Работа с массивами

admin
December 2021 edited February 10

Любой массив - это объект ссылочного типа данных. Размер массива неизменен. В OverScript поддерживаются только одномерные массивы.
Ниже рассмотрим функции для добавления и удаления элементов массива. Но нужно понимать, что функции, которые меняют размер массива, на самом деле создают каждый раз новый массив. Создание массива - ресурсоемкая операция, поэтому в C# нет таких функций, а есть тип List. Тем не менее я решил сделать функции для добавления и удаления элементов, а использовать их или нет - дело ваше.

Insert и ConvInsert

```
long[] arr={10, 20, 30}; //правильнее было бы: {10L, 20L, 30L}, но и так сработает
arr.Insert(1, 15L); //вставка по индексу 1. Тут, если написать 15, а не 15L, то будет ошибка!
WriteLine(Join(' ', arr)); //10 15 20 30
int c=arr.Insert(0, 3L, 5L); //вставка двух значений в начало. Можно и больше двух вставлять.
WriteLine(Join(' ', arr)+"; Новый размер: "+c); //3 5 10 15 20 30; Новый размер: 6
arr.ConvInsert(c, 40, "50"); //вставка в конец двух значений с конвертированием их в нужный тип
WriteLine(Join(' ', arr)); //3 5 10 15 20 30 40 50
```

При вставке в массив своего типа проверка типа вставляемых экземпляров не делается.

```
Foo[] arr=new Foo[0];
arr.Insert(0, new Foo(), new Bar());
WriteLine(Join(", ", arr)); //Instance of Foo, Instance of Bar
class Foo{}
class Bar{}
```

Вставлять можно в массивы любых типов:

```
object arr=Array("System.UInt64", 10, 20, 30); //функция Array создаёт массив заданного типа. Вместо "System.UInt64" лучше писать typeof("System.UInt64"), чтобы тип не искался пр
arr.ConvInsert(2, 777); //10 20 777 30
WriteLine(Join(' ', arr));
```

Поддерживается вставка в объекты реализующие System.Collections.IList:

```
object list=Create("System.Collections.Generic.List`1[System.Int64], System.Collections");
list.Insert(list.Length(), 10L);
```

```
list.Insert(list.Length(), 20L);
list.ConvInsert(list.Length(), 30);
WriteLine(Join(' ', list)); //10 20 30
```

InsertRange

Эта функция позволяет вставлять значения из массива.

```
long[] arr={10, 20, 30};
arr.InsertRange(1, new int[]{555, 777}); //тут int-значения копируются в long-массив, что позволяется
WriteLine(Join(' ', arr)); //10 555 777 20 30

//arr.InsertRange(0, new decimal[]{123, 789}); //выдаст ошибку, т.к. decimal нельзя запихнуть в long[]
arr.InsertRange(0, new decimal[]{123, 789}, true); //чтобы при вставке значения конвертировались, нужно добавить третий аргумент - true
WriteLine(Join(' ', arr)); //123 789 10 555 777 20 30
```

Вставка в список:

```
object list=Create("System.Collections.Generic.List`1[System.Int64], System.Collections");
list.InsertRange(0, new long[]{555, 777});
WriteLine(Join(' ', list)); //555 777
list.InsertRange(0, new decimal[]{123, 789}, true); //с конвертированием
WriteLine(Join(' ', list)); //123 789 555 777
```

Push и ConvPush

```
long[] arr=new long[]{10, 20, 30};
arr.Push(40L); //вставка в конец числа типа long
arr.ConvPush(50); //число 50 типа int сконвертируется в long
arr.Push(60L, 70L); //можно несколько значений добавлять
WriteLine(Join(' ', arr)); //10 20 30 40 50 60 70
```

В список:

```
object list=Create("System.Collections.Generic.List`1[System.Int64], System.Collections");
list.Push(10L, 20L, 30L);
list.ConvPush(40, 50);
WriteLine(Join(' ', list)); //10 20 30 40 50
```

В необобщенный стек (обобщенные не поддерживаются):

```
object stack=Create("System.Collections.Stack, System.Collections.NonGeneric");
stack.Push(10, 20L, "Hello");
WriteLine(stack.Peek()); //Hello
```

В необобщенную очередь (обобщенные не поддерживаются):

```
object queue=Create("System.Collections.Queue, System.Collections.NonGeneric");
queue.Push(10, 20L, "Hello"); //Push для очереди вызывает метод Enqueue
WriteLine(queue.Peek()); //10
```

Insert, ConvInsert, InsertRange, Push и ConvPush возвращают кол-во элементов после вставки.

