.Net Framework支持5种常用集合类型：

1. 非泛型的
2. 专用的
3. 基于位的
4. 泛型的
5. 并行的

非泛型的集合：包括动态数组、栈和队列，也包括字典等。非泛型集合操作的是object类型，能够用来存储任何类型的数据，不同类型的数据也可以混合存储在同一个集合中。非泛型集合不是类型安全的。

专用结合用于操作特殊类型的数据，或者以独特的方式执行操作。

BitArray是一个基于位 集合，支持位运算。

泛型集合是标准数据结构的泛型实现，都是类型安全的，只有与集合类型兼容的类型才能存储在集合。

并行集合支持多线程访问。

C#的集合类在本质上与C++ STL模板很类似。

非泛型集合：

ArrayList类：支持动态数组。

例：程序collection\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 非泛型集合：ArrayList

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test1

{

class ArrayListTest

{

static void Main(string[] args)

{

// 创建一个array list

ArrayList array = new ArrayList();

Console.WriteLine("Initial number of elements: " + array.Count);

Console.WriteLine();

array.Add('C');

array.Add('A');

array.Add('E');

array.Add('B');

array.Add('F');

// 添加不同类型的元素

array.Add(5);

Console.Write("Current contents: ");

for (int i = 0; i < array.Count; ++i)

Console.Write(array[i] + " ");

Console.WriteLine("Removing 2 elments");

array.Remove('F');

array.Remove('A');

Console.Write("Current contents: ");

for (int i = 0; i < array.Count; ++i)

Console.Write(array[i] + " ");

Console.WriteLine();

}

}

}

输出结果：

Initial number of elements: 0

Current contents: C A E B F 5

Removing 2 elments

Current contents: C E B 5

ArrayList排序和搜索

例：程序collection\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 非泛型类：ArrayList

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test2

{

class ArrayListTest

{

static void Main(string[] args)

{

ArrayList array = new ArrayList();

array.Add(55);

array.Add(43);

array.Add(-4);

array.Add(88);

array.Add(3);

array.Add(19);

Console.Write("Original contents: ");

foreach (var i in array)

Console.Write(i + " ");

Console.WriteLine("\n");

// 数组排序

// 集合中的所有元素必须能够相互比较

// 否则会抛出异常

array.Sort();

Console.Write("Contents after sorting: ");

foreach (var i in array)

Console.Write(i + " ");

Console.WriteLine("\n");

// 搜索

Console.WriteLine("Index of 43 is " + array.BinarySearch(43));

// 将ArrayList转换为数组

int[] iarray = (int[])array.ToArray(typeof(int));

}

}

}

输出结果：

Original contents: 55 43 -4 88 3 19

Contents after sorting: -4 3 19 43 55 88

Index of 43 is 3

Hashtable类：使用散列表作为存储器。

散列表：使用哈希算法机制存储信息。执行查找、检索和设置等操作花费的时间保持不变。

Hashtable存储“键/值”。

Hashtable不能确保它的元素按顺序排列。

例：程序collecton\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// Hashtable

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test3

{

class HashtableTest

{

static void Main(string[] args)

{

Hashtable hash\_table = new Hashtable();

hash\_table.Add("house", "Dwelling");

hash\_table.Add("car", "Means of transport");

hash\_table.Add("book", "Collection of printed words");

hash\_table.Add("apple", "Edible fruit");

hash\_table["tractor"] = "Farm implement";

// 获取键

ICollection keys = hash\_table.Keys;

foreach (string str in keys)

Console.WriteLine(str + " : " + hash\_table[str]);

}

}

}

输出结果：

apple : Edible fruit

car : Means of transport

book : Collection of printed words

tractor : Farm implement

house : Dwelling

从结果中可以看出，Hashtable的“键/值”不是已排序的顺序进行存储。

SortedList类：存储“键/值”对的集合，该集合中的元素参照键的值顺序存储。

例：程序collection\_test4

// Copyright 2016.刘珅珅

// author：刘珅珅

// SortedList

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test4

{

class SortedListTest

{

static void Main(string[] args)

{

SortedList sorted\_list = new SortedList();

sorted\_list.Add("house", "Dwelling");

sorted\_list.Add("car", "Means of transport");

sorted\_list.Add("book", "Collection of printed words");

sorted\_list.Add("apple", "Edible fruit");

sorted\_list["tractor"] = "Farm implement";

