IPL 电子信息学院 武汉大学 Wuhan University

快速掌握C#数据集合类型

QUICKSTART FOR MASTERING C# COLLECTIONS

王文伟 Wang Wenwei, Dr.-Ing.
Tel: 189-71562600
Email: wwwang@189.cn
Web: http://ipl.whu.edu.cn/sites/ced/st/

电子信息学院 Table of Contents 武汉大学 Wuhan University

掌握若干集合类型是良好编程的必要条件

0.掌握若干集合类型是良好编程的必要条件

- 程序往往要处理很多的数据，而众多的数据间存在着内在的联系。对一个数据（元素）的集合来说，如果在数据元素之间存在一种或多种特定的关系，则称为数据结构（data structure）。因此，“结构”就是指数据元素之间存在的关系。数据结构可以看成是关于数据集合的数据类型，是一种特殊的抽象数据类型ADT。
- 将待处理数据依内在关系作为集合处理才能写出符合逻辑并且高效的程序。掌握若干集合类型是良好编程的必要条件。

1.0 数组是一种基本而重要的数据集合类型

- 数组是由一组具有相同类型的元素组成的集合，元素依次存储于一个连续的内存空间。数组是其他数据结构实现顺序存储的基础。
- 数组元素在数组中的位置称为**数组的下标（index）**，通过下标，可以找到元素的存储地址，从而访问该元素。
- 数组下标的个数就是**数组的维数**，有一个下标的数组是一维数组，有两个下标的就是二维数组，以此类推。二维数组可以定义为“数组的数组”。
- C# 支持一维数组、多维数组（矩形数组）和数组的数组（交错的数组）。C# 中数组变量是引用类型变量（对象），数组元素的下标从零开始。

两种为数组分配内存空间的方式

- 编译时分配数组空间：程序声明数组时给出数组元素类型和元素个数，编译程序为数组分配所需的空间。当程序开始运行时，数组即获得系统分配的一块地址连续的内存空间。
  例：int a[10];
- 运行时分配数组空间：程序声明时，仅需说明数组元素类型，不指定数组长度。当程序运行中需要使用数组时，向系统申请指定长度数组所需的存储空间。当数组使用完之后，需要向系统归还所占用的内存空间。

```
int* a;
a = (int*)malloc(10*sizeof(int));
```

```
int[] a;
a = new int[10];
```

1.1 C#中的一维数组

- 声明数组: <类型>[ ] 数组名，例：int[] a;
- 声明数组并没有实际创建它，数组是对象，必须用new操作符为数组分配空间后，数组才真正占有实在的存储单元：<数组名> = new <类型>[<长度>]，例：a = new int[10];
- 声明数组的同时进行初始化，例：int[] a={1,2,3,4,5};
- C#中的数组是在运行时分配所需空间，声明数组变量时不指定数组长度，使用new运算符为数组分配空间后，数组才真正占用一片地址连续的存储单元空间。
- 当数组使用完之后，不需要立即向系统归还所占用的内存空间。因为.NET平台的垃圾回收机制将自动判断对象是否在使用，并能够自动销毁不再使用的对象，收回对象所占的资源。

C#中的一维数组(II)

- 数组类型是一种类类型，任何数组类型都隐含继承自基类型System.Array，因此数组实例都具有对象特性。
- System.Array 类中定义的属性以及其他公有成员都可以供数组使用，例如Length属性可以获得数组元素的个数。Array类还提供了许多用于排序、搜索和复制数组的方法。
- Array的**公共属性**:
  - virtual int Length {get;}
    //返回数组的所有维数中元素的总数。
  - virtual int Rank {get;}
    //获取数组的维数。

a.Length
a.Rank

C#中的一维数组(III)

- Array的**公共方法**:
  - int GetLength(int dimension);
    //获取数组指定维中的元素数
  - void CopyTo(Array dstArray, int Index)
    //将当前数组的所有元素复制到目的数组中的指定位置
  - static void Copy(Array srcAr, int srcIdx, Array dstAr, int dstIdx, int length)
    //从指定索引开始，复制源数组中指定长度的一系列元素到目数组中指定位置，有多个重载方法。
  - static int IndexOf<T>(T[] a, T k)
    //返回给定数据首次出现位置
  - static void Sort(Array a)
    //对整个一维数组元素进行排序，多个重载方法。

