



Введение в LINQ

Natalie Vegerina
Software engineer
Infostroy Ltd,
Kharkov, Ukraine



ISO 9001
BUREAU VERITAS
Certification



Составляющие LINQ

- Автоматические свойства
- Инициализаторы объектов и коллекций
- Анонимные типы
- Методы расширения
- Лямбда-выражения

Автоматические свойства

```
private string _firstName;  
  
public string firstName  
{  
    get { return _firstName; }  
    set { _firstName = value; }  
}
```

Автоматические свойства

```
private string _firstName;  
  
public string firstName  
{  
    get { return _firstName; }  
    set { _firstName = value; }  
}
```



```
public string FirstName { get; set; }
```



Неявно типизированные локальные переменные

```
var i = 5;  
var s = "Hello";  
var d = 1.0;  
var numbers = new int[] { 1, 2, 3 };
```

Правила неявно типизированных локальных переменных

- `var x;` // Ошибка
- `var y = {1, 2, 3};` // Ошибка
- `var z = null;` // Ошибка
- `var` не может использоваться в полях в области действия класса.
- `int i = (i = 20);` // ОК
- `var i = (i = 20);` // Ошибка
- `var a = 5, b = 10;` // Ошибка
- Если тип с именем `var` находится в области действия, то ключевое слово `var` будет преобразовано в это имя типа и не будет обрабатываться как часть неявно типизированного объявления локальной переменной.

Инициализаторы объектов

```
public class Car
{
    public string VIN { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }
    public int Year { get; set; }
    public string Color { get; set; }
}
```

Инициализаторы объектов

```
Car c = new Car();  
c.VIN = "ABC123";  
c.Make = "Ford";  
c.Model = "F-250";  
c.Year = 2000;
```


Инициализаторы объектов

```
Car c = new Car();  
c.VIN = "ABC123";  
c.Make = "Ford";  
c.Model = "F-250";  
c.Year = 2000;
```



```
Car c = new Car()  
{  
    VIN = "ABC123",  
    Make = "Ford",  
    Model = "F-250",  
    Year = 2000  
};
```

Инициализаторы коллекций

```
private List<Car> GetCars()
{
    return new List<Car>
    {
        new Car { VIN = "ABC123", Make = "Ford",
            Model = "F-250", Year = 2000 },
        new Car { VIN = "DEF123", Make = "BMW",
            Model = "Z-3", Year = 2005 },
        new Car { VIN = "ABC456", Make = "Audi",
            Model = "TT", Year = 2008 },
        new Car { VIN = "HIJ123", Make = "VW",
            Model = "Bug", Year = 1956 },
        new Car { VIN = "DEF456", Make = "Ford",
            Model = "F-150", Year = 1998 }
    };
}
```

Анонимные типы

```
var car = new { Make = "VW", Model = "Bug" };
```

Методы расширения

```
public static class MyExtensions
{
    public static bool IsValidEmailAddress(this string s)
    {
        Regex regex = new Regex(@"^[\\w-\\.]+@[\\w-]+\\.+[\\w-]{2,4}$");
        return regex.IsMatch(s);
    }
}
```

```
string s = "my_mail@domen.com";
bool result = s.IsValidEmailAddress();
```

Правила методов расширений

- Метод расширения и класс, в котором он объявлен, должны быть статическими с модификатором `public`
- Первый параметр определяет, с каким типом оперирует метод, и перед параметром идет модификатор `this`
- Методы расширения не могут получить доступ к закрытым переменным типа, для расширения которого они используются
- Методы расширения находятся в области действия, только если пространство имен было явно импортировано в исходный код с помощью директивы `using`
- Методы расширения не могут переопределить методы класса

Анонимные методы

```
List<Car> cars = GetCars();  
List<Car> filtered = new List<Car>();  
foreach (Car car in cars)  
{  
    if (car.Year > 2000 && car.Model.StartsWith("F"))  
    {  
        filtered.Add(car);  
    }  
}
```