ICollection keys = sorted\_list.Keys;

Console.WriteLine("Contents of list via indexer.");

// SortedList中的元素是按键排序的

foreach (string str in keys)

Console.WriteLine(str + " : " + sorted\_list[str]);

Console.WriteLine();

Console.WriteLine("Contents by integer indexed.");

for (int i = 0; i < sorted\_list.Count; ++i)

Console.WriteLine(sorted\_list.GetByIndex(i));

}

}

}

输出结果：

Contents of list via indexer.

apple : Edible fruit

book : Collection of printed words

car : Means of transport

house : Dwelling

tractor : Farm implement

Contents by integer indexed.

Edible fruit

Collection of printed words

Means of transport

Dwelling

Farm implement

Stack类：栈

例：程序collection\_test5

// Copyright 2016.刘珅珅

// author：刘珅珅

// Stack：栈

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test5

{

class StackTest

{

static void ShowPush(Stack st, int a)

{

st.Push(a);

Console.WriteLine("Push({0})", a);

Console.Write("stack: ");

foreach (int i in st)

Console.Write(i + " ");

Console.WriteLine();

}

static void ShowPop(Stack st)

{

Console.Write("Pop -> ");

int a = (int)st.Pop();

Console.WriteLine(a);

Console.Write("stack: ");

foreach (int i in st)

Console.Write(i + " ");

Console.WriteLine();

}

static void Main(string[] args)

{

Stack stack = new Stack();

ShowPush(stack, 22);

ShowPush(stack, 65);

ShowPop(stack);

ShowPop(stack);

try

{

// 栈弹出元素

// 如果为空，抛出异常

ShowPop(stack);

} catch (InvalidOperationException)

{

Console.WriteLine("Stack empty.");

}

}

}

}

输出结果：

Push(22)

stack: 22

Push(65)

stack: 65 22

Pop -> 65

stack: 22

Pop -> 22

stack:

Pop -> Stack empty.

Queue：队列

例：程序collection\_test6

// Copyright 2016.刘珅珅

// author：刘珅珅

// Queue：队列

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test6

{

class QueueTest

{

static void ShowEnq(Queue q, int a)

{

q.Enqueue(a);

Console.WriteLine("Enqueue(" + a + ")");

Console.Write("queue: ");

foreach (int i in q)

Console.Write(i + " ");

Console.WriteLine();

}

static void ShowDeq(Queue q)

{

Console.Write("Dequeue -> ");

int a = (int)q.Dequeue();

Console.WriteLine(a);

Console.Write("queue: ");

foreach (int i in q)

Console.Write(i + " ");

Console.WriteLine();

}

static void Main(string[] args)

{

Queue q = new Queue();

ShowEnq(q, 22);

ShowEnq(q, 33);

ShowEnq(q, 44);

ShowDeq(q);

ShowDeq(q);

ShowDeq(q);

try

{

ShowDeq(q);

} catch (InvalidOperationException)

{

Console.WriteLine("Queue empty.");

}

}

}

}

输出结果：

Enqueue(22)

queue: 22

Enqueue(33)

queue: 22 33

Enqueue(44)

queue: 22 33 44

Dequeue -> 22

queue: 33 44

Dequeue -> 33

queue: 44

Dequeue -> 44

queue:

Dequeue -> Queue empty.

BitArray：存储位

例：程序collection\_test7

// Copyright 2016.刘珅珅

// author：刘珅珅

// BitArray：位存储

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test7

{

class BitArrayTest

{

static void ShowBits(string rem, BitArray bits)

{

Console.WriteLine(rem);

for (int i = 0; i < bits.Count; ++i)

Console.Write("{0, -6}", bits[i]);

Console.WriteLine("\n");

}

static void Main(string[] args)

{

BitArray ba1 = new BitArray(8);

byte[] b = { 67};

BitArray ba2 = new BitArray(b);

ShowBits("Original contents of ba1: ", ba1);

ba1 = ba1.Not();

ShowBits("Contents of ba1 after Not:", ba1);

// 67的二进制补码为：01000011

// 输出为True True False False False False True False

// 从低位开始输出

ShowBits("Contents of ba2:", ba2);

}

}

}

输出结果：

Original contents of ba1:

False False False False False False False False

Contents of ba1 after Not:

True True True True True True True True

Contents of ba2:

True True False False False False True False

泛型集合：

List<T>：泛型的动态数组

例：程序collection\_test8

// Copyright 2016.刘珅珅

// author：刘珅珅

// 泛型动态数组

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test8

{

class List

{

static void Main(string[] args)

{

Queue<char> queue = new Queue<char>();

queue.Enqueue('a');

queue.Enqueue('b');

queue.Enqueue('c');

// 可以用队列Queue<T>给List传参

// Queue<T>实现了IEnumerable<T>

List<char> list = new List<char>(queue);

Console.Write("Current contents: ");

for (int i = 0; i < list.Count; ++i)

Console.Write(list[i] + " ");

Console.WriteLine("\n");

Console.WriteLine("Removing 1 element.");

list.Remove('c');

Console.Write("Contents: ");

foreach (char c in list)

Console.Write(c + " ");

Console.WriteLine();

}

}

}

输出结果：

Current contents: a b c

Removing 1 element.

Contents: a b

LinkedList<T>：泛型双向链表

例：程序collection\_test9

// Copyright 2016.刘珅珅

// author：刘珅珅

// 泛型双向链表

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test9

{

class LinkedListTest

{

static void Main(string[] args)

{

LinkedList<char> linked\_list = new LinkedList<char>();

linked\_list.AddFirst('A');

linked\_list.AddFirst('B');

linked\_list.AddFirst('C');

linked\_list.AddFirst('D');

linked\_list.AddFirst('E');

// 结点方式

Console.Write("Display contents by following links: ");

for (LinkedListNode<char> node = linked\_list.First; node != null; node = node.Next)

Console.Write(node.Value + " ");

Console.WriteLine("\n");

Console.Write("Display contents with foreach loop: ");

foreach (char c in linked\_list)

Console.Write(c + " ");

Console.WriteLine("\n");

// 倒序显示

Console.Write("Display contents backwards: ");

for (LinkedListNode<char> node = linked\_list.Last; node != null; node = node.Previous)

Console.Write(node.Value + " ");

Console.WriteLine("\n");

linked\_list.AddLast('K');

Console.Write("Contents after addition to end: ");

foreach (char c in linked\_list)

Console.Write(c + " ");

Console.WriteLine("\n");

}

}

}

输出结果：

Display contents by following links: E D C B A

Display contents with foreach loop: E D C B A

Display contents backwards: A B C D E

Contents after addition to end: E D C B A K

Dictionary<TKey, TValue>：泛型字典类，非泛型的Hashtable类

Dictionary不能添加相同的TKey

Dictionary可以预先指定容量。

例：程序collection\_test10

// Copyright 2016.刘珅珅

// author：刘珅珅

// 泛型字典类：Dictionary

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test10

{

class DictionaryTest

{

static void Main(string[] args)

{

Dictionary<string, double> dict = new Dictionary<string, double>();

dict.Add("Bulter, John", 730);

dict.Add("Swartz, Sarah", 590);

dict.Add("Pyke, Thoms", 450);

// 不能添加相同的key值

// dict.Add("Bulter, John", 730);

ICollection<string> keys = dict.Keys;

foreach (string str in keys)

Console.WriteLine("{0}, Salary: {1:C}", str, dict[str]);

}

}

}

SortedDictionary<TKey, TValue>：按键排序的字典

SortedList<TKey, TValue>：已排序的“键/值”列表，非泛型类SortedList

SortedList<TKey, TValue>可以预先指定容量

Stack<T>：泛型栈，非泛型类Stack

Stack<T>可以预先指定容量

Queue<T>：泛型队列，非泛型类Queue

Queue<T>可以预先指定容量

HashSet<T>：支持实现集的集合，使用散列表存储。

HashSet<T>所有元素都唯一，不允许出现重复元素，不指定元素的顺序。

例：程序collection\_test11

// Copyright 2016.刘珅珅

// author：刘珅珅

// HashSet：散列集

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test11

{

class HashSetTest

{

static void Show(string msg, HashSet<char> set)

{

Console.Write(msg);

foreach (char c in set)

Console.Write(c + " ");

Console.WriteLine();

}

static void Main(string[] args)

{

HashSet<char> hashSet1 = new HashSet<char>();

