

Индексаторы

Индексаторы

Индексатор позволяет индексировать объект, подобно массиву.

```
тип_элемента this [int индекс]
{
    get      // Аксессор для получения данных
    {
        // Возврат значения, которое определяет индекс.
    }
    set      // Аксессор для установки данных
    {
        // Установка значения, которое определяет индекс.
    }
}
```

Преимущество индексатора заключается, в частности, в том, что он позволяет полностью управлять доступом к массиву, избегая нежелательного доступа.

Индексаторы

На применение индексаторов накладываются следующие ограничения:

- ❑ Значение, выдаваемое индексатором, нельзя передавать методу в качестве параметра `ref` или `out`, поскольку в индексаторе не определено место в памяти для его хранения.
- ❑ Индексатор должен быть членом своего класса и поэтому не может быть объявлен как `static`.
- ❑ Индексаторы можно использовать с операторами **for**, **do**, **while**, но его **нельзя** использовать с оператором **foreach**

Пример 1:

```
class FailSoftArray
{
    int[] a; // ссылка на базовый массив
    public int Length; // открытая переменная длины массива
    public bool ErrFlag; // обозначает результат последней
операции
    // Построить массив заданного размера,
    public FailSoftArray(int size)
    {
        a = new int[size];
        Length = size;
    }
}
```

Пример 1:

Вспомогательный метод для проверки значения индекса

```
// Возвратить логическое значение true, если
// индекс находится в установленных границах,
private bool ok(int index)
{
    if (index >= 0 & index < Length) return true;
    return false;
}
```

Пример 1:

```
// Это индексатор для класса FailSoftArray.  
public int this[int index]  
{  
    get  
    {  
        if (ok(index))  
        {  
            ErrFlag = false;  
            return a[index];  
        }  
        else  
        {  
            ErrFlag = true;  
            return 0;  
        }  
    }  
  
    set  
    {  
        if (ok(index))  
        {  
            a[index] = value;  
            ErrFlag = false;  
        }  
        else ErrFlag = true;  
    }  
}
```

Пример 2

```
class PwrOfTwo
{
    /* Доступ к логическому массиву, содержащему степени числа 2 от 0 до 15. */
    public int this[int index]
    {
        // Вычислить и вернуть степень числа 2.
        get
        {
            if ((index >= 0) && (index < 16)) return pwr(index);
            else return -1;
        }
        // Аксессор set отсутствует.
    }
    int pwr(int p)
    {
        int result = 1;
        for (int i = 0; i < p; i++)
            result *= 2;
        return result;
    }
}
```

Методы расширения

Методы расширения

Методы расширения позволяют «добавлять» методы к существующим типам без создания нового производного типа, перекомпиляции или иного изменения исходного типа.

Методы расширения

Методы расширения — это статические методы, но они вызываются так, как если бы они были методами экземпляра расширенного типа.

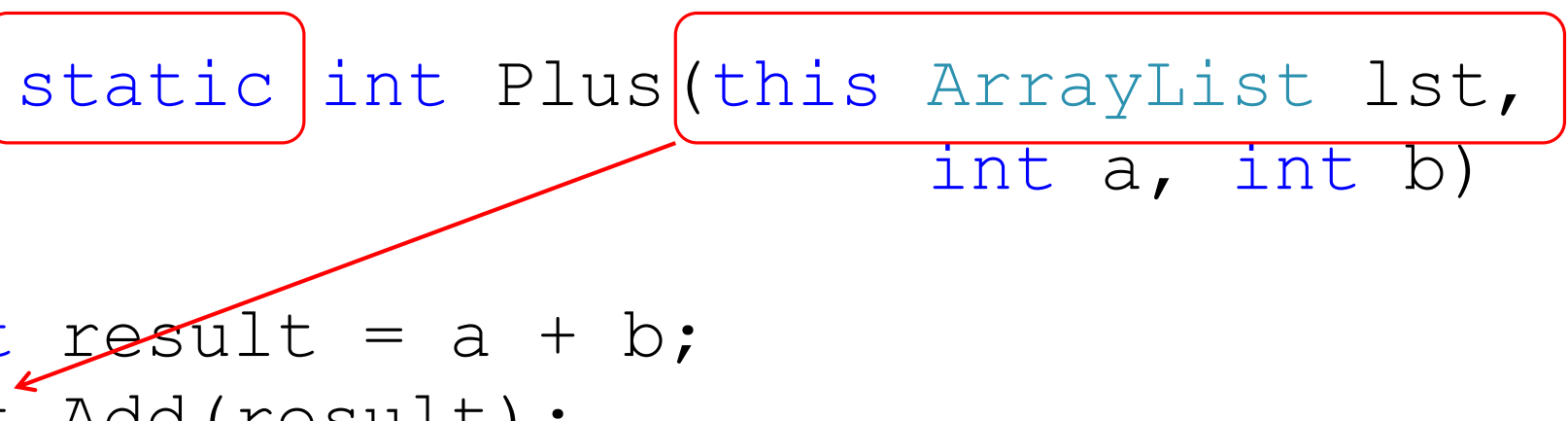
Для клиентского кода, написанного на C#, F# и Visual Basic, нет очевидной разницы между вызовом метода расширения и методов, определенных в типе.

