s4:" + s4);
```

```
            Console.WriteLine("s5:" + s5);
        }
    }
}
```

上述代码执行结果如下：

```
s1:"世界"你好!
s2:'世界'你好!
s3:Hello       World
s4:Hello\ World
s5:Hello
World
```

3.6.3　逐字字符串

微课视频

如果在一个字符串中有很多特殊的字符需要转义，那么可以使用逐字字符串，逐字字符串是在字符串前面加上“@”，逐字字符串中的反斜杠(\)不会转义字符。

示例代码如下：

```
//3.6.3 逐字字符串

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            String s1 = @"Hello\ World";                    // 反斜杠不需要转义
            String s2 = @"'世界'你好!";                      // 单引号不需要转义
            String s3 = @"Hello\t World";                   // 转义符失效
            String s4 = @"Hello\ World";                    // 转义符失效
            String s5 = @"Hello\n World";                   // 转义换行符失效

            Console.WriteLine("s1:" + s1);
            Console.WriteLine("s2:" + s2);
            Console.WriteLine("s3:" + s3);
            Console.WriteLine("s4:" + s4);
            Console.WriteLine("s5:" + s5);
        }
    }
}
```

上述代码中 s1～s5 字符串都是采用逐字字符串表示的，不需要转义符(\)进行转义，如果加上了转义符，反倒是画蛇添足。

上述代码执行结果如下：

```
s1:Hello\ World
s2:'世界'你好!
s3:Hello\t World
```

```
s4:Hello\ World
s5:Hello\n World
```

3.7 数组类型

在计算机语言中数组是非常重要的数据结构，大部分计算机语言中数组具有如下三个基本特性。

(1) 一致性：数组只能保存相同的数据类型元素，元素的数据类型可以是任何相同的数据类型。

(2) 有序性：数组中的元素是有序的，通过下标访问。

(3) 不可变性：数组一旦初始化，则长度(数组中元素的个数)不可变。

在C#语言中，数组的下标是从0开始的，事实上很多计算机语言的数组下标都是从0开始的。C#语言数组下标访问运算符是中括号，如intArray[0]，表示访问intArray数组的第一个元素，0是第一个元素的下标。

另外，C#语言中的数组本身是引用类型数据，它的长度属性是Length，也可以通过GetLength()方法获得数组长度。

3.7.1 数组声明

数组在使用之前一定要做两件事情：声明和初始化。数组声明完成后，数组的长度还不能确定。

数组声明语法格式如下：

```
元素数据类型[] 数组变量名;
```

元素数据类型可以是C#语言中任意的数据类型，包括值类型和引用类型。

数组声明示例代码如下：

```
int[] intArray;
float[] floatArray;
String[] strArray;
```

注意 C#语言声明一个数组的时候，不能用和C/C++语言一样的方式，即将[]写在变量的后面，例如代码int intArray[]是非法的。

微课视频

3.7.2 数组初始化

声明完成就要对数组进行初始化，数组初始化的过程就是为数组中每个元素分配内存空间，并为每个元素提供初始值。初始化之后，数组的长度就确定下来不能再变化了。

数组初始化可以分为静态初始化和动态初始化。

1. 静态初始化

静态初始化就是将数组的元素放到大括号中，元素之间用逗号(,)分隔。示例代码如下。

```
double[] doubleArry = {2.1, 32, 43, 45};
string[] strArry = {"刘备", "关羽", "张飞"};
```

静态初始化是在已知数组的每个元素内容的情况下使用的。很多情况下数据是从数据库或网络中获得的，在编程时不知道元素有多少，更不知道元素的内容，此时可采用动态初始化。

2. 动态初始化

动态初始化使用 new 运算符分配指定长度的内存空间，语法格式如下：

```
new 元素数据类型[数组长度];
```

示例代码如下：

```
//3.7.2 数组初始化

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // 1.静态初始化
            double[] doubleArry = { 2.1, 3.2, 0.43, 5.45 };
            string[] strArry = { "刘备", "关羽", "张飞" };

            // 2.动态初始化
            int[] intArray2;                    // 声明数组 intArray2
            intArray2 = new int[4];             // 通过 new 运算符分配了 4 个元素的内存空间
            intArray2[0] = 21;
            intArray2[1] = 32;
            intArray2[2] = 43;

            // 动态初始化 String 数组
            string[] strArry2 = new string[3];// 通过 new 运算符分配了 3 个元素的内存空间
                                              // 初始化数组中元素
            strArry2[0] = "刘备";
            strArry2[1] = "关羽";
            strArry2[2] = "张飞";
        }
    }
}
```

3.7.3　多维数组

数组可以具有多个维度，声明创建一个 4 行 2 列的二维数组的示例代码如下：