Array.Sort(a)　　Array.IndexOf<int>(a, 10)

1.2 C#的多维数组

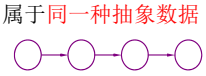
- 用说明多个下标的形式来定义多维数组，例如：
  int[,] items = new int[5,4];
  声明了一个二维数组items，并分配5×4个存储单元。
- 可以声明并初始化多维数组，例如：
  int[,] numbers = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };
- 初始化时可以省略数组的大小，如下所示：
  int[,] numbers = new int[,] { {1, 2}, {3, 4}, {5, 6} };
- C#中的二维数组按行优先顺序存储数组的元素。

【例5. 1】 数组的搜索与排序

```
using System;
 class ArrayTest {
  static void Main(string[] args) {
    double[] a = { 3.0, 4.0, 1.0, 2.0, 5.0 };
    int i = Array.IndexOf<double>(a, 5.0);
    Console.WriteLine("{0}'s index is: {1}", 5.0, i);
    Console.Write("Sorted Array: ");
    Array.Sort(a);
    foreach(double f in a) Console.Write("{0} ", f);
    Console.WriteLine();
  } }
```


2.0 线性表的概念及类型定义

- 线性表是一组具有某种共性的数据元素的有序排列，元素之间具有顺序关系。记作：LinearList = {a_0，a_1，a_2，…，a_{n-1}}
  除第一个和最后一个元素外，每个元素只有一个前驱元素和一个后继元素。第i个数据元素a_i的直接前驱数据元素a_{i-1}，直接后继数据元素a_{i+1}。
- 线性表可在任意位置进行插入和删除元素的操作。
- 线性表中元素的类型可以是数值型或字符串型，也可以是其他更复杂的自定义数据类型。线性表中的数据元素至少具有一种相同的属性，属于同一种抽象数据类型。



2.1 C#中的泛型线性表类List<T>

- 在System.Collections.Generic命名空间中定义了泛型线性表类List<T>，它提供了一种“元素个数可按需动态增加”的数组。
- List类的属性和方法：
  公共构造函数
  - List<T>();　//构造 List<T> 类的新实例
  - List<T>(IEnumerable<T> c);// 新实例包含从指定集合c复制的元素
  - List<T>(int initCapacity);

List<double> ml = new List<double>();

List的公共属性

- virtual int Count {get;}　　//返回线性表的长度
- virtual int Capacity {get; set; }
  //获取或设置线性表可包含的元素数
- virtual T this[int index] {get; set;}
  //获取或设置指定索引处的元素

```
List<double>  ml = new List<double>();
ml.Add(1.5);   ml.Add(4.3);
double d = ml [5];        ml[10] = 10.5;
```


List的公共方法

- virtual void Insert(int i, T x);//将数据元素插入指定位置
- virtual void Add(T x);//将对象添加到表的结尾处
- virtual int IndexOf(T x);//返回给定数据首次出现位置
- virtual bool Contains(T x);//确定某个元素是否在表中
- virtual void Remove(T x);
  //从表中移除特定对象的第一个匹配项
- virtual void RemoveAt(int i);//删除指定位置的数据元素
- virtual void Reverse();//将表中元素的顺序反转
- virtual void Sort();//对表中元素进行排序

声明并构造特定类型的列表举例

```
List<int>  a = new List<int> ();
        // 声明并构造int型数列表
a.Add(86); a.Add(100);
        // 向列表中添加整型元素
List<int>  nums = new List<int> {0,1,2,3};
List<string>  s = new List<string> ();
        // 声明并构造字符串列表
s.Add("Hello"); s.Add("C# 2.0");
var  st = new List< Student>();
// 声明并构造学生列表
st.Add(new Student ("8001", "王兵" , "男", 18, 92));
```