HashSet<char> hashSet2 = new HashSet<char>();

hashSet1.Add('A');

hashSet1.Add('B');

hashSet1.Add('C');

hashSet2.Add('C');

hashSet2.Add('D');

hashSet2.Add('E');

Show("Initial content of hashSet1: ", hashSet1);

Show("Initial content of hashSet2: ", hashSet2);

// 两个集合的不同元素

hashSet1.SymmetricExceptWith(hashSet2);

Show("hashSet1 after Symmetric difference with hashSet2: ", hashSet1);

// 两个集合的并集

hashSet1.UnionWith(hashSet2);

Show("hashSet1 after union with hashSet2: ", hashSet1);

// 一个集合剔除掉另一个集合

hashSet1.ExceptWith(hashSet2);

Show("hashSet1 after subtracting with hashSet2: ", hashSet1);

}

}

}

输出结果：

Initial content of hashSet1: A B C

Initial content of hashSet2: C D E

hashSet1 after Symmetric difference with hashSet2: A B D E

hashSet1 after union with hashSet2: A B D E C

hashSet1 after subtracting with hashSet2: A B

并行集合：可以安全的用于多线程编程。

BlockingCollection<T>

ConcurrentBag<T>

ConcurrentDictionary<TKey, TValue>：并行字典

ConcurrentQueue<T>：并行队列

ConcurrentStack<T>：并行栈

例：程序collection\_test12

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace collection\_test12

{

class BlockingCollectionTest

{

static BlockingCollection<char> blocking\_collection;

// 生产者：生产A-Z

static void Producer()

{

for (char c = 'A'; c <= 'Z'; ++c)

{

blocking\_collection.Add(c);

Console.WriteLine("Producing " + c);

}

blocking\_collection.CompleteAdding();

}

// 消费者

static void Consumer()

{

char c;

while (!blocking\_collection.IsCompleted)

{

if (blocking\_collection.TryTake(out c))

Console.WriteLine("Consuming " + c);

}

}

static void Main(string[] args)

{

// 阻塞前，集合可存储的最大对象数为4

blocking\_collection = new BlockingCollection<char>(4);

Task producer = new Task(Producer);

Task consumer = new Task(Consumer);

// 任务开始

consumer.Start();

producer.Start();

try

{

Task.WaitAll(producer, consumer);

} catch (AggregateException e)

{

Console.WriteLine(e);

}

finally

{

producer.Dispose();

consumer.Dispose();

blocking\_collection.Dispose();

}

}

}

}

在集合中存储用户自定义的类

在非泛型集合实现IComparable比较接口

例：程序collection\_test13

// Copyright 2016.刘珅珅

// author：刘珅珅

// 非泛型集合中自定义类的比较

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test13

{

class Inventory : IComparable

{

string name;

double cost;

int onhand;

public Inventory(string n, double c, int h)

{

name = n;

cost = c;

onhand = h;

}

public override string ToString()

{

return String.Format("{0,-10}Cost: {1, 6:C} On hand: {2}",

name, cost, onhand);

}

// 实现IComparable接口

public int CompareTo(object obj)

{

Inventory b = (Inventory)obj;

return name.CompareTo(b.name);

}

}

class IComparableTest

{

static void Main(string[] args)

{

ArrayList array = new ArrayList();

array.Add(new Inventory("Pliers", 5.95, 3));

array.Add(new Inventory("Wrenches", 8.29, 2));

array.Add(new Inventory("Hammers", 3.50, 4));

array.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("Inventory list before sorting:");

foreach (Inventory i in array)

Console.WriteLine(" " + i);

Console.WriteLine();

// 排序

array.Sort();

Console.WriteLine("Inventory list after sorting:");

foreach (Inventory i in array)

Console.WriteLine(" " + i);

}

}

}

输出结果为：

Inventory list before sorting:

Pliers Cost: ￥5.95 On hand: 3

Wrenches Cost: ￥8.29 On hand: 2

Hammers Cost: ￥3.50 On hand: 4

Drills Cost: ￥19.88 On hand: 8

Inventory list after sorting:

Drills Cost: ￥19.88 On hand: 8

Hammers Cost: ￥3.50 On hand: 4

Pliers Cost: ￥5.95 On hand: 3

Wrenches Cost: ￥8.29 On hand: 2

为泛型集合实现IComparable<T>接口

例：程序collection\_test14

// Copyright 2016.刘珅珅

// author：刘珅珅

// 泛型比较接口：IComparable<T>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test14