```
int[,] array = new int[4, 2];
```

声明创建一个三维数组的示例代码如下：

```
int[, ,] array1 = new int[4, 2, 3];
```

也可以在声明数组时将其初始化，如下所示：

```
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
int[, ,] array3D = new int[,,] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };
```

完整的示例代码如下：

```
//3.7.3 多维数组

namespace HelloProj
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // 静态初始化
            string[] strArry = { "刘备", "关羽", "张飞" };

            Console.WriteLine("--------- 遍历一维数组 strArry ----------");
            for (int i = 0; i < strArry.GetLength(0); i++)                    ①
            {
                Console.WriteLine(strArry[i]);
            }

            // 声明创建一个 4 行 2 列的二维数组
            int[,] array = new int[4, 2];


            //静态初始化二维数组 array2D
            int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };

            Console.WriteLine("--------- 遍历二维数组 array2D ----------");
            for (int i = 0; i < array2D.GetLength(0); i++)                    ②
            {
                for (int j = 0; j < array2D.GetLength(1); j++)                ③
                {
                    Console.WriteLine(array2D[i, j]);
                }
            }

            //静态初始化三维数组 array3D
            int[,,] array3D = new int[,,] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };

            Console.WriteLine("--------- 遍历三维数组 array3D ----------");
            for (int i = 0; i < array3D.GetLength(0); i++)
            {
```

```
                for (int j = 0; j < array3D.GetLength(1); j++)
                {
                    for (int k = 0; k < array3D.GetLength(2); k++)
                    {
                        Console.WriteLine(array3D[i, j, k]);
                    }
                }
            }
        }
    }
}
```

上述代码第①行遍历一维数组 strArry，其中 GetLength(0)可以获得数组的长度，也可通过 Length 属性获得数组长度。

代码第②行遍历二维数组 array2D，注意 array2D.GetLength(0)是获得数组第一维度上的长度。

代码第③行中< array2D.GetLength(1)是获得数组第二维度上的长度。

上述代码运行结果如下：

```
--------遍历一维数组 strArry----------
刘备
关羽
张飞
--------遍历二维数组 array2D----------
1
2
3
4
5
6
7
8
--------遍历三维数组 array3D----------
1
2
3
4
5
6
```

3.8　枚举

微课视频

枚举是用户定义的整数类型数据，在声明一个枚举时，要指定枚举包含的一组实际值。

枚举以 enum 关键字声明，默认情况下，枚举的第一个成员的值是 0，然后对每个后续的枚举成员按 1 递增，初始化过程中可重写默认值。

使用枚举表示一周中的每个工作日的示例代码如下：

```
//3.8 枚举
// 声明枚举
enum WeekDays { Monday, Tuesday, Wednesday, Thursday, Friday }      ①


internal class Program
{
    static void Main(string[] args)
    {
        WriteGreeting(WeekDays.Friday);                             ②
    }

    static void WriteGreeting(WeekDays day)                         ③
    {
        switch (day)
        {
            case WeekDays.Monday:
                Console.WriteLine("星期一好!");
                break;
            case WeekDays.Tuesday:
                Console.WriteLine("星期二好!");
                break;
            case WeekDays.Wednesday:
                Console.WriteLine("星期三好!");
                break;
            case WeekDays.Thursday:
                Console.WriteLine("星期四好!");
                break;
            case WeekDays.Friday:
                Console.WriteLine("星期五好!");
                break;
            default:
                Console.WriteLine("大家好!");
                break;
        }
    }
}
```

上述代码第①行声明枚举类型 WeekDays，代码第②行调用自定义的 WriteGreeting() 方法，传递的参数 WeekDays.Friday 是一个枚举值。

代码第③行是自定义方法，在该方法中通过 switch 语句判断枚举值，并打印相关内容，有关 switch 语句将在 5.2 节详细介绍，这里不再赘述。

上述示例代码采用默认的方式声明枚举，这里没有为枚举的成员指定实际整数值，第一个成员实际值是 0。此外，开发人员还可以指定各个成员实际值，重新声明枚举，示例代码如下：

```
enum WeekDays
{
```

```
    Monday = 1,
    Tuesday = 2,
    Wednesday = 3,
    Thursday = 4,
    Friday = 5,
}
```

3.9　动手练一练

选择题

(1) 下面哪些代码在编译时不会弹出警告或错误信息(　　)?

A. float f=1.3;　　B. char c="a";

C. byte b=257;　　D. Boolean b=null;

E. 以上都不是

(2) byte 的取值范围是?(　　)

A. −128～127　　B. −256～256

C. −255～256　　D. 依赖于计算机本身硬件

(3) 下列选项中正确的表达式有哪些?(　　)

A. byte=128;　　B. Boolean=null;

C. long l=0xfffL;　　D. double=0.9239d;

(4) 下列选项中哪些是 C＃语言的基本数据类型?(　　)

A. short　　B. Boolean

C. int　　D. float