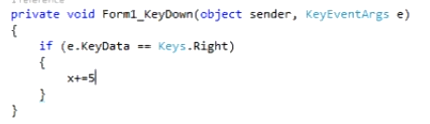
Keys



OfType

object[] values = new object[] {"1", "2", "3", "AAA", 5};  
IEnumerable<string> strings = values.OfType<string>(); // последний элемент будет пропущен, поскольку он не string

Аналог Cast<T> в SQL-похожем синтаксисе LINQ, выглядит так:

IEnumerable<string> strings = from string x in values select x;

Where<T>

object[] values = new object[] { "1", "2", "3", "AAA", 5, "ABB" };  
IEnumerable<string> strings = values.OfType<string>().Where(i => i.StartsWith(“A”)); // извлекаем все строки, которые начинаются на A

**First<T>** и **Last<T>** — извлечь первый и последний элемент перечисления. Заваливаются с исключением, если перечесление не содержит элементов. 

var values = new[] {"1", "AAA", "2", "3", "ABB"};  
IEnumerable<string> strings = values.Where(i => i.StartsWith(“A”));   
string AAA = strings.First();  
string ABB = strings.Last();

**FirstOrDefault<T>** и **LastOrDefault<T>** — извлечь первый и последний элемент перечисления. Работают даже если перечисление не содержит элементов, в этом случае возвращают default(T), что соответствует null для reference типов, 0 для числовых типов, пустым структурам для структур.  
  
**Fist<T>**, **Last<T>**, **FirstOrDefault<T>**, **LastOrDefault<T>** имеют дополнительный перегруженный вариант вызова, который принимает лямбда-функцию критерия для фильтрации.

var values = new[] { "1", "2", "3", "AAA", "ABB" };  
string AAA = values.FirstOrDefault(i => i.StartsWith("A"));

Для того, чтобы узнать, существуют ли элементы, удовлетворяющие какому-то условию, можно использовать метод **Any<T>**:

var values = new[] { "1", "2", "3", "AAA", "ABB" };  
bool hasAAA = values.Any(i => i.StartsWith(“A”)); // true

Этот же метод можно использовать для проверки перечисления на наличие элементов:

var values = new[] { "1", "2", "3", "AAA", "ABB" };  
bool hasItems = values.Any(); // true

Чтобы узнать, все ли элементы удовлетворяют какому-то условию, можно использовать метод **All<T>**:

var values = new[] { "1", "2", "3", "AAA", "ABB" };  
bool hasAAA = values.All(i => i.StartsWith("A")); // false

Посчитать количество элементов удовлетворяющие определенному критерию можно так:

var values = new[] { "1", "2", "3", "AAA", "ABB" };  
int countAAA = values.Count(i => i.StartsWith("A")); // 2

Условие можно опустить, тогда посчитаются все элементы:

int countAll = values.Count(); // 5, понятно, что в данном случае это не самый оптимальный вариант узнать размер массива :)

Очень удобно можно сортировать элементы с помощью **OrderBy<T>** и **OrderByDescending<T>**:

var values = new[] { "1", "2", "3", "AAA", "ABB" };  
IEnumerable<string> strings = values.OrderBy(i => i.Length);

Сортировать по нескольким критериям можно так:

var values = new[] { "1", "2", "3", "AAA", "ABB", "5", "6" };  
IEnumerable<string> strings = values.OrderBy(i => i.Length).ThenBy(i => i);

То же самое, но в SQL-подобном синтаксисе:

IEnumerable<string> strings = from s in values order by s.Lenght, s select s;

Удалить дубликаты из перечисления можно методом **Distinct<T>**:

var values = new[] { "1", "1", "2", "2", "3", "3", "3" };  
IEnumerable<string> strings = values.Distinct(); // "1", "2", "3"

Один из вариантов этого метода принимает Comparer<T>, т.е. можно, например, удалить дубликаты строчек без проверки на регистр:

var values = new[] { "A", "B", "b", "C", "C", "c", "C" };  
IEnumerable<string> strings = values.Distinct(StringComparer.OrdinalIgnoreCase); // "A", "B", "C"

