第2章

C#语言基础

目标

- 在C#中定义变量和常量
- 使用C#中的基本数据类型
- 理解装箱和拆箱的概念
- 使用C#中的运算符，选择结构和循环结构
- 定义和使用数组，了解结构和枚举
- 了解C#中的预处理指令
- 掌握C#中常用的字符串处理方法

系统定义类型

类型	描 述	范围/精度	例子
object	所有其它类型的最根本的基础类型		object o = null;
string	字符串类型，一个字符串是一个Unicode字符序列		string s= "Hello";
sbyte	8-bit 有符号整数类型	–128...127	sbyte val = 12;
short	16-bit有符号整数类型	–32,768...32,767	short val = 12;
int	32-bit有符号整数类型	–2,147,483,648...2,147,483,647	int val = 12;
long	64-bit有符号整数类型	–9,223,372,036,854,775,808 ...9,223,372,036,854,775,807	long val1 = 12; long val2 = 34L;
byte	8-bit无符号整数类型	0...255	byte val1 = 12; byte val2 = 34U;
ushort	16-bit无符号整数类型	0...65,535	ushort val1 = 12; ushort val2 = 34U;
uint	32-bit无符号整数类型	0...4,294,967,295	uint val1 = 12; uint val2 = 34U;

系统定义类型

类型	描 述	范围/精度	例子

ulong	64-bit无符号整数类型	0...18,446,744,073,709,551,615	ulong val1 = 12;
			ulong val2 = 34U;
			ulong val3 = 56L;
			ulong val4 = 78UL;

float	32-bit单精度浮点数类型	1.5 × 10^{-45} 至 3.4 × 1038，7 位精度	float val = 1.23F;
double	64-bit双精度浮点数类型	5.0 × 10^{-324} 至 1.7 × 10308，15 位精度	double val1 = 1.23; double val2 = 4.56D;
bool	布尔类型类型; 一个布尔类型数据不是真就是假	true,false	bool val1 = true; bool val2 = false;
char	字符类型; 一个字符数据是一个Unicode字符		char val = 'h';
decimal	精确十进制类型，有28个有效位	1.0 × 10^{-28} 至 7.9 × 1028，28 位精度	decimal val = 1.23M;

常量

类型	类别	后缀	示例/允许的值
bool	布尔	无	true或false
int	整数	无	int x = 100
Uint	整数	U或u	uint x = 1000u
long	长整型	L或l	long x = 100000L
ulong	无符号长整型	ul,uL,Ul,UL,lu, Lu,lU或LU	ulong x = 4324ul
float	单精度浮点数类型	F或f	float x = 34.76F
double	双精度浮点数类型	D或d	double x = 763.7245D
decimal	精确十进制类型	M或m	decimal x = 1.544M
char	字符	无	char x = ‘a’
string	字符串	无	string str=“abc”

System.Object方法

名称	说明
Equals	确定两个 Object 实例是否相等。
Finalize	允许 Object 在“垃圾回收”回收 Object 之前尝试释放资源并执行其他清理操作。
GetHashCode	用作特定类型的哈希函数。
GetType	获取当前实例的 Type。
MemberwiseClone	创建当前 Object 的浅拷贝副本。
ReferenceEquals	确定指定的 Object 实例是否是相同的实例。
ToString	返回表示当前 Object 的 String。

- **int:**

```
int iMax = int.MaxValue;
int pVal = int.Parse("100");
short i16 = 50;
int i32 = i16;
i16 = i32;  错误
i16 = (short)i32;
```

- **decimal**

```
decimal iRate = 3.9835M;
int i = (int)iRate;
iRate = decimal.Round(iRate, 2); //四舍五入
iRate = decimal.Remainder(512.0M, 51.0M);
```

- **bool**

```
bool bt = (bool)1; //错误
bt = true;
```

- **char**

```
string pattern = "123abcd?";
bool bt;
bt = char.IsLetter('a');
bt = char.IsLetter(pattern, 3);
bt = char.IsNumber(pattern, 3);
bt = char.IsLower(pattern, 3);
bt = char.IsPunctuation(pattern, 7);
bt = char.IsLetterOrDigit(pattern, 3);
```

- **float,double**

```
float f = 24567.66f;
double d = 124.45;
If(Single.IsNaN(0.0/0) {…}
```

- **使用Parse转换数字字符串**

```
short shParse = Int16.Parse("100");
int iParse = Int32.Parse("100");
long shParse = Int64.Parse("100");
decimal dParse=decimal.Parse("99.99");
float sParse=float.Parse("99.99");
double dParse=double.Parse("99.99");
```

字符串

- 字符串常量

```
string path;
path = @"C:\note.txt";
path = "C:\\note.txt";
```

- 字符串操作
  - 索引字符串中的单个字符

```
string str = "abcd";
char c = str[0];
c = str.ElementAt(0);
```

  - 字符串连接

```
string s1 = "My age = ";
int myAge = 28;
string cat = s1 + myAge;
cat = string.Join(" ", s1, myAge.ToString());
```

- 字符串操作
  - 抽取和定位子串

```
string poem = "In Xanadu did Kubla Khan";
string poemSeg = poem.Substring(10);
poemSeg = poem.Substring(0,9);
int index = poem.IndexOf("I");
index = poem.LastIndexOf("n");
```

  - 比较字符串

```
bool isMatch;
string title = "Ancient Mariner";
isMatch = (title == "ANCIENT AMRINER");
isMatch = (title.ToUpper() == "ANCIENT MARINER");
isMatch = title.Equals("Ancient Mariner");
```

String 常用方法应用

输入一个字符串，输出每个单词，重新用下划线连接输出

- 使用 Split() 方法分割字符串
- 使用 Join() 方法连接字符串

分割字符串：

Split()方法——分割字符串

```
splitStrings = inputString.Split(' ');
```

返回值为
字符串数组

字符串变量名

参数：分隔符（char型）

Join()方法——连接字符串

```
// 将分割后的字符串使用下划线连接在一起
joinString = string.Join("_", splitStrings);
```

返回字符串

静态方法

参数1：连接符
参数2：字符串数组

输入一个字符串，输出每个单词，重新用下划线连接输出。

```
using System;

class Program
{
    static void Main(string[] args)
    {
        string inputString;
        inputString = "I am a student";
        string[] splitStrings = inputString.Split(' ');
        string joinString = string.Join("_", splitStrings);
        Console.WriteLine(joinString);
    }
}
```

输入一个字符串，输出每个单词，重新用下划线连接输出。

如果字符串两边有空格？

```
using System;

class Program
{
    static void Main(string[] args)
    {
        string inputString;
        inputString = "I am a student";
        inputString = inputString.Trim();
        string[] splitStrings = inputString.Split(' ');
        string joinString = string.Join("_", splitStrings);
        Console.WriteLine(joinString);
    }
}
```