Методы расширения

- ☐ Определите статический класс, содержащий метод расширения. Класс должен быть виден клиентскому коду.
- ☐ Реализуйте метод расширения как статический метод, по крайней мере, с той же видимостью, что и содержащий класс.
- ☐ Первый параметр метода указывает тип, с которым работает метод; перед ним должен стоять модификатор **this**.
- ☐ В вызывающем коде добавьте директиву `using`, чтобы указать пространство имен, содержащее класс метода расширения.
- ☐ Вызовите методы, как если бы они были методами экземпляра типа.

Методы расширения

```
public static class ArrayListExtensions
{
    public static int Plus(this ArrayList lst,
                           int a, int b)
    {
        int result = a + b;
        lst.Add(result);
        return result;
    }
}
```



Методы расширения

```
ArrayList al = new();  
int result = al.Plus(10, 11);  
Console.WriteLine(result);  
foreach (var item in al)  
    Console.WriteLine(item);
```

Перегрузка операторов

Перегрузка операторов

ОБЩАЯ ИНФОРМАЦИЯ

Перегрузка операторов

ПЕРЕГРУЖАЕМЫЕ ОПЕРАТОРЫ

Унарные операторы

$+x$, $-x$, $!x$, $\sim x$, $++$, $--$, `true`, `false`

Бинарные операторы

$x + y, x - y, x * y, x / y,$
 $x \% y, x \& y, x | y, x ^ y,$
 $x << y, x >> y, x == y, x != y,$
 $x < y, x > y, x <= y, x >= y$

Бинарные операторы

Операторы сравнения необходимо перегружать попарно. То есть, если один из операторов пары перегружен, другой оператор также должен быть перегружен. К таким парам относятся операторы:

`=` и `!=`

`<` и `>`

`<=` и `>=`

Условные операторы

Условные логические операторы (**&&** или **||**) не могут быть перегружены.

Однако, если тип с перегруженными операторами **true** и **false** также перегружает операторы **&** или **|**, то соответственно и операторы **&&** или **||** могут быть вычислены для операндов этого типа.

Операторы составного присваивания

(+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=)

не могут быть явно перегружены. Однако при перегрузке бинарного оператора соответствующий составной оператор присваивания, если таковой имеется, также неявно перегружается. Например, += вычисляется с помощью +, который может быть перегружен.

Перегрузка операторов

РЕАЛИЗАЦИЯ

Для объявления оператора используется ключевое слово
operator

Объявление оператора должно удовлетворять следующим правилам:

- ❑ Он включает в себя модификаторы как **public**, так и **static**.
- ❑ Унарный оператор имеет один входной параметр. Бинарный оператор имеет два входных параметра. В каждом случае хотя бы один параметр должен иметь тип **T** или **T?** где **T** - тип, содержащий объявление оператора.

Общая форма перегрузки унарного оператора:
public static T operator op (T операнд)
{ операции }

Общая форма перегрузки бинарного оператора:
public static T operator op (T операнд1, T операнд2)
{ операции }

Пример

```
class Account
{
    public decimal Deposit { get; set; }
    public void PrintInfo()
        => Console.WriteLine($"Сумма вклада: {Deposit}");

    . . .
}
```

Перегрузка бинарного оператора (пример)

```
public static Account operator +(Account obj1, Account obj2)  
    => new Account { Deposit = obj1.Deposit + obj2.Deposit };
```

Перегрузка бинарного оператора (пример)

```
public static Account operator +(Account obj, decimal value)
{
    obj.Deposit += value;
    return obj;
}
```

Перегрузка унарного оператора (пример)

```
public static Account operator ++(Account obj)
{
    obj.Deposit += 1;
    return obj;
}
```

```
var account = new Account { Deposit = 100 };  
account.PrintInfo();  
account += 20;  
account.PrintInfo();  
var account2 = new Account { Deposit = 200 };  
account = account + account2;  
account.PrintInfo();
```

Перегрузка операторов отношения

Операторы == и != должны перегружаться попарно:

```
public static bool operator ==(Account obj1, Account obj2)  
    => obj1.Deposit == obj2.Deposit;
```

```
public static bool operator !=(Account obj1, Account obj2)  
    => obj1.Deposit != obj2.Deposit;
```

Перегрузка операторов true и false

```
public static bool operator true(Account obj)  
    => obj.Deposit > 0;
```

```
public static bool operator false(Account obj)  
    => obj.Deposit <= 0;
```


Перегрузка операторов & и |

```
public static bool operator &(Account obj1, Account obj2)  
    => obj1.Deposit > 0 && obj2.Deposit > 0;
```

```
public static bool operator |(Account obj1, Account obj2)  
    => obj1.Deposit > 0 || obj2.Deposit > 0;
```

Перегрузка укороченных операторов && и ||

Для перегрузки укороченных операторов && и || требуется:

- ☐ В классе должна быть произведена перегрузка логических операторов & и |.
- ☐ Тип возвращаемого объекта и тип параметров методов операторов & и | должны быть такие же, что и у класса, для которого эти операторы перегружаются.
- ☐ Для класса должны быть перегружены операторы true и false.

Перегрузка укороченных операторов && и ||

```
public static Account operator &(Account obj1, Account obj2)
{
    if ((obj1.Deposit > 0) & (obj2.Deposit > 0))
        return new Account { Deposit = 1 };
    return new Account { Deposit = 0 };
}

public static Account operator |(Account obj1, Account obj2)
{
    if ((obj1.Deposit > 0) | (obj2.Deposit > 0))
        return new Account { Deposit = 1 };
    return new Account { Deposit = 0 };
}
```