

LINQ запросы

LINQ запросы

Отложенное выполнение — ключевая концепция в LINQ, подразумевающая, что выполнение запроса произойдет не сразу после определения запроса, а в ходе выполнения некоторых функций.

```
List<string> sourceArray = new List<string>();
    sourceArray.AddRange(new string[] { "cat", "dog", "fish"
});
    var query = from animal in sourceArray
                select animal;
    sourceArray.Add("bear");
    var result = query.ToArray();
    foreach (var s in result)
        Console.WriteLine(s);
```

Функции немедленного выполнения

- **Преобразование в список/массив или словарь**

`ToList()`, `ToArray()`, `ToDictionary()`

- **Цикл `for/foreach`**

- **Функции агрегации**

`Count()`, `First()`, `Average()`, `Min()`, `Max()`

- **Квантификаторы**

`Any()`, `All()`, `Contains()`

Операции отложенного выполнения

Ограничение	Where
Проекция/выборка	Select, Take, TakeWhile
Упорядочивание	OrderBy, ThenBy, Reverse
Группирование	GroupBy
Операции на множествах	Distinct, Union, Intersect
Преобразование типов	OfType
Генерация последовательностей	Range, Repeat, Empty
Соединение	Join, GroupJoin

Лямбда- выражение

Лямбда-выражение это блок кода, заменяющий анонимные методы.

$n \Rightarrow 5 * n$



*«n» переходит в «5 * n»*

Лямбда-

выражения

Простой

```
x => x * x  
() => { return 3.5; }
```

Продвинутый

```
(x, y) =>  
{  
    x = x * x;  
    return x * y;  
}
```

Пример применения в LINQ

```
// Исходный массив  
double[] source = new double[] { 4.5, 4.3, 1.2, 13.2, 394.3, 0.707 };  
  
// еще одна форма запроса в LINQ  
var query = source.Where((x) => { return x < 5.0; });  
  
// получаем все числа, меньшие 5.0  
foreach ( double elem in query )  
{  
    Console.WriteLine("elem = " + elem);  
}
```

АНОНИМНЫЙ ТИП

Анонимный тип — тип объекта, не требующий предварительного декларирования

```
var x = new { Num = 5.6, Message = "Hello!" };
```

АНОНИМНЫЙ ТИП

```
struct Complex {  
    public double Re { get; set; }  
    public double Im { get; set; }  
}  
  
void Main()  
{  
    Complex[] nums = new Complex[2];  
    nums[0] = new Complex() { Re = 1.0, Im = 0.4 };  
    nums[1] = new Complex() { Re = 2.3, Im = -1.4 };  
  
    var query = from z in nums  
                select new { Module = Math.Sqrt( z.Re * z.Re + z.Im*z.Im ), Arg = Math.Atan(z.Im/z.Re) };  
  
    query.Dump();  
}
```

Возврат из функции

```
public object Func()  
{  
    return new { X = "Hi!", Y = 32.222 };  
}
```


АНОНИМНЫЙ ТИП

Ограничение — фильтрация входящих данных согласно заданным условиям.

Проекция — возврат выходной последовательности элементов, сгенерированной за счет выбора элементов из исходных данных, либо за счет создания новых элементов, частично содержащих данные входной последовательности.

	Декларативный синтаксис	Императивный синтаксис
Ограничение (Where)	<pre>var query = from s in Students where s.Name == «Helen» select s;</pre>	<pre>var query = Students .Where(s => s.Name == «Helen»);</pre>
Проекция (Select)	<pre>var query = from s in Students where s.Name == «Helen» select s;</pre>	<pre>var query = Students .Where(s => s.Name == «Helen») .Select(s);</pre>

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Задача

Составить программу для вывода списка студентов старше от 18 до 24 лет с заданной фамилией или номером группы

```
var query3 = students.Where( s => s.Name.Contains("va")),
                    .Select( x => x.Name );
foreach ( string name in query3 )
{
    Console.WriteLine( String.Format("Student name: {0} ", name) );
}
```

```
List<Student> students = new List<Student>();
students.Add(new Student("Piter", "Collins", 19));
students.Add(new Student("Jennifer", "Lawrence", 24));
students.Add(new Student("Ivan", "Chertkov", 22));
students.Add(new Student("Sidney", "Nikolson", 27));
students.Add(new Student("Boris", "Nikolson", 22));
```

```
var query = students.Where(x=>x.Age >= 22);
```

```
foreach ( Student stud in query )
{
    Console.WriteLine( String.Format(" Student: {0}, {1} Age: {2} ", stud.Name, stud.Surname, stud.Age) );
}
```