Развернуть перечисление «к лесу передом»:

var values = new[] { "A", "B", "C", "C" };  
IEnumerable<string> strings = values.Reverse(); // "C", "C", "B", "A"

Вот вкратце некоторые примеры использования LINQ to Objects. Надеюсь, примеры достаточно красноречивы для того, чтобы возбудить ваш интерес к дальнейшему изучению и использованию LINQ to Objects…

\*\*\*

Открыть главную форму

private void button1\_Click(object sender, EventArgs e)

{

this.DialogResult = DialogResult.OK;

}

Нумерация строк в DataGridView

**private** **void** dgv\_RowPrePaint(**object** sender,

    DataGridViewRowPrePaintEventArgs e)

{

**object** head =

**this**.dgv.Rows[e.RowIndex].HeaderCell.Value;

**if** (head == **null** ||

        !head.Equals((e.RowIndex + 1).ToString()))

**this**.dgv.Rows[e.RowIndex].HeaderCell.Value =

            (e.RowIndex + 1).ToString();

}

Linq sorting

list.Sort((em1, em2) => em1.A.CompareTo(em2.A));

Func

class Program

{

static int Sum(int x, int y)

{

return x + y;

}

static void Main(string[] args)

{

Func<int,int, int> add = Sum;

int result = add(10, 10);

Console.WriteLine(result);

}

Async & await

*// Синхронная версия* **private** **void**

**OnButtonClick**() { TextBox.Text = **new** WebClient().DownloadString("http://habrahabr.ru/"); }

*// Асинхронная версия*

**private** **async** **void** **OnButtonClick**() { TextBox.Text = **await** **new** WebClient().DownloadStringTaskAsync("http://habrahabr.ru/"); }

\*\*\*

**private** **async** **void** **StartButtonClick**(**object** sender, RoutedEventArgs e) {

*// Убираем возможность повторного нажатия на кнопку* StartButton.IsEnabled = **false**; *// Вызываем новую задачу, на этом выполнение функции закончится* *// а остаток функции установится в продолжение*

TextBox.Text = **await** **new** WebClient().DownloadStringTaskAsync("http://habrahabr.ru/"); StatusLabel.Content = "Загрузка страницы завершена, начинается обработка"; *// В продолжении можно также запускать асинхронные операции со своим продолжением*

**var** result = **await** Task<**string**>.Factory.StartNew(() => { Thread.Sleep(5000); *// Имитация длительной обработки...*

**return** "Результат обработки"; }); *// Продолжение второй асинхронной операции* StatusLabel.Content = result;

StartButton.IsEnabled = **true**; }

\*\*\*

DevExpress

DevExpress.Skins.SkinManager.EnableFormSkins();

DevExpress.UserSkins.BonusSkins.Register();

\*\*\*

Byte array to string

string result = System.Text.Encoding.UTF8.GetString(byteArray);

\*\*\*

ArrayList to Int Array

(**int**[])aList.ToArray(**typeof**(**int**)

\*\*\*

int[] nums = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

int[] oddNums = Common.FilterArrayOfInts(nums, delegate(int i)

{ return ((i & 1) == 1); });

foreach (int i in oddNums)

Console.WriteLine(i);

\*\*\*

Var

var mySpouse = new { FirstName = "Alex", LastName = "Erohin" };

Инициализация обьектов

Не проще ли создавать экземпляры объектов, как показано ниже:

**Address** address = new Address

{

**address** = "105 Elm Street",

city = "Atlanta",

state = "GA",

postalCode = "30339"

};

**Инициализация коллекций**

List<**string**> presidents = **new** List<**string**> { "Adams", "Arthur", "Buchanan" };

**foreach** (**string** president **in** presidents)

{

Console.WriteLine(president);

}П

оэтому пришлось бы изменить также и тип переменной address. Здесь снова на помощь приходит ключевое слово var:

var address = **new**

{

address = “105 Elm Street”,

city = “Atlanta”,

state = “GA”,

postalCode = “30339”