String常用方法

- C# 中常用的字符串处理方法：
  - Equals()：比较两个字符串的值是否相等
  - ToLower()：将字符串转换成小写形式
  - IndexOf()：查找某个字符在字符串中的位置
  - Substring()：从字符串中截取子字符串
  - Join()：连接字符串
  - Split()：分割字符串
  - Trim()：去掉字符串两边的空格
  - ……

实例

- 将一文件名的扩展名改为.dat。例：1.txt改为1.dat

实例

- 将一文件名的扩展名改为.dat。例：1.txt改为1.dat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _011
{
    class Program
    {
        static void Main(string[] args)
        {
            string filename = @"1.2.txt";
            int indexDot = filename.LastIndexOf('.');
            string extendName = "dat";
            filename = filename.Substring(0, indexDot+1);
            filename += extendName;
            Console.WriteLine(filename);
        }
    }
}
```

已有如下代码，按要求增加功能：

```
static void Main(string[ ] args)
{
    string email;  // 电子邮件地址
    Console.WriteLine("请输入你的邮箱：");
    email = Console.ReadLine();
    Console.WriteLine("你的邮箱是 {0}", email);
    Console.WriteLine("继续输入邮箱吗？");
    string input = Console.ReadLine();
}
```

需实现的功能：

1、 输入 yes 时，程序循环执行

2、 兼容以下各种形式 yes（YES、yes、 YeS .....）

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _011
{
    class Program
    {
        static void Main(string[] args)
        {
            while (1)
            {
                string email;   // 电子邮件地址
                Console.WriteLine("请输入你的邮箱：");
                email = Console.ReadLine();
                Console.WriteLine("你的邮箱是 {0}", email);
                Console.WriteLine("继续输入邮箱吗？");
                string input = Console.ReadLine();
                if (input.ToUpper() == "YES") continue;
                else break;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _011
{
    class Program
    {
        static void Main(string[] args)
        {
            while (1)
            {
                string email;   // 电子邮件地址
                Console.WriteLine("请输入你的邮箱：");
                email = Console.ReadLine();
                Console.WriteLine("你的邮箱是 {0}", email);
                Console.WriteLine("继续输入邮箱吗？");
                string input = Console.ReadLine();
                if (input.ToUpper() == "YES") continue;
                else break;
            }
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _011
{
    class Program
    {
        static void Main(string[] args)
        {
            while (true)
            {
                string email;   // 电子邮件地址
                Console.WriteLine("请输入你的邮箱：");
                email = Console.ReadLine();
                Console.WriteLine("你的邮箱是 {0}", email);
                Console.WriteLine("继续输入邮箱吗？");
                string input = Console.ReadLine();
                if (input.ToUpper() == "YES") continue;
                else break;
            }
        }
    }
}
```

数值与字符串的转换

- 数值型字符串转数值：Parse
- 数值转字符串：ToString

```
int i = int.Parse("12");
string str = i.ToString();
str = 1.ToString();
```

string类的Format()方法用来格式化字符串


```
string myString = string.Format("格式字符串", 参数列表);
```

包括占位符{x}

代表

```
string myString = string.Format ("{0} 乘以 {1} 等于 {2} ", 2, 3, 2*3);
```

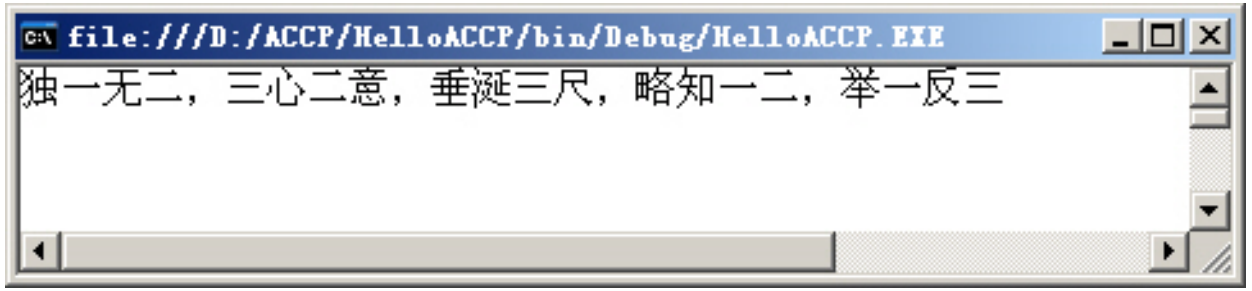
2乘以3等于6

格式字符串

参数列表

补充下划线处的占位符，输出五个成语

```
string yi = "一";
string er = "二";
string san = "三";
string word = string.Format(
    "独{0}无{1}，{2}心{1}意，垂涎{2}尺，略知{0}{1}，举{0}反{2}",
    yi, er, san);
Console.WriteLine(word);
```



枚举

默认为internal

整型(除char),
默认为int

- [access modifiers] enum <identifier> [:enum_base] {enum body}

```
class MyApp
{
    enum Fabric : int {
        Cotton = 1,
        Silk = 2,
        Wool = 4
    }
    static void Main()
    {
        Fabric fab = Fabric.Cotton;
        int fabNum = (int) fab;
        string fabType = fab.ToString();
        Console.WriteLine(fab);
    }
}
```

枚举

默认为internal

整型(除char)，默认为int

- [access modifiers] enum <identifier> [:enum_base] {enum body}

```
class MyApp
{
    enum Fabric : int {
        Cotton = 1,
        Silk = 2,
        Wool = 4
    }
    static void Main()
    {
        Fabric fab = Fabric.Cotton;
        int fabNum = (int) fab;
        string fabType = fab.ToString();
        Console.WriteLine(fab);
    }
}
```

Cotton

枚举和位标志

枚举元素的值都是2的若干次幂，可以执行位运算。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   [Flags]
   enum Days2
   {
      None = 0x0,
      Sunday = 0x1,
      Monday = 0x2,
      Tuesday = 0x4,
      Wednesday = 0x8,
      Thursday = 0x10,
      Friday = 0x20,
      Saturday = 0x40
   }
   class myApp
   {
      static void Main()
      {
         Days2 meetingDays = Days2.Tuesday | Days2.Thursday;
         Console.WriteLine(meetingDays.ToString());
      }
   }
}
```

枚举和位标志

枚举元素的值都是2的若干次幂，可以执行位运算。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   [Flags]
   enum Days2
   {
      None = 0x0,
      Sunday = 0x1,
      Monday = 0x2,
      Tuesday = 0x4,
      Wednesday = 0x8,
      Thursday = 0x10,
      Friday = 0x20,
      Saturday = 0x40
   }
   class myApp
   {
      static void Main()
      {
         Days2 meetingDays = Days2.Tuesday | Days2.Thursday;
         Console.WriteLine(meetingDays.ToString());
      }
   }
}
```