{

class Inventory : IComparable<Inventory>

{

string name;

double cost;

int onhand;

public Inventory(string n, double c, int h)

{

name = n;

cost = c;

onhand = h;

}

public override string ToString()

{

return String.Format("{0,-10}Cost: {1, 6:C} On hand: {2}",

name, cost, onhand);

}

// 实现IComparable<Inventory>接口

public int CompareTo(Inventory obj)

{

return name.CompareTo(obj.name);

}

}

class IComparable\_T

{

static void Main(string[] args)

{

List<Inventory> list = new List<Inventory>();

list.Add(new Inventory("Pliers", 5.95, 3));

list.Add(new Inventory("Wrenches", 8.29, 2));

list.Add(new Inventory("Hammers", 3.50, 4));

list.Add(new Inventory("Drills", 19.88, 8));

Console.WriteLine("Inventory list before sorting:");

foreach (Inventory i in list)

Console.WriteLine(" " + i);

Console.WriteLine();

// 排序

list.Sort();

Console.WriteLine("Inventory list after sorting:");

foreach (Inventory i in list)

Console.WriteLine(" " + i);

}

}

}

输出结果为：

Inventory list before sorting:

Pliers Cost: ￥5.95 On hand: 3

Wrenches Cost: ￥8.29 On hand: 2

Hammers Cost: ￥3.50 On hand: 4

Drills Cost: ￥19.88 On hand: 8

Inventory list after sorting:

Drills Cost: ￥19.88 On hand: 8

Hammers Cost: ￥3.50 On hand: 4

Pliers Cost: ￥5.95 On hand: 3

Wrenches Cost: ￥8.29 On hand: 2

枚举器：

枚举器相对于集合是只读的，不能通过枚举器更改正在列举的集合。

对正在列举的集合执行任何改动都会使枚举器无效。

例：程序collection\_test15

// Copyright 2016.刘珅珅

// author：刘珅珅

// 枚举器

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test15

{

class EnumeratorTest

{

static void Main(string[] args)

{

ArrayList list = new ArrayList();

for (int i = 0; i < 10; ++i)

list.Add(i);

// 使用枚举器

IEnumerator etr = list.GetEnumerator();

while (etr.MoveNext())

Console.Write(etr.Current + " ");

Console.WriteLine();

// 重置枚举器

etr.Reset();

while (etr.MoveNext())

Console.Write(etr.Current + " ");

Console.WriteLine();

}

}

}

输出结果为：

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

IDcitionaryEnumerator：“键/值”枚举器

例：程序collection\_test16

// Copyright 2016.刘珅珅

// author：刘珅珅

// 字典枚举器：IDictionaryEnumerator

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test16

{

class DictionaryEnumeratorTest

{

static void Main(string[] args)

{

Hashtable hash = new Hashtable();

hash.Add("Tom", "555-3456");

hash.Add("Mary", "555-9876");

hash.Add("Todd", "555-3452");

hash.Add("Ken", "555-7756");

// 字典枚举器

IDictionaryEnumerator etr = hash.GetEnumerator();

Console.WriteLine("Display info using Entry.");

while (etr.MoveNext())

Console.WriteLine(etr.Entry.Key + " : " + etr.Entry.Value);

Console.WriteLine();

Console.WriteLine("Display info using Key and Value directly.");

etr.Reset();

while (etr.MoveNext())

Console.WriteLine(etr.Key + " : " + etr.Value);

}

}

}

输出结果：

Display info using Entry.

Tom : 555-3456

Todd : 555-3452

Ken : 555-7756

Mary : 555-9876

Display info using Key and Value directly.