АНОНИМНЫЕ МЕТОДЫ

```
List<Car> cars = GetCars();  
List<Car> filtered = new List<Car>();  
foreach (Car car in cars)  
{  
    if (FilterYear(car) && FilterModel(car))  
    {  
        filtered.Add(car);  
    }  
}  
  
public bool FilterYear(Car car) { return car.Year > 2000; }  
public bool FilterModel(Car car) { return car.Model.StartsWith("F"); }
```

АНОНИМНЫЕ МЕТОДЫ

```
public delegate bool FilterPredicate(Car car);
```

```
public bool CheckPredicates(Car car, IList<FilterPredicate> predicates)
{
    foreach (FilterPredicate p in predicates)
    {
        if (!p(car)) { return false; }
    }

    return true;
}
```


АНОНИМНЫЕ МЕТОДЫ

```
List<Car> cars = GetCars();
```

```
List<FilterPredicate> predicates = new List<FilterPredicate>();  
predicates.Add(delegate(Car car) { return car.Year > 2000; });  
predicates.Add(delegate(Car car) { return car.Model.StartsWith("F"); });
```

```
List<Car> filtered = new List<Car>();  
foreach (Car car in cars)  
{  
    if (CheckPredicates(car, predicates)) { filtered.Add(car); }  
}
```

АНОНИМНЫЕ МЕТОДЫ

```
List<Car> cars = GetCars();
```

```
List<FilterPredicate> predicates = new List<FilterPredicate>();  
predicates.Add ((Car car) => car.Year > 2000);  
predicates.Add((Car car) => car.Model.StartsWith("F"));
```

```
List<Car> filtered = new List<Car>();  
foreach (Car car in cars)  
{  
    if (CheckPredicates(car, predicates)) { filtered.Add(car); }  
}
```

АНОНИМНЫЕ МЕТОДЫ

```
List<Car> cars = GetCars();
```

```
List<FilterPredicate> predicates = new List<FilterPredicate>();  
predicates.Add (car => car.Year > 2000);  
predicates.Add(car => car.Model.StartsWith("F"));
```

```
List<Car> filtered = new List<Car>();  
foreach (Car car in cars)  
{  
    if (CheckPredicates(car, predicates)) { filtered.Add(car); }  
}
```

Лямбда-выражения

- `(string s) => { return s.Length; }`
`(s) => { return s.Length; }`
- `() => { /* statement(s) */ }`
- `(s) => { return s.Length; }`
`s => { return s.Length; }`
- `(x,y) => { return x < y; }`

Архитектура

C#

.NET Language Integrated Query (LINQ)
LINQ data source
providers

ADO.NET support for LINQ

LINQ
to
Objects

LINQ
to
Datasets

LINQ
to SQL

LINQ
to Entities


LINQ
to XML

LINQ-примеры (Query-синтаксис)

```
var query = from book in list  
            where book.Category == ".NET"  
            orderby book.NumberOfCopies  
            select new { book.Title, book.Author};
```

LINQ-примеры (Query-синтаксис)

Источник
данных

A yellow rounded rectangular callout box containing the text "Источник данных". A black arrow points from the bottom-left corner of the box towards the word "from" in the LINQ query below.

```
var query = from book in list  
            where book.Category == ".NET"  
            orderby book.NumberOfCopies  
            select new { book.Title, book.Author};
```

LINQ-примеры (Query-синтаксис)

Ссылка на сущность
в
источнике данных

Источник
данных

```
var query = from book in list  
where book.Category == ".NET"  
orderby book.NumberOfCopies  
select new { book.Title, book.Author};
```


LINQ-примеры (Query-синтаксис)

Ссылка на сущность
в
источнике данных

Источник
данных

```
var query = from book in list
            where book.Category == ".NET"
            orderby book.NumberOfCopies
            select new { book.Title, book.Author};
```

Фильтрация и
сортировка

LINQ-примеры (Query-синтаксис)

Ссылка на сущность
в
источнике данных

Источник
данных

```
var query = from book in list
            where book.Category == ".NET"
            orderby book.NumberOfCopies
            select new { book.Title, book.Author};
```

Фильтрация и
сортировка


Преобразова
ние
данных

Синтаксис

```
var query = from book in list  
where book.Category == ".NET"  
orderby book.NumberOfCopies  
select new { book.Title, book.Author};
```