Peek и Pop

Peek возвращает последнее значение в массиве, списке, стеке, и первое значение в очереди. Pop делает то же самое, но с удалением. Обобщенные стеки и очереди не поддерживаются.

```
int[] arr={10, 20, 30};
WriteLine(arr.Peek()); //30
WriteLine(arr.Pop()); //30
WriteLine(arr.Peek()); //20
WriteLine();

object list=Create("System.Collections.Generic.List`1[System.Int32], System.Collections");
list.Push(10, 20, 30);
WriteLine(list.Peek()); //30
WriteLine(list.Pop()); //30
WriteLine(list.Peek()); //20
WriteLine();

object stack=Create("System.Collections.Stack, System.Collections.NonGeneric");
stack.Push(10, 20, 30);
WriteLine(stack.Peek()); //30
WriteLine(stack.Pop()); //30
WriteLine(stack.Peek()); //20
WriteLine();

object queue=Create("System.Collections.Queue, System.Collections.NonGeneric");
queue.Push(10, 20, 30);
WriteLine(queue.Peek()); //10
WriteLine(queue.Pop()); //10 //Pop для очереди вызывает метод Dequeue
WriteLine(queue.Peek()); //20
```

At

At возвращает элемент по указанному индексу. Кроме массивов поддерживаются объекты реализующие System.Collections.IList. Можно указывать отрицательный индекс.

```
int[] arr={10, 20, 30};
WriteLine(arr.At(1)); //20
WriteLine(arr.At(-1)); //30

object arr2=new long[]{100L, 200L, 300L};
WriteLine(arr2.At(1));  //200

object list=Create("System.Collections.Generic.List`1[System.Int32], System.Collections");
list.Push(10, 20, 30);
WriteLine(list.At(1)); //20
WriteLine(list.At(-1)); //30
```

Если объект не реализует IList, а только IEnumerable, то получить элемент можно функцией ElementAt:

```
object stack=Create("System.Collections.Stack, System.Collections.NonGeneric");
stack.Push("hello", "world", "test");
WriteLine(stack.ElementAt(1)); //world
```

Поиск в массиве (в любом IList)

Поиск элемента делается функцией Find.

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
WriteLine(arr.Find(byte v, v>44)); //50
WriteLine(arr.Find(byte $v, v>99)); //0 //$ перед v - разрешение на повторное декларирование

Foo[] arr2={new Foo("hello"), new Foo("world"), new Foo("test")};
WriteLine(arr2.Find(Foo f, f.Str.EndsWith('d')).Str); //world

class Foo{
    public string Str;
    New(string s){Str=s;}
}
```

Первый аргумент (arr) - где искать (поддерживаются любые IEnumerable). Второй (byte v) - переменная, в которую записывать значение элемента на каждой итерации. Третий (v>44) - условие, при котором брать элемент.
Есть функция FindIndex, которая также ищет элемент и возвращает его индекс.

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
byte v;
WriteLine(arr.FindIndex(v, v>44)); //4
WriteLine(arr.FindIndex(v, v>99)); //-1
```

Есть FindAll, которая возвращает массив всех элементов, соответствующих условию.