【例2.1】 创建并初始化 List 以及打印出其值

```
using System;
using System.Collections. Generic;
public class SamplesList  {
   public static void Main()  {
    // Creates and initializes a new List.
     List<string>  al = new List<string>();
     al.Add("Hello");
     al.Add("World");      al.Add("!");
     al. Insert(1, "C#");
```



```
// Displays the properties and values of the List.
    Console.WriteLine( "al" );
    Console.WriteLine( "\tCount:    {0}", al.Count );
    Console.Write( "\tValues:" );
    foreach(string o in  al){
       Console.Write( "\t{0}", o);
    }
    Console.WriteLine();
    al.Sort();
    Console.Write( "\tSorted Values:" );
    for(int i=0; i<al. Count;i++){
       Console.Write( "\t{0}", al[i]);
    }
    Console.WriteLine();
} }
```

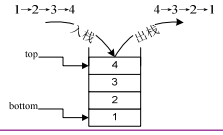
程序运行结果

```
My List:
        Count:          4
        Values:         Hello C#  World  !
Sorted Values:  !  C#  Hello  World
```

该例本身很简单，但演示了List类的实例（al线性表）的元素数目可按需动态增加，可在表中任意位置进行插入和删除数据元素的操作，一般的数组不具备这种方便的特性。

3.0 栈的概念及类型定义

- 栈（stack）是一种特殊的线性数据结构，元素之间具有顺序的逻辑关系，但插入和删除操作只允许在结构的一端进行。“后进先出”（Last In First Out，LIFO）。
- 栈中插入数据元素的过程称为入栈(push)，删除数据元素的过程称为出栈(pop)。
- 允许插入和删除操作的一端称为栈顶（stack top），不允许操作的一端称为栈底（ stack bottom）。



栈就像某种只有单个出入口的仓库，每次只允许一件件地往里面堆货物（入栈），然后一件件地往外取货物（出栈），不允许从中间放入或抽出货物。

3.2 C#中的泛型栈类Stack<T>

- 在System.Collections.Generics名字空间中定义了一个泛型栈类Stack，刻画一种具有后进先出性质的数据集合。
- Stack类的属性和方法：

公共构造函数

- Stack<T>(); //初始化Stack类的新实例
- Stack<T> (ICollection c);
- Stack<T>(int capacity);

公共属性

- virtual int Count {get;} //获取包含在栈中的元素数

```
Stack<int> s1 = new Stack<int> ();
s1.Push(12);
var s2 = new Stack<string> ();
s2.Push("Wuhan University");
i = s1.Count
```


Stack的公共方法

- virtual void Push(T x);
  //将对象插入栈的顶部
- virtual T Pop();
  //移除并返回位于栈顶部的对象
- virtual T Peek();
  //返回栈顶的对象，但不将其移除
- virtual bool Contains(T x);
  //确定某个元素是否在栈中

【例3.1】 创建Stack对象，向其添加元素以及打印出其值

```
using System; using System.Collections.Generics;
  Stack<string> s = new Stack<string>();
  s.Push("Hello");
  s.Push("World");s.Push("!");
  Console.WriteLine( "My Stack:" );
  Console.WriteLine("\tCount: {0}",
     s.Count );
  Console.Write( "\tValues:\n" );
  foreach(string o in  s)
      Console.WriteLine( "\t{0}", o);
```


程序运行结果

```
My Stack:
        Count:    3
        Values:
        !
        World
        Hello
```

输出序列的顺序与入栈的顺序相反，这是栈的先进后出（LIFO）特性造成的。

【例3.2】利用栈进行数制转换

$$N = a_n \times d^n + a_{n-1} \times d^{n-1} + \ldots + a_1 \times d^1 + a_0$$

数制转换就是要确定序列 $\{a_0, a_1, a_2, \ldots, a_n\}$

$$N = (N / d) \times d + N \% d$$

例如： $(2468)_{10}$ 转换成 $(4644)_8$ 的运算过程如下：

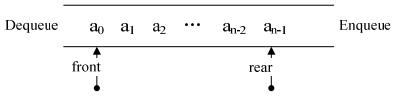
	N	N / 8	N % 8
计算顺序 ↓	2468	308	4
	308	38	4
	38	4	6
	4	0	4

输出顺序 ↑

```
using System; using System.Collections.Generic;
namespace stackqueuetest {
  public class DecOctConversion {
    public static void Main(string[] args){
          int n = 2468;
         Stack<int> s = new Stack<int>(20);
         Console.Write("十进制数: {0} -> 八进制:",n);
         while (n!=0) {
               s.Push(n%8);
               n = n/8;
         }
          int i = s.Count;
         while (i>0){
               Console.Write(s.Pop()); i--;
         }
         Console.WriteLine();
    } } }
```