Синтаксис

Обязательное
начало
запроса

A black arrow points downwards from the yellow box to the start of the code snippet.

```
var query = from book in list  
where book.Category == ".NET"  
orderby book.NumberOfCopies  
select new { book.Title, book.Author};
```

Синтаксис

Обязательное
начало
запроса

```
var query = from book in list  
where book.Category == ".NET"  
orderby book.NumberOfCopies  
select new { book.Title, book.Author};
```

Обязательный конец
запроса

Синтаксис

Обязательное
начало
запроса

В середине
запроса
могут
содержаться
одно или более
условий:

- where
- orderby
- join
- let
- from
- into

```
query = from book in list
where book.Category == ".NET"
orderby book.NumberOfCopies
select new { book.Title, book.Author};
```

Обязательный конец
запроса

Ключевые слова

C# LINQ

from
select
where
orderby
join
group
Distinct()
into
let
Count(), Sum(),...
Skip(), SkipWhile()
Take(), TakeWhile()

ANSI SQL

FROM
SELECT
WHERE
ORDER BY
JOIN
GROUP BY
DISTINCT
INTO
AS
COUNT, SUM,...
N/A
N/A

Отложенное выполнение

```
string[] colors = { "Red", "Brown", "Orange", "Yellow", "Black", "Green", "White", "Violet",  
"Blue" };
```

```
IEnumerable<string> results = from c in colors  
    where c.StartsWith("B")  
    orderby c  
    select c;
```

```
foreach (var color in results)  
{  
    txtLog.AppendText(color + ",");  
}
```

Output: Brown, Black, Blue

Отложенное выполнение

```
string[] colors = { "Red", "Brown", "Orange", "Yellow", "Black", "Green", "White", "Violet",  
"Blue" };
```

```
IEnumerable<string> results = from c in colors  
    where c.StartsWith("B")  
    orderby c  
    select c;
```

```
colors[4] = "Slate";
```

```
foreach (var color in results)  
{  
    txtLog.AppendText(color + ",");  
}
```

Output: Brown, Blue



Query Extension Methods

Enumerable static methods

Empty

```
var numbers = Enumerable.Empty<long>();
```

Range

```
var numbers = Enumerable.Range(1, 5);
```

Output: 1, 2, 3, 4, 5

Repeat

```
var numbers = Enumerable.Repeat(10, 5);
```

Output: 10, 10, 10, 10, 10



Query Extension Methods

All, Any

All

```
bool result = GetCars().All(c => c.Year > 1960);
```

Any

```
bool result = GetCars().Any(c => c.Year <= 1960);
```

Query Extension Methods

AsEnumerable

```
public class MyStringList : List<string>
{
    public IEnumerable<string> Where(Predicate<string> filter)
    {
        return this.Select(s => filter(s) ? s.ToUpper() : s);
    }
}

var strings = new MyStringList {"orange", "apple", "grape", "pear"};
foreach (var item in strings.Where(s => s.Length == 5))
{
    txtLog.Write(item);
}
```

Output: orange, APPLE, GRAPE, pear

Query Extension Methods

AsEnumerable

```
public class MyStringList : List<string>
{
    public IEnumerable<string> Where(Predicate<string> filter)
    {
        return this.Select(s => filter(s) ? s.ToUpper() : s);
    }
}

var strings = new MyStringList {"orange", "apple", "grape", "pear"};
foreach (var item in strings.AsEnumerable().Where(s => s.Length == 5))
{
    txtLog.Write(item);
}
```

Output: apple, grape



Query Extension Methods

AsQueryable

```
IEnumerable<string> strings = new MyStringList {"orange", "apple", "grape", "pear"};  
var querable = strings.AsQueryable();  
txtLog.WriteLine("Element Type: {0}", querable.ElementType);  
txtLog.WriteLine("Expression: {0}", querable.Expression);  
txtLog.WriteLine("Provider: {0}", querable.Provider);
```

Output:

Element Type: System.String

Expression: MyStringList

Provider: MyStringList



Query Extension Methods

ToArray, ToList

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
var evenScores = scores.Where(s => s % 2 == 0).ToList(); // ToArray()  
scores[2] = 2;  
foreach (var item in evenScores)  
{  
    txtLog.WriteLine(item);  
}
```

Output:

```
88  
56  
78  
90
```



Query Extension Methods ToDictionary, ToLookUp

```
var cars = GetCars();  
var carsByVin = cars.ToDictionary(c => c.VIN);  
Car myCar = carsByVin["HIJ123"];  
txtLog.WriteLine("Car VIN:{0}, Make:{1}, Model:{2} Year:{3}",  
    myCar.VIN, myCar.Make, myCar.Model, myCar.Year);
```

Output:

Car VIN:HIJ123, Make:VW, Model:Bug Year:1956



Query Extension Methods

Cast, OfType

```
int[] scores = {88, 56, 23, 99, 65, 93, 78, 23, 99, 90};  
IEnumerable<Object> objects = scores.Cast<object>();
```

```
object[] items = new object[] {55, "Hello", 22, "Goodbye"};  
foreach (var intItem in items.Of<int>())  
{  
    txtLog.WriteLine(intItem);  
}
```

Output:

```
55  
22
```



Query Extension Methods

Max, Min, Average, Sum, Count

```
int[] scores = {88, 56, 23, 99, 65, 93, 78, 23, 99, 90};  
txtLog.WriteLine("Max: " + scores.Max());  
txtLog.WriteLine("Min: " + scores.Min());  
txtLog.WriteLine("Average: " + scores.Average());  
txtLog.WriteLine("Sum: " + scores.Sum());  
txtLog.WriteLine("Count: " + scores.Count());
```

Output:

Max: 99

Min: 23

Average: 71,4

Sum: 714

Count: 10



Query Extension Methods

ElementAt, First, Last

```
int[] scores = {88, 56, 23, 99, 65, 93, 78, 23, 99, 90};  
txtLog.WriteLine(scores.ElementAt(4));  
txtLog.WriteLine(scores.First());  
txtLog.WriteLine(scores.Last());
```

Output:

```
65  
88  
90
```



Query Extension Methods

ElementAtOrDefault, FirstOrDefault, LastOrDefault

```
int[] scores = {};  
txtLog.WriteLine(scores.ElementAtOrDefault(4));  
txtLog.WriteLine(scores.FirstOrDefault());  
txtLog.WriteLine(scores.LastOrDefault());
```

Output:

```
0  
0  
0
```



Query Extension Methods

Single, SingleOrDefault

```
int[] scores = {88, 56, 23, 99, 65, 93, 78, 23, 99, 90};  
int score1 = scores.Where(x => x > 50 && x < 60).Single();  
int score2 = scores.Where(x => x > 100).SingleOrDefault();  
txtLog.WriteLine(score1);  
txtLog.WriteLine(score2);
```

Output:

```
56  
0
```

Query Extension Methods

DefaultIfEmpty

```
List<Car> cars = new List<Car>();  
IEnumerable<Car> oneNullCar = cars.DefaultIfEmpty();  
foreach (var car in oneNullCar)  
{  
    txtLog.WriteLine(car == null ? "Null Car" : "Not Null Car");  
}
```

Output: Null Car



Query Extension Methods

Contains

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
txtLog.WriteLine(scores.Contains(56));  
txtLog.WriteLine(scores.Contains(24));
```

Output:

True

False



Query Extension Methods

Distinct

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
foreach (var score in scores.Distinct())  
{  
    txtLog.Write(score);  
}
```

Output: 88, 56, 23, 99, 65, 93, 78, 90



Query Extension Methods

Concat

```
int[] lastYearScores = { 88, 56, 23, 99, 65 };  
int[] thisYearScores = { 93, 78, 23, 99, 90 };  
  
foreach (var item in lastYearScores.Concat(thisYearScores))  
{  
    txtLog.Write(item);  
}
```

Output: 88, 56, 23, 99, 65, 93, 78, 23, 99, 90



Query Extension Methods

Except