Tuesday, Thursday

OR,AND,XOR 运算

```
static void Main()
{
        // Initialize with two flags using bitwise OR.
        Days2 meetingDays = Days2.Tuesday | Days2.Thursday;

        // Set an additional flag using bitwise OR.
        meetingDays = meetingDays | Days2.Friday;
        Console.WriteLine("Meeting days are {0}", meetingDays);

        // Test value of flags using bitwise AND.
        bool test = (meetingDays & Days2.Thursday) == Days2.Thursday;
        Console.WriteLine("Thursday {0} a meeting day.", test == true ? "is" : "is not");

        // Remove a flag using bitwise XOR.
        meetingDays = meetingDays ^ Days2.Tuesday;
        Console.WriteLine("Meeting days are {0}", meetingDays);
}
```

OR,AND,XOR 运算

Meeting days are Tuesday, Thursday, Friday
Thursday is a meeting day.
Meeting days are Thursday, Friday

```
static void Main()
{
        // Initialize with two flags using bitwise OR.
        Days2 meetingDays = Days2.Tuesday | Days2.Thursday;

        // Set an additional flag using bitwise OR.
        meetingDays = meetingDays | Days2.Friday;
        Console.WriteLine("Meeting days are {0}", meetingDays);

        // Test value of flags using bitwise AND.
        bool test = (meetingDays & Days2.Thursday) == Days2.Thursday;
        Console.WriteLine("Thursday {0} a meeting day.", test == true ? "is" : "is not");

        // Remove a flag using bitwise XOR.
        meetingDays = meetingDays ^ Days2.Tuesday;
        Console.WriteLine("Meeting days are {0}", meetingDays);
}
```

System.Enum中的方法

```
using System;
namespace App1
{
   class myApp
   {
      enum Fabric
      {
         Cotton = 1,
         Silk = 2
      }

      static void Main()
      {
         string fabStr = "Cotton";
         if (Enum.IsDefined(typeof(Fabric), fabStr))
         {
            Fabric fab = (Fabric)Enum.Parse(typeof(Fabric), fabStr);
            Console.WriteLine(Enum.GetName(typeof(Fabric), 2));
         }
      }
   }
}
```

System.Enum中的方法

```
using System;
namespace App1
{
   class myApp
   {
      enum Fabric
      {
         Cotton = 1,
         Silk = 2
      }

      static void Main()
      {
         string fabStr = "Cotton";
         if (Enum.IsDefined(typeof(Fabric), fabStr))
         {
            Fabric fab = (Fabric)Enum.Parse(typeof(Fabric), fabStr);
            Console.WriteLine(Enum.GetName(typeof(Fabric), 2));
         }
      }
   }
}
```

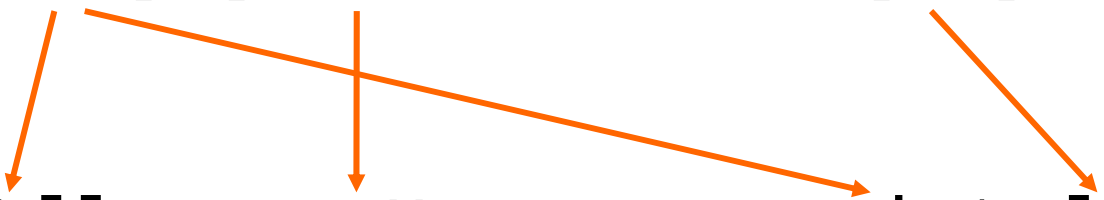
Silk

数组

- 数组是同一数据类型的一组值
- 数组属于引用类型，因此存储在堆内存中
- 数组元素初始化或给数组元素赋值都可以在声明数组时或在程序的后面阶段中进行

语法:

数据类型[,…] 数组名称 = new 类型 [n,…];



```
int[] arrayHere = new int [6];
int[,] arrayHere1 = new int [6,2];
```

数组

```
static void Main(string[] args)
{
        int count;
        Console.WriteLine("请输入准备登机的乘客人数 ");
        count=int.Parse(Console.ReadLine());
        // 声明一个存放姓名的字符串数组，其长度等于乘客人数
        string[] names = new string[count];
        // 用一个 for 循环来接受姓名
        for(int i=0; i<count; i++)
        {
                Console.WriteLine("请输入第 {0} 个乘客的姓名 ",i+1);
                names[i]=Console.ReadLine();
        }

        Console.WriteLine("已登机的乘客有： ");
        // 用 foreach 循环显示姓名
        foreach(string disp in names)
        {
                Console.WriteLine("{0}", disp);
        }
}
```

数组声明

初始化数组元素的循环

显示输出的循环

不规则数组

一维数组和多维数组都属于矩形数组，而C#所特有的不规则数组是数组的数组，在它的内部，每个数组的长度可以不同，就像一个锯齿形状。

（1）不规则数组的声明

语法形式：

type [ ] [ ] [ ] arrayName ;