};

Console.WriteLine(“address = {0} : city = {1} : state = {2} : zip = {3}”,

address.address, address.city, address.state, address.postalCode);

Console.WriteLine(“{0}”, address.GetType().ToString());

\*\*\*

**string**[] names = {

"Adams", "Arthur", "Buchanan", "Bush", "Carter", "Cleveland",

"Clinton", "Coolidge", "Eisenhower", "Fillmore", "Ford", "Garfield",

"Grant", "Harding", "Harrison", "Hayes", "Hoover", "Jackson",

"Jefferson", "Johnson", "Kennedy", "Lincoln", "Madison", "McKinley",

"Monroe", "Nixon", "Obama", "Pierce", "Polk", "Reagan", "Roosevelt",

"Taft", "Taylor", "Truman", "Tyler", "Van Buren", "Washington", "Wilson"};

*// Использование точечной нотации*

IEnumerable<**string**> sequence = names

.Where(n => n.Length < 6)

.Select(n => n);

*// Использование синтаксиса выражения запроса*

IEnumerable<**string**> sequence = **from** n **in** names

**where** n.Length < 6

**select** n;

**foreach** (**string** name **in** sequence)

{

Console.WriteLine("{0}", name);

}

Первое, что можно заметить в примере с выражением запроса — это то, что в отличие от SQL, операция from предшествует операции select. Одной из причин этого была необходимость сужения контекста для средства IntelliSense. Без такой инверсии операций ввод в текстовом редакторе Visual Studio слова "select", за которым следует пробел, не позволит средству IntelliSense определить, какие переменные должны отображаться в его раскрывающемся списке.

\*\*\*

*// Создать массив целых чисел.*

**int**[] intArray = **new** **int**[] { 1, 2, 3 };

IEnumerable<**int**> ints = intArray.Select(i => i);

**foreach** (**int** i **in** ints)

Console.WriteLine(i);

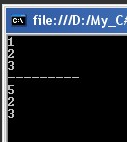
*// Изменить элемент, в источнике данных*

intArray[0] = 5;

Console.WriteLine("---------");

**foreach** (**int** i **in** ints)

Console.WriteLine(i);



\*\*\*

Extension to List

List<int> ints = intArray.Select(i => i).ToList();

\*\*\*

Funct

Аргумент-предикат указан как Func<T, bool>. Отсюда видно, что метод-предикат или лямбда-выражение должны принимать один аргумент — параметр T и возвращать bool. Вы знаете это потому, что известен тип возврата, указанный в конце списка параметров шаблона. Давайте посмотрим использование делегата Func на примере:

**int**[] ints = **new** **int**[] { 1, 2, 3, 4, 5, 6 };

*// Объявление делегата*

Func<**int**, **bool**> GreaterThanTwo = i => i > 2;

*// Выполнить запрос... He забывайте об отложенных запросах!*

IEnumerable<**int**> intsGreaterThanTwo = ints.Where(GreaterThanTwo);

**foreach** (**int** i **in** intsGreaterThanTwo)

Console.WriteLine(i);

\*\*\*

Thread-Safe Call and Task

public delegate void Helper();

public Task DownloadVHPercentage(string comText, string comValue)

{

Helper help = () =>

{

count++;

if (count == 36)

{

progressBar1.Value++;

count = 0;

}

};

Helper helpFull = () =>

{

progressBar1.Value = 100;

};

return Task.Factory.StartNew(() =>

{

}

}

);

}

private async void DownloadData\_Click(object sender, EventArgs e)

{

progressBar1.Value = 0;

button1.Enabled = false;

if (!IfFileExist(comboBox1.Text))

{

await DownloadVHData(comboBox1.Text, comboBox1.SelectedValue.ToString());

await DownloadVHPercentage(comboBox1.Text, comboBox1.SelectedValue.ToString());

}

else MessageBox.Show("Даний файл вже завантажено!");

button1.Enabled = true;

}

\*\*\*