```
int[] lastYearScores = { 88, 56, 23, 99, 65 };
```

```
int[] thisYearScores = { 93, 78, 23, 99, 90 };
```

```
foreach (var item in lastYearScores.Except(thisYearScores))  
{  
    txtLog.WriteLine(item);  
}
```

Output:

88

56

65



Query Extension Methods

Intersect

```
int[] lastYearScores = { 88, 56, 23, 99, 65 };
```

```
int[] thisYearScores = { 93, 78, 23, 99, 90 };
```

```
foreach (var item in lastYearScores.Intersect(thisYearScores))  
{  
    txtLog.WriteLine(item);  
}
```

Output:

23

99



Query Extension Methods

Union

```
int[] lastYearScores = { 88, 56, 23, 99, 65, 56 };  
int[] thisYearScores = { 93, 78, 23, 99, 90, 99 };  
var allScores = lastYearScores.Union(thisYearScores);  
foreach (var item in allScores.OrderBy(s => s))  
{  
    txtLog.WriteLine(item);  
}
```

Output: 23, 56, 65, 78, 88, 90, 93, 99



Query Extension Methods

Zip

```
var numbers = Enumerable.Range(1, 1000);
var cars = GetCars();
var zip = numbers.Zip(cars, (i, c) => new
    {
        Number = i,
        CarMake = c.Make
    });
foreach (var it in zip)
{
    txtLog.WriteLine("Number:{0} Make:{1}", it.Number, it.CarMake);
}
```

Output:

```
Number:1 CarMake:Ford
Number:2 CarMake:BMW
Number:3 CarMake:Audi
Number:4 CarMake:VW
Number:5 CarMake:Ford
```



Query Extension Methods

Reverse

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };  
foreach (var score in scores.Reverse())  
{  
    txtLog.Write(score);  
}
```

Output: 90, 99, 23, 78, 93, 65, 99, 23, 56, 88



Query Extension Methods

SequenceEqual

```
List<int> lastYearScores = new List<int> {93, 78, 23, 99, 91};  
List<int> thisYearScores = new List<int> {93, 78, 23, 99, 90};  
txtLog.WriteLine(lastYearScores.SequenceEqual(thisYearScores));  
lastYearScores[4] = 90;  
txtLog.WriteLine(lastYearScores.SequenceEqual(thisYearScores));
```

Output:

False

True



Query Extension Methods

Skip, SkipWhile

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };
```

```
foreach (var s in scores.OrderBy(i => i).Skip(9))  
{ txtLog.WriteLine(s); }
```

Output: 99

```
foreach (var s in scores.OrderBy(i => i).SkipWhile(s => s < 80))  
{ txtLog.Write(s); }
```

Output: 88, 90, 93, 99, 99

```
foreach (var s in scores.SkipWhile(s => s < 80))  
{ txtLog.Write(s); }
```

Output: 88, 56, 23, 99, 65, 93, 78, 23, 99, 90



Query Extension Methods

Take, TakeWhile

```
int[] scores = { 88, 56, 23, 99, 65, 93, 78, 23, 99, 90 };
```

```
foreach (var item in scores.OrderBy(i => i).Skip(3).Take(2))  
{  
    txtLog.Write(item);  
}
```

Output: 65, 78

```
foreach (var item in scores.OrderBy(i => i).TakeWhile(s => s < 80))  
{  
    txtLog.WriteLine(item);  
}
```

Output: 23, 23, 56, 65, 78



Query Extension Methods

GroupBy

```
var cars = GetCars();  
var query = cars.GroupBy(c => c.Make);  
foreach (IGrouping<string, Car> group in query)  
{  
    txtLog.WriteLine("Key:{0}", group.Key);  
    foreach (Car c in group)  
    { txtLog.WriteLine("Car VIN:{0} Make:{1}", c.VIN, c.Make); }  
}
```

Output:

```
Key:Ford  
Car VIN:ABC123 Make:Ford  
Car VIN:DEF456 Make:Ford  
Key:BMW  
Car VIN:DEF123 Make:BMW  
Key:Audi  
Car VIN:ABC456 Make:Audi  
Key:VW  
Car VIN:HIJ123 Make:VW
```



Query Extension Methods

OrderBy, OrderByDescending, ThenBy, ThenByDescending