方括号[ ]的个数与数组的维数相关。

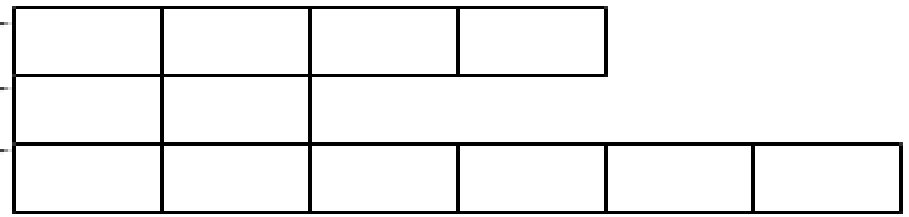
例如：

int [ ] [ ] jagged ;　　// jagged是一个int类型的二维不规则数组

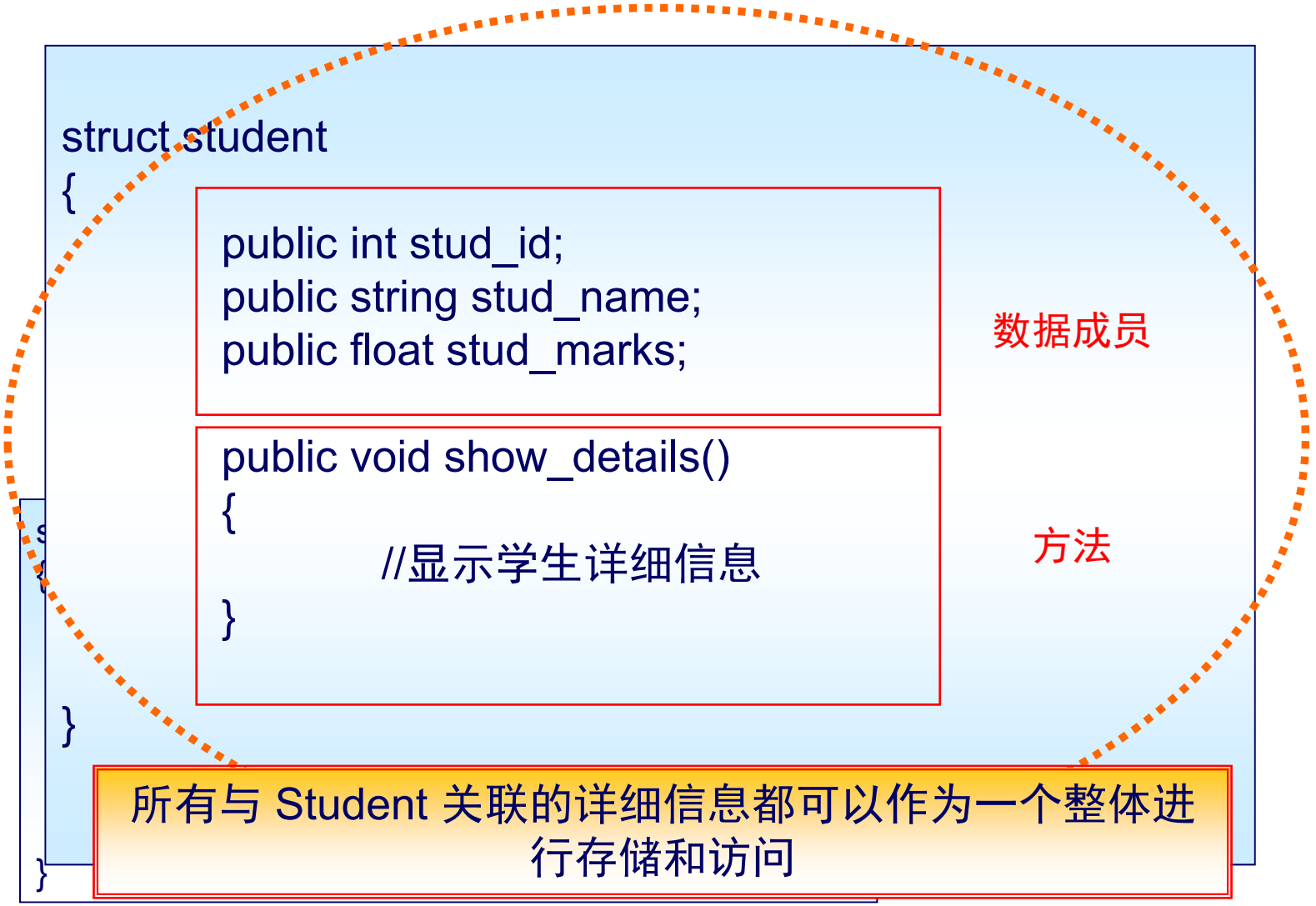
（2）创建数组对象

以二维不规则数组为例：

```
int [] [] jagged;
jagged = new int [3][];
jagged[0] = new int [4];
jagged[1] = new int [2];
jagged[2] = new int [6];
```



结构体


struct student
{
public int stud_id;
public string stud_name;
public float stud_marks;
public void show_details()
{
//显示学生详细信息
}
}
数据成员
方法
所有与 Student 关联的详细信息都可以作为一个整体进行存储和访问

变量

```
static void Main(string[] args)
{
        // 声明布尔型、字符串型、整型、短整型和浮点型变量
        bool t = false;
        short n1 = 30;
        int n2 = 1200;
        string str = "jeny";
        float n3 = 23.1f;

        // 显示变量值
        Console.WriteLine ("布尔值  = " + t);
        Console.WriteLine ("短整型值  = " + n1);
        Console.WriteLine ("整型值  = " + n2);
        Console.WriteLine ("字符串值  = " + str);
        Console.WriteLine ("浮点值  = " + n3);
}
```

变量中存储的值取决于该变量的类型

布尔值 = False
短整型值 = 30
整型值 = 1200
字符串值 = jeny
浮点值 = 23.1

常量

- const 关键字用于声明常量

```
static void Main(string[] args)
{
        // PI常量PI
        const float _pi = 3.1415169F;
        // 由地球引力引起的加速度常量，单位为 cm/s*s
        const float _gravity = 980;
        // 钟摆的长度
        int length = 60;
        // 钟摆的周期
        double period = 0;
        // 钟摆周期的计算公式
        period = 2 * _pi * Math.Sqrt(length / _gravity);
        Console.WriteLine ("钟摆的周期为 {0} 秒", period);
}
```