4.0 队列的概念及类型定义

- 队列(queue)是一种特殊的线性数据结构，其插入和删除操作分别在表的两端进行，是一种“先进先出”（First In First Out，FIFO）的线性结构。
- 在计算机系统中，如果多个进程需要使用某个资源，它们就要排队等待该资源的就绪。此类问题的求解，需要用到队列数据结构。
- 向队列中插入元素的操作称为**入队**(enqueue)，删除元素的操作称为**出队**（dequeue）。允许入队的一端为**队尾**（rear），允许出队的一端为**队头**（front）。



4.1 C#中的泛型队列类Queue<T>

- 在System.Collections.Generics命名空间中定义了一个泛型队列类Queue，提供了一种数据先进先出的集合。
- Queue类的属性和方法：

公共构造函数

- Queue<T>(); //初始化Queue类的新实例
- Queue<T>(ICollection c);
- Queue<T>(int capacity);

公共属性

- virtual int Count {get;} //获取包含在队列中的元素数

```
Queue<int> q = new Queue<int> ();
q.Enqueue(14);   // √
var p = new Queue<string> ();
p.Enqueue(14);   // ×
int  i = q.Count;
```


Queue的公共方法

- virtual void Enqueue(T d);
  //将对象添加到Queue的结尾
- virtual T Dequeue ();
  //移除并返回位于Queue开始处的对象
- virtual T Peek();
  //返回队头处的对象但不将其移除。
- virtual bool Contains(T x);
  //确定某个元素是否在队列中

```
Queue<string> q = new Queue<string>();
q.Enqueue("WHU");
```


【例3.5】 创建Queue对象，向其添加值并打印出其值

```
using System; using System.Collections.Generics;
Queue myQ<string> = new Queue<string>();
myQ.Enqueue("Hello"); myQ.Enqueue("World");
myQ.Enqueue("!");
Console.WriteLine( "myQ" );
Console.WriteLine( "\tCount: {0}", myQ.Count );
Console.Write( "\tValues: " );
foreach(string o in  myQ)
    Console.Write( "{0}\t", o);
Console.WriteLine();
```

程序运行结果

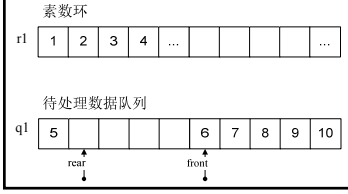
```
myQ
    Count:    3
    Values: Hello    World    !
```

输出序列的顺序与入队的顺序相同，这是队列的先进先出（FIFO）特性的体现。

[例3.8] 解素数环问题

试探法

- 创建一个线性表对象r1存放素数环的数据元素，创建一个队列对象q1，存放待检测的数据元素。方法IsPrime(k) 判断k是否为素数。
- 首先将2~n的数全部入队q1，将出队数据k与素数环最后一个数据元素相加，若两数之和是素数，则将k加入到素数环r1中，否则说明k暂时无法处理，必须再次入队等待。重复上述操作，直到队列q1为空。



```
ring1.Add(1);      // 初始化素数环，1就位
for(i=2;i<=n;i++) //  2—n 全部入队q1
  q1.Enqueue(i);
i = 0;
while(!q1.Empty){
  k = q1.Dequeue(); //出队
  Console.Write("Dequeue: " + k + "\t");
  j = ring1[i] + k;
  if(IsPrime(j)){   //判断j是否为素数
    ring1.Add(k);   //k添加到素数环中
    Console.Write("add into ring\t");
    i++;
  } else{
    q1.Enqueue(k);  //k再次入队
    Console.Write("wait again\t");
  }
  q1.Show(true); }
```

5. 哈希表：Dictionary

- Dictionary是表示<键，值>对(Key-Value Pair)的集合的类，这些<键，值>对根据键的哈希码进行组织。它们的元素可以直接通过键来索引。通过键来检索值的速度非常快，时间效率接近于O(1)。如下例中用“bmp”可以得到它的值“paint.exe”。

```
using System; using System.Collections.Generic;
public static void Main() {
  Dictionary<string, string> openWith =
        new Dictionary<string, string>();
  openWith.Add("txt","notepad.exe");
  openWith.Add("bmp", "paint.exe");
  openWith["doc"] = "winword.exe";
  if (!openWith.ContainsKey("ppt")) {
    Console.WriteLine("Key \"ppt\" is not found.");
  } }
```