```
var cars = GetCars().OrderBy(c => c.Make)
                    .ThenByDescending(c => c.Model)
                    .ThenBy(c => c.Year);
foreach (var item in cars)
{
    txtLog.WriteLine("Car VIN:{0} Make:{1} Model:{2} Year:{3}",
        item.VIN, item.Make, item.Model, item.Year);
}
```

Output:

```
Car VIN:ABC456 Make:Audi Model:TT Year:2008
Car VIN:DEF123 Make:BMW Model:Z-3 Year:2005
Car VIN:ABC123 Make:Ford Model:F-250 Year:2000
Car VIN:DEF456 Make:Ford Model:F-150 Year:1998
Car VIN:HIJ123 Make:VW Model:Bug Year:1956
```



Query Extension Methods Where

```
var cars = GetCars();  
foreach (var myCar in cars.Where(c => c.Make == "Ford"))  
{  
    txtLog.WriteLine("Car VIN:{0}, Make:{1}, Model:{2} Year:{3}",  
                    myCar.VIN, myCar.Make, myCar.Model, myCar.Year);  
}
```

Output:

```
Car VIN:ABC123, Make:Ford, Model:F-250 Year:2000  
Car VIN:DEF456, Make:Ford, Model:F-150 Year:1998
```

Query Extension Methods

Select

```
var vehicles = new List<Tuple<string, string, int>>
{
    Tuple.Create("123", "VW", 1999),
    Tuple.Create("234", "Ford", 2009),
    Tuple.Create("567", "Audi", 2005),
    Tuple.Create("678", "Ford", 2003),
    Tuple.Create("789", "Mazda", 2003),
    Tuple.Create("999", "Ford", 1965)
};
```



Query Extension Methods

Select

```
var fordCars = vehicles
    .Where(v => v.Item2 == "Ford")
    .Select(v => new Car
    {
        VIN = v.Item1,
        Make = v.Item2,
        Year = v.Item3
    });
foreach (var item in fordCars)
{
    txtLog.WriteLine("Car VIN:{0} Make:{1} Year:{2}",
        item.VIN, item.Make, item.Year);
}
```



Query Extension Methods

Select

Output:

Car VIN:234 Make:Ford Year:2009

Car VIN:678 Make:Ford Year:2003

Car VIN:999 Make:Ford Year:1965

Query Extension Methods

Select

```
var fordCars = from v in vehicles
               let makeYear = v.Item2 + " " + v.Item3
               where makeYear.StartsWith("Ford")
               select new
               {
                   VIN = v.Item1,
                   MakeYear = makeYear
               };
foreach (var item in fordCars)
{
    txtLog.WriteLine("Car VIN:{0} MakeYear:{1}",
                    item.VIN, item.MakeYear);
}
```




Query Extension Methods

Select

Output:

Car VIN:234 MakeYear:Ford 2009

Car VIN:678 MakeYear:Ford 2003

Car VIN:999 MakeYear:Ford 1965

Query Extension Methods

SelectMany

```
var repairs = new List<Tuple<string, List<string>>>
{
    Tuple.Create("ABC123",
        new List<string> {"Rotate Tires", "Change oil"}),
    Tuple.Create("DEF123",
        new List<string> {"Fix Flat", "Wash Vehicle"}),
    Tuple.Create("ABC456",
        new List<string> {"Alignment", "Vacuum", "Wax"}),
    Tuple.Create("HIJ123",
        new List<string> {"Spark plugs", "Air filter"}),
    Tuple.Create("DEF456",
        new List<string> {"Wiper blades", "PVC valve"}),
};
```



Query Extension Methods

SelectMany

```
var query = repairs.SelectMany(t =>
    t.Item2.Select(r => new { VIN = t.Item1, Repair = r }));

foreach (var item in query)
{
    txtLog.WriteLine("VIN:{0} Repair:{1}", item.VIN, item.Repair);
}
```



Query Extension Methods

SelectMany

Output:

VIN:ABC123 Repair:Rotate Tires
VIN:ABC123 Repair:Change oil
VIN:DEF123 Repair:Fix Flat
VIN:DEF123 Repair:Wash Vehicle
VIN:ABC456 Repair:Alignment
VIN:ABC456 Repair:Vacuum
VIN:ABC456 Repair:Wax
VIN:HIJ123 Repair:Spark plugs
VIN:HIJ123 Repair:Air filter
VIN:DEF456 Repair:Wiper blades
VIN:DEF456 Repair:PVC valve



Query Extension Methods

Join