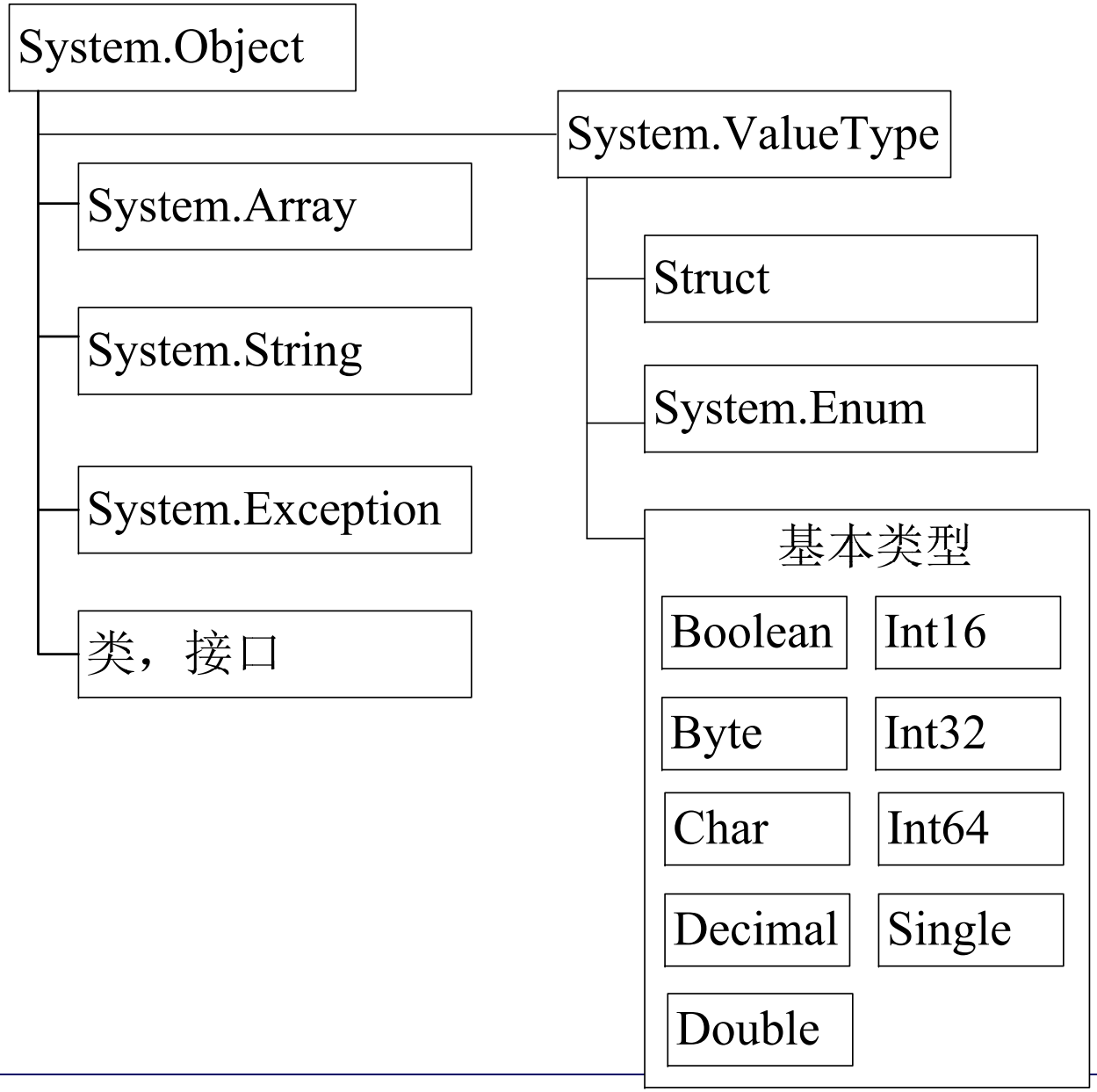
声明常量

在表达式中使用常量

C#的基本数据类型

- C#中的数据类型分为两个基本类别
  - 值类型
    - 表示实际数据
    - 只是将值存放在内存中
    - 值类型都存储在栈中
    - int、char、enum、struct
  - 引用类型
    - 表示指向数据的指针或引用
    - 包含内存堆中对象的地址
    - 为 null，则表示未引用任何对象
    - 类、接口、数组、字符串

引用类型和值类型


System.Object
System.ValueType
System.Array
Struct
System.String
System.Enum
System.Exception
基本类型
类，接口
Boolean
Int16
Byte
Int32
Char
Int64
Decimal
Single
Double

```
static void Main(string[] args)
{
        // 声明一个值类型的整型数据类型
        int value = 130;
        Console.WriteLine("该变量的初始值为 {0}", value);
        Test(value);
        // 由于该数据类型属于值类型，所以将恢复其初始值
        Console.WriteLine("该变量的值此时为 {0}", value);
}
static void Test(int byVal)
{
        int t = 20;
        byVal = t* 30;
}
```

将value的初始值传递给Test()方法

不反映已经改变的val值，而保留原始值

```
class DataType
{
    public int Value;
}
```

```
static void Main(string[] args)
{
        DataType objTest = new DataType ();
        objTest.Value = 130;
        // 传递属于引用类型的对象
        Test(objTest);
        // 由于该数据类型属于引用类型，所以会考虑新处理的值
        Console.WriteLine("变量的值为 {0}", objTest.Value);
}
static void Test(DataType data)
{
        int t = 20;
        data.Value = temp * 30;
}
```

将 DataTypeTest 的引用传递给 Test()

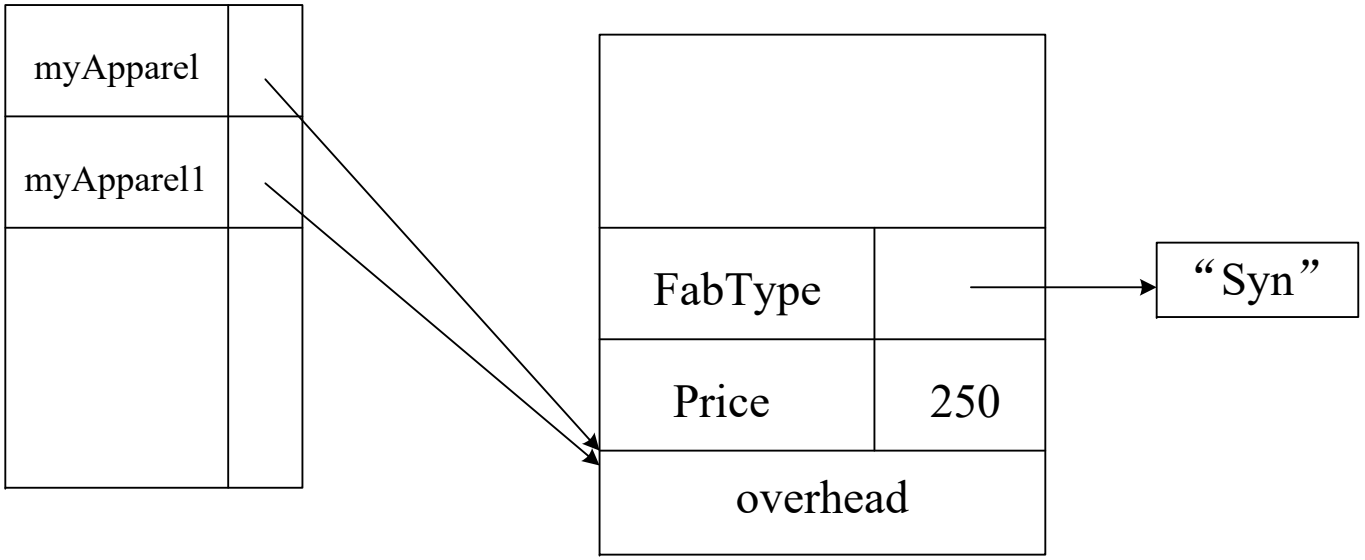
反映已经改变的value值

被传递的value在Test()中改变

引用类型和值类型的内存分配

```
class Apparel
{
   public double Price = 250.0;
   public string FabType = “Syn”;
}
```

- Apparel myApparel = new Apparel();
- Apparel myApparel1 = myApparel; //处理高效



引用类型和值类型的转换
装箱与拆箱

- 装箱即将值类型转换为引用类型

```
int age = 17;
Object refAge = age;
```

- 拆箱即将引用类型转换为值类型

```
Object refAge = 17;
int newAge = (int)refAge;
double newAge = (double)refAge; //错误，要具有相同类型
```

运算符和表达式

类别	运算符	说明	表达式
算术运算符	+	执行加法运算（如果两个操作数是字符串，则该运算符用作字符串连接运算符，将一个字符串添加到另一个字符串的末尾）	操作数1 + 操作数2
	-	执行减法运算	操作数1 - 操作数2
	*	执行乘法运算	操作数1 * 操作数2
	/	执行除法运算	操作数1 / 操作数2
	%	获得进行除法运算后的余数	操作数1 % 操作数2
	++	将操作数加 1	操作数++或++操作数
	--	将操作数减 1	操作数-- 或--操作数
	~	将一个数按位取反	~操作数

运算符和表达式

类别	运算符	说明	表达式
三元运算符（条件运算符）	?:	检查给出的第一个表达式 **expression** 是否为真。如果为真，则计算 **operand1**，否则计算 **operand2**。这是唯一带有三个操作数的运算符	表达式**?** 操作数**1:** 操作数**2**

运算符和表达式

类别	运算符	说明	表达式
比较运算符	>	检查一个数是否大于另一个数	操作数1 > 操作数2
	<	检查一个数是否小于另一个数	操作数1 < 操作数2
	>=	检查一个数是否大于或等于另一个数	操作数1 >= 操作数2
	<=	检查一个数是否小于或等于另一个数	操作数1 <= 操作数2
	==	检查两个值是否相等	操作数1 == 操作数2
	!=	检查两个值是否不相等	操作数1 != 操作数2