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
byte[] arr2=arr.FindAll(byte v, v>20 && v<60);
WriteLine(Join(' ', arr2)); //30 40 50
```

Есть функция Select для выборки значений:

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
int[] arr2=arr.Select(byte v, int\v, v>40); //в C# было бы:  int[] arr2 = arr.Where(v => v > 40).Select(v=> (int)v).ToArray();
WriteLine(Join(' ', arr2)); //50 60 70
```

В этом примере выбираются значения больше 40 с преобразованием в int. Аргумент int\v - это значение, которое добавляется в новый массив.
Эту функцию можно использовать без условия:

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
int[] arr2=arr.Select(byte v, int\v*2);
WriteLine(Join(' ', arr2)); //20 40 60 80 100 120 140
```

Её также можно использовать для обычного конвертирования массива в другой тип, но лучше использовать ConvertArray:

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
int[] arr2:=arr.ConvertArray(int);
WriteLine(Join(' ', arr2)); //10 20 30 40 50 60 70
```

Есть ещё функция ToArray, позволяющая создать массив из IEnumerable:

```
object stack=Create("System.Collections.Stack, System.Collections.NonGeneric");
stack.Push("hello", "world", "test");
string[] arr:=stack.ToArray(string); //чтобы ToArray работала быстрее, лучше задавать размер: stack.ToArray(string, 3)
WriteLine(Join(' ',arr)); //test world hello

arr:=stack.ToArray(string, 2); //брать только два элемента
WriteLine(Join(' ',arr)); //test world
```

Удаление элементов

Удалить элемент можно функцией Remove:

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
int c=arr.Remove(20, 40, 70); //удаление элементов 20, 40, 70
WriteLine(Join(' ', arr)); //10 30 50 60
WriteLine("New length: "+c); //New length: 4
```

RemoveAt удаляет по индексу:

```
byte[] arr={10, 20, 30, 40};
int c=arr.RemoveAt(1); //удаление элемента по индексу 1
WriteLine(Join(' ', arr)); //10 30 40
WriteLine("New length: "+c); //New length: 3
```

RemoveRange удаляет указанное кол-во элементов, начиная с заданного индекса:

```
byte[] arr={10, 20, 30, 40};
int c=arr.RemoveRange(1, 2); //удаление двух элементов начиная с индекса 1
WriteLine(Join(' ', arr)); //10 40
WriteLine("New length: "+c); //New length: 2
```

RemoveAll удаляет все элементы, удовлетворяющие условию:

```
byte[] arr={10, 20, 30, 40};
int c=arr.RemoveAll(byte v, v>25); //удаление элементов, которые больше 25
WriteLine(Join(' ', arr)); //10 20
WriteLine("New length: "+c); //New length: 2
```

Все эти функции удаления поддерживают объекты реализующие интерфейс IList.

Slice и Splice

Slice возвращает новый массив, содержащий копию части исходного массива.

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
byte[] newArr=Slice(arr, 2, -2);
WriteLine(Join(' ', newArr)); //30 40 50

newArr=Slice(arr, 2);
WriteLine(Join(' ', newArr)); //30 40 50 60 70
```

```
newArr=Slice(arr, , 2);
WriteLine(Join(' ', newArr)); //10 20
```

Splice изменяет содержимое массива, удаляя существующие элементы и/или добавляя новые.

```
byte[] arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
byte[] arr2=Splice(arr, 2, 3, 33, 44, 55); //замена элементов 2, 3, 4 (30 на 33, 40 на 44, 50 на 55)
WriteLine(Join(' ', arr));  //10 20 33 44 55 60 70
WriteLine(Join(' ', arr2)); //30 40 50

arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
arr2=Splice(arr, 2, 3); //удаление трёх элементов начиная с индекса 2
WriteLine(Join(' ', arr));  //10 20 60 70
WriteLine(Join(' ', arr2)); //30 40 50

arr=new byte[]{10, 20, 30, 40, 50, 60, 70};
arr2=Splice(arr, 2, 0, 22, 222); //вставить 22 и 222 без удаления других
WriteLine(Join(' ', arr));  //10 20 22 222 30 40 50 60 70
WriteLine(Join(' ', arr2)); //ничего, т.к. массив пуст
```

Slice и Splice работают только с массивами встроенных в OverScript типов (int[], string[], double[] и т.д.).