```
var makes = new string[] { "Audi", "BMW", "Ford", "Mazda", "VW" };
var cars = GetCars();
var query = makes.Join(cars,
    make => make, car => car.Make,
    (make, innerCar) => new { Make = make, Car = innerCar });
foreach (var item in query)
{
    txtLog.WriteLine("Make: {0}, Car:{1} {2} {3}",
        item.Make, item.Car.VIN, item.Car.Make, item.Car.Model);
}
```

Output:

```
Make: Audi, Car:ABC456 Audi TT
Make: BMW, Car:DEF123 BMW Z-3
Make: Ford, Car:ABC123 Ford F-250
Make: Ford, Car:DEF456 Ford F-150
Make: VW, Car:HIJ123 VW Bug
```



Query Extension Methods

GroupJoin

```
var makes = new string[] { "Audi", "BMW", "Ford", "Mazda", "VW" };
var cars = GetCars();
var query = makes.GroupJoin(cars,
    make => make, car => car.Make,
    (make, innerCars) => new { Make = make, Cars = innerCars });
foreach (var item in query)
{
    txtLog.WriteLine("Make: {0}", item.Make);
    foreach (var car in item.Cars)
    { txtLog.WriteLine("Car VIN:{0}, Model:{1}", car.VIN, car.Model); }
}
```

Output:

```
Make: Audi
Car VIN:ABC456, Model:TT
Make: BMW
Car VIN:DEF123, Model:Z-3
Make: Ford
Car VIN:ABC123, Model:F-250
Car VIN:DEF456, Model:F-150
```

LINQ Joins

```
public class Repair {
    public string VIN { get; set; }
    public string Desc { get; set; }
    public decimal Cost { get; set; }
}

private List<Repair> GetRepairs()
{
    return new List<Repair> {
        new Repair {VIN = "ABC123", Desc = "Change Oil", Cost = 29.99m},
        new Repair {VIN = "DEF123", Desc = "Rotate Tires", Cost =19.99m},
        new Repair {VIN = "HIJ123", Desc = "Replace Brakes", Cost = 200},
        new Repair {VIN = "DEF456", Desc = "Alignment", Cost = 30},
        new Repair {VIN = "ABC123", Desc = "Fix Flat Tire", Cost = 15},
        new Repair {VIN = "DEF123", Desc = "Fix Windshield", Cost =420},
        new Repair {VIN = "ABC123", Desc = "Replace Wipers", Cost = 20},
        new Repair {VIN = "HIJ123", Desc = "Replace Tires", Cost = 1000},
        new Repair {VIN = "DEF456", Desc = "Change Oil", Cost = 30} };
}
```

LINQ Joins

Inner join

```

var cars = GetCars();
var repairs = GetRepairs();
var carsWithRepairs = from c in cars
                      join r in repairs
                      on c.VIN equals r.VIN
                      orderby c.VIN, r.Cost
                      select new
                      {
                          c.VIN,
                          c.Make,
                          r.Desc,
                          r.Cost
                      };
foreach (var item in carsWithRepairs)
{
    txtLog.WriteLine("Car VIN:{0}, Make:{1}, Description:{2} Cost:{3:C}",
        item.VIN, item.Make, item.Desc, item.Cost);
}

```




LINQ Joins

Inner join

Output:

Car VIN:ABC123, Make:Ford, Description:Fix Flat Tire Cost:\$15.00
Car VIN:ABC123, Make:Ford, Description:Replace Wipers Cost:\$20.00
Car VIN:ABC123, Make:Ford, Description:Change Oil Cost:\$29.99
Car VIN:DEF123, Make:BMW, Description:Rotate Tires Cost:\$19.99
Car VIN:DEF123, Make:BMW, Description:Fix Windshield Cost:\$420.00
Car VIN:DEF456, Make:Ford, Description:Alignment Cost:\$30.00
Car VIN:DEF456, Make:Ford, Description:Change Oil Cost:\$30.00
Car VIN:HIJ123, Make:VW, Description:Replace Brakes Cost:\$200.00
Car VIN:HIJ123, Make:VW, Description:Replace Tires Cost:\$1,000.00

LINQ Joins

Outer join