运算符和表达式

类别	运算符	说明	表达式
成员访问运算符	.	用于访问数据结构的成员	数据结构.成员
赋值运算符	=	给变量赋值	操作数1 = 操作数2
逻辑运算符	&&	对两个表达式执行逻辑“与”运算	操作数1 && 操作数2
	||	对两个表达式执行逻辑“或”运算	操作数1 || 操作数2
	!	对两个表达式执行逻辑“非”运算	! 操作数
	( )	将操作数强制转换为给定的数据类型	(数据类型) 操作数

运算符和表达式

赋值运算符 (=)

变量 = 表达式;

例如：

身高 = 177.5;
体重 = 78;
性别 = “m”;

运算符和表达式

一元运算符 (++/--)

Variable ++;
相当于
Variable = Variable + 1;

Variable --;
相当于
Variable = Variable - 1;

运算符和表达式

运算符	计算方法	表达式	求值	结果（设 X = 10）
+=	运算结果 = 操作数1 + 操作数2	X+= 2	X=X+2	12
-=	运算结果 = 操作数1 - 操作数2	X-= 2	X=X-2	8
*=	运算结果 = 操作数1 * 操作数2	X*= 2	X=X*2	20
/=	运算结果 = 操作数1 / 操作数2	X/= 2	X=X/2	5
%=	运算结果 = 操作数1 % 操作数2	X%= 2	X=X%2	0

运算符和表达式

优先级	说明	运算符	结合性
1	括号	( )	从左到右
2	自加/自减运算符	+ + / --	从右到左
3	乘法运算符 除法运算符 取模运算符	* / %	从左到右
4	加法运算符 减法运算符	+ -	从左到右
5	小于 小于等于 大于 大于等于	< <= > >=	从左到右
6	等于 不等于	= !=	从左到右 从左到右
7	逻辑与	&&	从左到右
8	逻辑或	||	从左到右
9	赋值运算符和快捷运算符	= += *= /= %= -=	从右到左

选择结构

- 选择结构用于根据表达式的值执行语句

if … else

语法：

```
if (<条件>)
{
<语句块>
}
else
{
<语句块>
}
```

条件：只能是bool类型的值

选择结构

switch…case

```
switch (“cotton”)
{
        case “COTTON”:
        case “cotton”:
        …
        break;
        case 值3:
        case 值4:
        …
        break;
}
```

- 表达式可以是int、字符或字符串
- C#不允许从一个case块继续执行到下一个case块。每个case块必须以一个跳转控制语句break、goto或return结束
- 多个case标签可以对应一个代码块

```
switch(“cotton”)
{
    case “cotton”:
        Console.WriteLine(“case1”);
    case “fabric”:
        Console.WriteLine(“case2”);
    ….
}
```

```
switch(“cotton”)
{
    case “cotton”:
    case “fabric”:
        Console.WriteLine(“case1 and case2”);
        break;
    ….
}
```

```
switch(“cotton”)
{
    case “cotton”:
        Console.WriteLine(“case1”);
        break;
    case “fabric”:
        Console.WriteLine(“case2”);
    ….
}


switch(“cotton”)
{
    case “cotton”:
    case “fabric”:
        Console.WriteLine(“case1 and case2”);
        break;
    ….
}
```

循环结构

- 循环结构用于对一组命令执行一定的次数或反复执行一组命令，直到指定的条件为真。
- 循环结构的类型
  - while 循环
  - do 循环
  - for 循环
  - foreach 循环

条件：只能是bool类型的值

while循环

- while 循环反复执行指定的语句，直到指定的条件为真
- 语法 :

```
while (条件)
{
    // 语句
}
```

- break 语句可用于退出循环
- continue 语句可用于跳过当前循环并开始下一循环

do…while循环

- do…while 循环与 while 循环类似，二者区别在于do…while 循环中即使条件为假时也至少执行一次该循环体中的语句。
- 语法：

```
do
   {
      // 语句
   } while (条件)
```

for循环

- for 循环要求只有在对特定条件进行判断后才允许执行循环
- 这种循环用于将某个语句或语句块重复执行预定次数的情形

语法：

```
for (初始值; 条件; 增/减)
{
    //语句
}
```

foreach循环

- **foreach** 循环用于遍历整个集合或数组

语法：

```
foreach (数据类型 元素(变量) in 集合或者数组)
{
    //语句
}
```

```
foreach(int a in array)
{
    Console.WriteLine(a);
}
```

foreach循环

```
static void Main(string[] args)
{
        // 存放字母的个数
        int Letters = 0;
        // 存放数字的个数
        int Digits = 0;
        // 存放标点符号的个数
        int Punctuations = 0;
        // 用户提供的输入
        string instr;
        Console.WriteLine("请输入一个字符串 ");
        instr = Console.ReadLine();

        // 声明 foreach 循环以遍历输入的字符串中的每个字符。
        foreach(char ch in instr)
          {
                // 检查字母
                if(char.IsLetter(ch))
                    Letters++;
                // 检查数字
                if(char.IsDigit(ch))
                    Digits++;
                // 检查标点符号
                if(char.IsPunctuation(ch))
                    Punctuations++;
        }
        Console.WriteLine("字母个数为： {0}", Letters);
        Console.WriteLine("数字个数为： {0}", Digits);
        Console.WriteLine("标点符号个数为： {0}", Punctuations);
}
```

为所有计数器设置初始值

接受输入

对输入的每一个字符都进行循环

使用了所有输入的字符之后，循环自动终止

```
using System;
public class WriteTest
{
    public static void Main()
    {
      int[] array = { 1, 2, 3, 4, 5 };
      foreach (int item in array)
      {
         Console.WriteLine(item);
      }
    }
}
```

```
using System;
public class WriteTest
{
    public static void Main()
    {
      int[] array = { 1, 2, 3, 4, 5 };
      foreach (int item in array)
      {
         Console.WriteLine(item);
      }
    }
}
```

```
using System;
public class WriteTest
{
    public static void Main()
    {
      int[] array = { 1, 2, 3, 4, 5 };
      foreach (int item in array)
      {
          item += item;
          Console.WriteLine(item);
      }
    }
}
```