Операции разбиения

Take — возвращает первые N элементов входного множества

TakeWhile — возвращает первые элементы входного множества, удовлетворяющие заданному условию

Skip — пропускает первые N элементов входного множества

SkipWhile — пропускает первые элементы входного множества, удовлетворяющие заданному условию

Операции разбиения

```
string[] streets = new string[] { "Новый Арбат", "Басманная", "Старый Арбат", "Бронная", "Остоженка"  
var query = streets.Take(3);  
  
// var query = streets.TakeWhile( x => x.Contains("н") )  
  
// var query = streets.Skip(5);  
  
//var query = streets.SkipWhile( x => x.Length > 7);
```

	Декларативный синтаксис	Императивный синтаксис
Конкатенация (Concat)	-	<code>var query = list1.Concat(list2);</code>
Упорядочивание (OrderBy, OrderByDescending)	<code>var query = from c in cities orderby c select c;</code>	<code>var query = cities.OrderBy(c=>c).Select(c1=>c1);</code>

Задача

Составить программу для вывода списка студентов старше 22 года и с заданной фамилией или номером группы с сортировкой по фамилии возрасту

	Декларативный синтаксис	Императивный синтаксис
Упорядочивание (ThenBy)	<pre>var query = from person in persons orderby person.Name, person.Surname select person;</pre>	<pre>var query = persons.OrderBy(c=>c.Name) .ThenBy(c1=>c1.Surname) .Select(c2=>c2);</pre>
Инвертирование (Reverse)	-	<pre>var query = cities.Reverse();</pre>

Join

	Декларативный синтаксис	Императивный синтаксис
Join	<pre>var query = from u in Universities join s in Students on u.Id equals s.UniversityId select new {Student = s.Name, University = u.Name};</pre>	<pre>var query = Universities.Join(Students, u => u.Id, s => s.UniversityId, (un1, std) => new { Student = std.Name, University = un1.Name });</pre>
GroupJoin	-	<pre>var query = regions.GroupJoin(cities, r => r.Id, c => c.RegionId, (reg, cts) => new { Name = reg.Name, Population = cts.Sum(c => c.Population) });</pre>

Активация Windows
Чтобы активировать Windows
найдите "Параметры"

Примеры

```
new Студент {Фамилия="Стороженко",
             Оценки= new List<int> {3, 3, 2, 3}},
new Студент {Фамилия="Ломачинская",
             Оценки= new List<int> {3, 4, 4, 5}},
new Студент {Фамилия="Погребницкий",
             Оценки= new List<int> {2, 4, 3, 2}},
new Студент {Фамилия="Левочкин",
             Оценки= new List<int> {3, 3, 4, 3}}
};
// Для доступа к внутреннему списку оценок предложение From
// используем два раза:
var СписокДвоечников = from Студент in Студенты
                       from Оценка in Студент.Оценки
                       where Оценка <= 2
                       orderby Студент.Фамилия
                       select new { Студент.Фамилия, Оценка };
foreach (var Студик in СписокДвоечников)
    textBox1.Text += string.Format("Студент {0} " +
    "имеет оценку: {1}\r\n", Студик.Фамилия, Студик.Оценка);
```

Примеры

```
new продукт {наименование="чай",      цена=10.25F },
new Продукт {Наименование="Мясо",     Цена=150.00F },
new Продукт {Наименование="Гречка",   Цена=62.50F },
};

var Запрос1 = from П in Продукты
              group П by new
              {
                  Критерий = П.Цена > 90,
              }
              into g
              select g;

var Запрос2 = from p in Продукты
              group p by p.Цена > 90 into g
              select new
              {
                  g.Key,
                  СредЦенаПоГруппе = g.Average(p => p.Цена)
              };

Single СредЦенаПоГруппе1 = Запрос2.ElementAt(0).СредЦенаПоГруппе;
Single СредЦенаПоГруппе2 = Запрос2.ElementAt(1).СредЦенаПоГруппе;
```

Актив
Чтобы
раздел

Словарь с.229