```

var carsWithRepairs = from c in cars
                      join r in repairs
                      on c.VIN equals r.VIN into g
                      from r in g.DefaultIfEmpty()
                      orderby c.VIN, r == null ? 0 : r.Cost
                      select new
                      {
                          c.VIN,
                          c.Make,
                          Desc = r == null ? "***No Repairs***" : r.Desc,
                          Cost = r == null ? 0 : r.Cost
                      };

foreach (var item in carsWithRepairs)
{
    txtLog.WriteLine("Car VIN:{0}, Make:{1}, Description:{2} Cost:{3:C}",
        item.VIN, item.Make, item.Desc, item.Cost);
}

```



LINQ Joins

Outer join

Output:

Car VIN:ABC123, Make:Ford, Description:Fix Flat Tire Cost:\$15.00

Car VIN:ABC123, Make:Ford, Description:Replace Wipers Cost:\$20.00

Car VIN:ABC123, Make:Ford, Description:Change Oil Cost:\$29.99

Car VIN:ABC456, Make:Audi, Description:***No Repairs*** Cost:\$0.00

Car VIN:DEF123, Make:BMW, Description:Rotate Tires Cost:\$19.99

Car VIN:DEF123, Make:BMW, Description:Fix Windshield Cost:\$420.00

Car VIN:DEF456, Make:Ford, Description:Alignment Cost:\$30.00

Car VIN:DEF456, Make:Ford, Description:Change Oil Cost:\$30.00

Car VIN:HIJ123, Make:VW, Description:Replace Brakes Cost:\$200.00

Car VIN:HIJ123, Make:VW, Description:Replace Tires Cost:\$1,000.00

LINQ Joins

Cross join

```
var cars = GetCars();
var colors = new string[] { "Red", "Blue", "Green" };

var carsWithRepairs = from car in cars
                      from color in colors
                      orderby car.VIN, color
                      select new
                      {
                          car.VIN,
                          car.Make,
                          car.Model,
                          Color = color
                      };

foreach (var item in carsWithRepairs)
{
    txtLog.WriteLine("Car VIN:{0}, Make:{1}, Model:{2} Color:{3}",
                    item.VIN, item.Make, item.Model, item.Color);
}
```



LINQ Joins

Cross join

Output:

Car VIN:ABC123, Make:Ford, Model:F-250 Color:Blue
Car VIN:ABC123, Make:Ford, Model:F-250 Color:Green
Car VIN:ABC123, Make:Ford, Model:F-250 Color:Red
Car VIN:ABC456, Make:Audi, Model:TT Color:Blue
Car VIN:ABC456, Make:Audi, Model:TT Color:Green
Car VIN:ABC456, Make:Audi, Model:TT Color:Red
Car VIN:DEF123, Make:BMW, Model:Z-3 Color:Blue
Car VIN:DEF123, Make:BMW, Model:Z-3 Color:Green
Car VIN:DEF123, Make:BMW, Model:Z-3 Color:Red
Car VIN:DEF456, Make:Ford, Model:F-150 Color:Blue
Car VIN:DEF456, Make:Ford, Model:F-150 Color:Green
Car VIN:DEF456, Make:Ford, Model:F-150 Color:Red
Car VIN:HIJ123, Make:VW, Model:Bug Color:Blue
Car VIN:HIJ123, Make:VW, Model:Bug Color:Green
Car VIN:HIJ123, Make:VW, Model:Bug Color:Red

Литература

- Glenn Johnson - Exam 70-516: TS: Accessing Data with Microsoft .NET Framework 4
- Mike Liu - WCF 4.0 Multi-tier Services Development with LINQ to Entities
- MSDN