Tom : 555-3456

Todd : 555-3452

Ken : 555-7756

Mary : 555-9876

如果希望自定义的类能够使用foreach循环列举这些对象，该类必须实现IEnumerator接口和IEnumerable接口的泛型或非泛型形式。

例：程序collection\_test17

// Copyright 2016.刘珅珅

// author：刘珅珅

// IEnumerator和IEnumerable

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test17

{

class MyClass : IEnumerable, IEnumerator

{

char[] chrs = { 'A', 'B', 'C', 'D'};

int index = -1;

// 实现IEnumerable

public IEnumerator GetEnumerator()

{

return this;

}

// 实现IEnumerator

public object Current

{

get { return chrs[index]; }

}

// 实现IEnumerator

public bool MoveNext()

{

if (index == chrs.Length - 1)

{

Reset();

return false;

}

++index;

return true;

}

// 实现IEnumerator

public void Reset() { index = -1; }

}

class EnumTest

{

static void Main(string[] args)

{

MyClass obj = new MyClass();

// 使用foreach循环遍历

foreach (char c in obj)

Console.Write(c + " ");

Console.WriteLine();

}

}

}

foreach循环会自动调用MoveNext()方法和Reset()方法。

迭代器：yield，每次返回一个元素，同Python中的yield关键字有相似之处。

迭代器遍历，类不需要指定IEnumerable和IEnumerator接口

例：程序collection\_test18

// Copyright 2016.刘珅珅

// author：刘珅珅

// 迭代器

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test18

{

class MyClass

{

char c = 'A';

// 隐式实现IEnumerable

public IEnumerator GetEnumerator()

{

// 迭代器

// yield每次返回一个元素

for (int i = 0; i < 5; ++i)

{

Console.Write("GetEnumerator {0} ", i);

yield return (char)(c + i);

}

}

}

class YieldTest

{

static void Main(string[] args)

{

MyClass obj = new MyClass();

foreach (char c in obj)

Console.WriteLine(" " + c );

Console.WriteLine();

}

}

}

输出结果：

GetEnumerator 0 A

GetEnumerator 1 B

GetEnumerator 2 C

GetEnumerator 3 D

GetEnumerator 4 E

从结果中可以看出，yield return返回后，下一次执行从之后的代码开始执行，和Python中生成器的yield类似。

yield break; 停用迭代器

使用多个yield。

例：程序collection\_test19

// Copyright 2016.刘珅珅

// author：刘珅珅

// 多个迭代器yield

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test19

{

class MyClass

{

public IEnumerator GetEnumerator()

{

yield return 'A';

yield return 'B';

yield return 'C';

yield return 'D';

yield return 'E';

}

}

class YieldTest

{

static void Main(string[] args)

{

MyClass obj = new MyClass();

foreach (char c in obj)

Console.WriteLine(c + " ");

Console.WriteLine();

}

}

}

输出结果：

A

B

C

D

E

从结果可以看出，yield return执行完后，下一次从它之后的代码处开始执行。

命名迭代器：

public IEnumerble itr-name(param-list) {

Yield return obj;

}

命名迭代器允许为迭代器传递参数。

例：程序collection\_test20

// Copyright 2016.刘珅珅

// author：刘珅珅

// 命名迭代器

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test20

{

class MyClass

{

char c = 'A';

// 命名迭代器

public IEnumerable MyIterator(int begin, int end)

{

for (int i = begin; i < end; ++i)

yield return (char)(c + i);

}

}

class NamedYieldTest

{

static void Main(string[] args)

{

MyClass obj = new MyClass();

Console.WriteLine("Iterate letters from F to L:");

foreach (char c in obj.MyIterator(5, 12))

Console.Write(c + " ");

Console.WriteLine();

}

}

}

输出结果：

Iterate letters from F to L:

F G H I J K L

创建泛型迭代器：

例：程序collection\_test21

// Copyright 2016.刘珅珅

// author：刘珅珅

// 泛型迭代器

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace collection\_test21

{

class MyClass<T>

{

T[] array;

public MyClass(T[] a)

{

array = a;

}

// 泛型迭代器

public IEnumerator<T> GetEnumerator()

{

foreach (T obj in array)

yield return obj;

}

}

class GenericYieldTest

{

static void Main(string[] args)

{

int[] nums = { 1, 2, 3, 4, 5 };

MyClass<int> obj1 = new MyClass<int>(nums);

foreach (var x in obj1)

Console.Write(x + " ");

Console.WriteLine();

bool[] bVals = { true, true, false, false};

MyClass<bool> obj2 = new MyClass<bool>(bVals);

foreach (var b in obj2)

Console.Write(b + " ");

Console.WriteLine();

}

}

}

输出结果：

1 2 3 4 5

True True False False

集合初始化器：

List<char> list = new List<char>() {'A', 'B', 'C'};

SortedList<int, string> list = new SortedList<int, string>() {{1, "One"},{2, "Two"},{3, "Three"}};