C#的预处理指令

- 预处理指令由字符#标识，并且字符#必须是该行的第一个非空字符。
- 预处理指令最常见的三个用途是：完成条件编译、增加诊断来报告错误和警告、定义代码域。

C#的预处理指令

C#预处理命令	说明
#define #undef	用于定义一个符号，后取消对一个符号得定义。如果定义了一个符号，那么在#if指令中使用时这个符号计算为true。
#if #elif #else #endif	类似C#中的if、elseif和else语句
#line	改变行号序列，并且可以标识该行的源文件
#region #endregion	用于指定一个代码块，使用Visual studio.NET的大纲特性时可以展开或折叠这个代码块
#error	导致编译器报告一个致命错误
#warming	导致编译器报告一个警告，并继续处理

```
        1 个引用
59      public static List<System.Diagnostics.PerformanceCounter> GetPerformanceCounters()
60      {
61          #region
62          List<System.Diagnostics.PerformanceCounter> performanceCounters = new List<System.
63          int procCount = System.Environment.ProcessorCount;
64          for (int i = 0; i < procCount; i++)
65          {
66              System.Diagnostics.PerformanceCounter pc = new System.Diagnostics.PerformanceCou
67              performanceCounters.Add(pc);
68          }
69          #endregion
70
71          return performanceCounters;
72      }
```

```
1 个引用
59  public static List<System.Diagnostics.PerformanceCounter> GetPerformanceCounters()
60  {
61      #region
62      List<System.Diagnostics.PerformanceCounter> performanceCounters = new List<System.
63      int procCount = System.Environment.ProcessorCount;
64      for (int i = 0; i < procCount; i++)
65      {
66          System.Diagnostics.PerformanceCounter pc = new System.Diagnostics.PerformanceCou
67          performanceCounters.Add(pc);
68      }
69      #endregion
70
71      return performanceCounters;
72  }
```

```
1 个引用
59  public static List<System.Diagnostics.PerformanceCounter> GetPerformanceCounters()
60  {
61      #region
70
71      return performanceCounters;
72  }
```

```
#define CLIENT
#define DEBUG
using System;
public class MyApp
{
   public static void Main()
   {
#if DEBUG && INHOUSE
#warning Debug in on.
#elif DEBUG && CLIENT
#error Debug not allowed in Client Code.
#endif
   }
}
```

```
#define CLIENT
#define DEBUG
using System;
public class MyApp
{
    public static void Main()
    {
#if DEBUG && INHOUSE
#warning Debug in on.
#elif DEBUG && CLIENT
#error Debug not allowed in Client Code.
#endif
    }
}
```

控制台输入和输出

- 数据输入(Read,ReadLine)

Console.Read方法用来从控制台读取一个字符，其定义如下:

```
Public static int Read();
```

- Read方法返回所读取的字符的Unicode编码值。
- 注意： Read方法的返回变量是32位的整数，如果需要得到输入的字符，则必须通过数据类型的显式转换才能得到相应的字符。

控制台输入和输出

Console.Read ()

```
//ReadTest.cs
using System;
public class ReadTest
{
        public static void Main()
        {
                int     i;
                char    ch;
                i=Console.Read();
                ch=(char) i;        //显式类型转换
                Console.WriteLine(i);
                Console.WriteLine(ch);
        }
}
```

控制台输入和输出

Console.Read ()

```
//ReadTest.cs
using System;
public class ReadTest
{
        public static void Main()
        {
                int     i;
                char    ch;
                i=Console.Read();
                ch=(char) i;        //显式类型转换
                Console.WriteLine(i);
                Console.WriteLine(ch);
        }
}
```

运行结果：

A

65

A

控制台输入和输出

- Console.ReadLine方法用来从控制台读取一行字符，定义如下:

  Public static string ReadLine();

- Read方法返回所读取一行字符的字符串。一般情况下，一行输入是指从输入一个字符开始，遇到回车符号为止。

控制台输入和输出

```
using System;
using System.Globalization;
public class ReadTest
{
        public static void Main()
        {
                int         i;
                double    d;
                string     str;
                str=Console.ReadLine();    //由控制台输入整数字符串
                i=int.Parse(str);              //整数字符串转换为整数
                Console.WriteLine(i);
                str=Console.ReadLine();    //由控制台输入浮点字符串
                d=double.Parse(str);         //浮点字符串转换为浮点数
                Console.WriteLine(d);
        }
}
```

控制台输入和输出

```
using System;
using System.Globalization;
public class ReadTest
{
        public static void Main()
        {
                int         i;
                double      d;
                string      str;
                str=Console.ReadLine();    //由控制台输入整数字符串
                i=int.Parse(str);          //整数字符串转换为整数
                Console.WriteLine(i);
                str=Console.ReadLine();    //由控制台输入浮点字符串
                d=double.Parse(str);       //浮点字符串转换为浮点数
                Console.WriteLine(d);
        }
}
```

运行结果：

```
1234
1234
123.456
123.456
```

控制台输入和输出

- 数据输出(Write WriteLine)
- Console.Write方法用来向控制台输出一个字符，但控制台的光标不会移到下一行。其定义如下:

```
public static void Write(value);

public static void Write(string format,object o1,……);
```

- 注意：格式化format同格式化函数Format中的格式化串类似，其格式如下：

```
{N[,M][:formatstring]}
```

其中，字符N表示输出变量的序号，M表示输入变量在控制台中所占的字符空间，如果这个数字为负数，则按照左对齐的方式输出，若为正数，则按照右对齐方式输出。

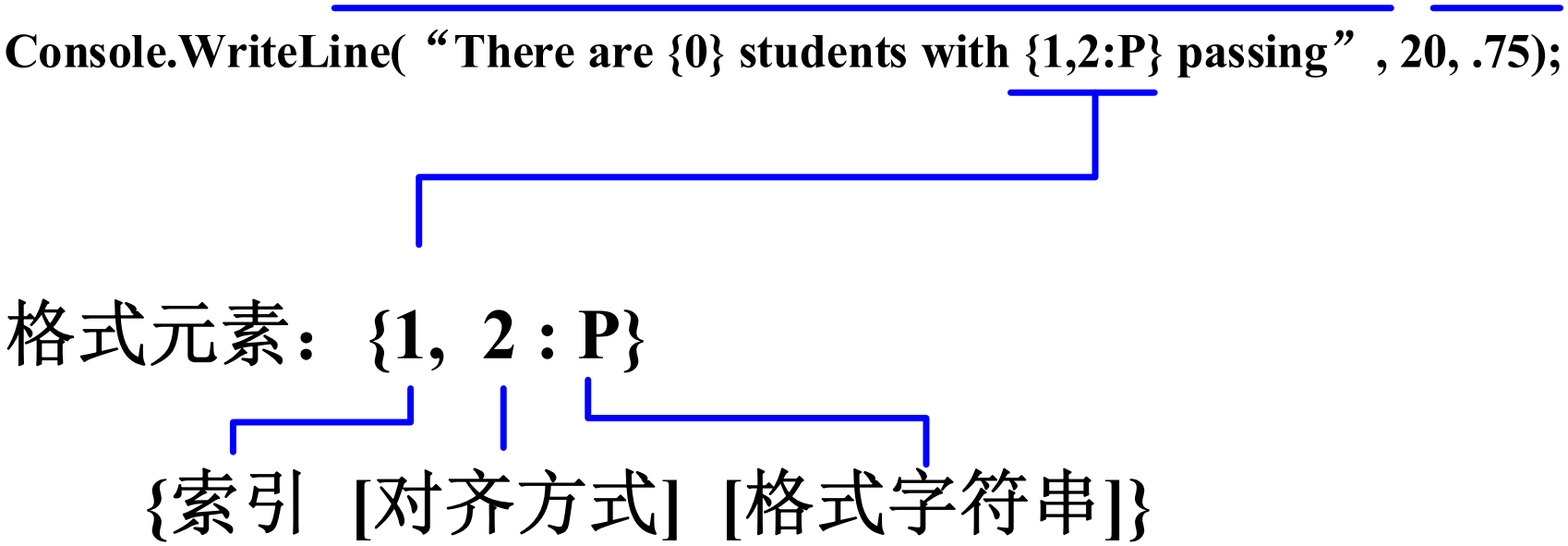
控制台输入和输出

- Console.WriteLine方法用来向控制台输出一行字符，即WriteLine方法在输出信息之后，在信息的尾部自动添加“\r\n”字符，表示回车换行。

```
public static void WriteLine(value);

public static void WriteLine(string format,object o1,……);
```