6.0 串的概念及类型定义

- 串是由n（n≥0）个字符a_0，a_1，a_2，…，a_{n-1}组成的有限数据序列。元素是单个字符。
- 串记作：String = { a_0，a_1，a_2，…，a_{n-1}}
- 串中所能包含的字符依赖于所使用的字符集。C#中字符(char)采用16位Unicode编码，而非8位的ASCII编码，能处理包括中文在内的字符。
- C#中用单引号将字符括起来，用双引号将串括起来。例如，

```
s1 = "C#"                          //串长度为2
s2 = "data structure in C#" //串长度为20
s3 = ""    //空串，长度为0
s4 = " "   //两个空格的串，长度为2
```


6.1 字符及字符串的比较

- 每个字符根据所使用的字符集会有一个特定的编码，目前常用的字符集编码有ASCII码（8位），而C#采用的是Unicode码（16位）。
- 不同的字符在字符集中是按顺序排列编码的，字符可以按其编码次序规定它的大小，因此两个字符/字符串可以进行比较。例如：

'A' < 'a'	比较结果为true
'A' < '9'	比较结果为false
"data"<"date"	比较结果为true

char类型和string类型都是可比较的（comparable）

子串及子串在主串中的序号

- 由串中若干个连续的字符组成的一个子序列称为该串的一个子串（substring）；原串称为子串的主串。
  - 空串是任何串的子串
  - 一个串s也可看成是自身的子串
  - 除主串本身外，它的其他子串都称为真子串
- 串s中的某个字符c的位置用其位置序号整数表示，称为字符c在串s中的序号（index）。串的第一个字符的序号为0。如果串不包含字符c，则称c在s中的序号为-1。
- 子串的序号是该子串的第一个字符在主串中的序号。例如，s1在s2中的序号为19。如果串sub不是串mainstr的子串，则称sub在mainstr中的序号为-1。

4.2 C#中的串类：String和StringBuild

- 在System命名空间中定义了一个串类String。C#的关键字string类型是System.String类的别名。
- 在System.Text命名空间中还定义了另一个串类StringBuilder。
- 一个String对象一旦创建就不能再修改其内容。StringBuilder类的对象在创建后可以进行修改。
- String类的构造函数：

```
String(char[ ]); //初始化String类的新实例
String(char[] cs, int startIndex, int length);
String(char c, int count);
```

```
char[ ] ca = {'c', 'h' , 'i' , 'n', 'a'};
String s1= new String(ca); String s2="China";
```

String的公共属性和方法

- int Length {get;} //获取串中的字符个数
- char this[int index] {get;}//获取串中指定位置的字符
- int CompareTo(string b);
  //将此实例与指定的 String 对象进行比较
- static int Compare(string strA, string strB);
  //比较两个指定的 String 对象
- string Substring(int startIndex, int length);
  //返回特定的子字符串

```
i= s.Length; c = s[4]; String.Compare( “China”, “Chinese”);
```


String的公共属性和方法(II)

- int IndexOf(char c);
  //返回指定字符在串中的第一个匹配项的索引
- int IndexOf(string str);
  //返回指定串在此实例中的第一个匹配项的索引
- string Insert(int startIndex, string sub);
  //在指定位置插入指定的 String 实例
- string Remove(int startIndex, int count);
  //从指定位置开始删除指定数目的字符
- string Replace(string oldstr, string newstr);
  //将指定子串的所有匹配项替换为指定子串

【例4.1】从身份证号码中提取出生年月日信息

```
using System;
namespace stringstest {
  class substringtest {
    static void Main(string[] args) {
      string id = "420100199012311234";
      int y = int.Parse(id.Substring(6, 4));
      int m = int.Parse(id.Substring(10,2));
      int d = int.Parse(id.Substring(12,2));
      DateTime dt = new DateTime(y,m,d);
      Console.WriteLine(dt.Year);
      Console.WriteLine(dt.Month);
      Console.WriteLine(dt.Day);     } } }
```