- 注意：格式化format同Write中的格式化参数完全一样。


Console.WriteLine(“There are {0} students with {1,2:P} passing”, 20, .75);
格式元素：{1, 2 : P}
{索引 [对齐方式] [格式字符串]}

整数数据格式(D或d)

- 字符D（或d）用来将数据转换为十进制数整数格式。紧跟在字符D后面的数字则规定了数字将表示的位数。如果这个数字小于整数数据的实际位数，则显示所有的整数位，若这个数字大于整数数据的实际位数，则在整数数据的前面用数字0补足所有的位数。如:

```
int CurValue=12345678;
Console.WriteLine("{0:D5}",CurValue);
Console.WriteLine ("{0:D9}",CurValue);
```

结果：

```
12345678
012345678
```

十六进制数格式(X或x)

- 字符X（或x）用来表示十六进制数格式。这种数据格式是将整数转换成“xxxxxx”形式的十六进制整数，字符X后面的数字规定了格化数据的数字个数。

```
int CurValue=123456;
Console.WriteLine (“0:x}”,CurValue);
Console.WriteLine (“{0:X6}”,CurValue);
```

结果：

1e240

01E240

自然数据格式(N或n) xxx,xxx,xx

- 字符N（或n）用来表示自然数据格式。这种数据格式是将数据表示成“xxx,xxx,xx”，字符N后面的数字规定了数据格式中小数点后面的数字个数。

```
int CurValue=12345678;
double fCurValue=12345678.125;
Console.WriteLine("{0:N}",CurValue);
Console.WriteLine("{0:N2}",fCurValue);
```

结果：

```
12,345,678.00
12,345,678.13
```

货币金额格式(C或c)

- 字符C（或c）用来将数据转换为货币金额格式。紧跟在字符C后面的数字定义货币金额数据小数点后应保留的位数。如:

```
int CurValue=12345678;

double fCurValue=12345678.125;

Console.WriteLine("{0:C2}",CurValue);

Console.WriteLine("{0:C2}",fCurValue);
```

结果：

￥12,345,678.00

￥12,345,678.13

定点数据格式(F或f) xxxxxx.xx

- 字符F（或f）用来将浮点数据转换为定点数据格式。紧跟在字符F后面的数字规定小数点后面应保留的位数，如果指定的数字大于数据小数部分的位数，则在小数点的最后用数字0补足，如果F后面没有数字，则默认为两位小数。如:

```
int CurValue=12345678;
double fCurValue=12345678.125;
Console.WriteLine ("{0:F2}",CurValue);
Console.WriteLine ("{0:F4}",fCurValue);
```

结果：

12345678.00

12345678.1250

科学计数法格式(E或e)

x.xxxxE+xxx或x.xxxxE-xxx

xxxxe+xxx或x.xxxxe-xxx

- 浮点数常数可以使用科学计数法表示（也叫指数形式）
- 字符E（或e）用来将数据转换为科学计数法形式。紧跟在字符E后面的数字规定小数点后应保留的位数，如果E后面没有数字，则小数点后保留6位(有效位数7位)。如:

```
double fCurValue=12345678.125;
Console.WriteLine ("{0:E3}",fCurValue);
Console.WriteLine("{0:E}",fCurValue);
```

结果：

```
1.235E+007
1.234568E+007
```

通用数据格式(G或g)

- 字符G（或g）用来表示通用数据格式。这个数据可能使用科学计数法来表示，也可能使用定点数据格式表示。在C#中，若字符G后面没有数字，即没有规定浮点数的精度，则用定点数据格式；如果G后面有指定数字（精度），则用科学计数法表示。

```
double fCurValue=12345678.125;
Console.WriteLine ("{0:G}",fCurValue);
Console.WriteLine ("{0:G4}",fCurValue);
```

结果：

12345678.125

1.235E+07

控制台输入和输出

```
using System;
public class WriteTest
{
   public static void Main()
   {
         int  i=32767;
         double d=456.56789;
         Console.Write(“i=0x{0,8:X}\td={1,10:F3}”,i,d);
         Console.Write(“i=0x{0,-8:X}\td={1,-10:F3}”,i,d);
    }
}
```

控制台输入和输出

```
using System;
public class WriteTest
{
   public static void Main()
   {
          int i=32767;
          double d=456.56789;
         Console.Write(“i=0x{0,8:X}\td={1,10:F3}”,i,d);
         Console.Write(“i=0x{0,-8:X}\td={1,-10:F3}”,i,d);
    }
}
```

输出结果：

```
 i=0x    7FFF      d=   456.568i=0x7FFF      d=456.568
```

控制台输入和输出

```
using System;
public class WriteTest
{
        public static void Main()
        {
                int  i=32767;
                double d=456.56789;
                Console.WriteLine("i=0x{0,8:X}\td={1,10:F3}",i,d);
                Console.WriteLine("i=0x{0,-8:X}\td={1,-10:F3}",i,d);
                Console.WriteLine("i={0,-8:D}\td={1,-10:C3}",i,d);
        }
}
```

控制台输入和输出

```
using System;
public class WriteTest
{
        public static void Main()
        {
                int  i=32767;
                double d=456.56789;
                Console.WriteLine("i=0x{0,8:X}\td={1,10:F3}",i,d);
                Console.WriteLine("i=0x{0,-8:X}\td={1,-10:F3}",i,d);
                Console.WriteLine("i={0,-8:D}\td={1,-10:C3}",i,d);
        }
}
```

输出结果：

```
i=0x    7FFF      d=   456.568
i=0x7FFF          d=456.568
i=32767           d=￥456.568
```

实例：输入两个数x和y，计算x和y的加、减、乘、除，并输出计算结果。

实例：输入两个数x和y，计算x和y的加、减、乘、除，并输出计算结果。

```
using System;
class MyApp
{
    static void Main()
    {
        double x = double.Parse(Console.ReadLine());
        double y = double.Parse(Console.ReadLine());
        Console.WriteLine("{0} + {1} = {2}", x, y, x+y);
        Console.WriteLine("{0} - {1} = {2}", x, y, x-y);
        Console.WriteLine("{0} * {1} = {2}", x, y, x*y);
        Console.WriteLine("{0} / {1} = {2}", x, y, x/y);
    }
}
```