

Появление LINQ

[LINQ \(../base/level1/info_linq.php\)](#) --- [LINQ to Objects \(../level1/linq_index.php\)](#) --- Появление LINQ

По мере становления платформы .NET Framework и поддерживаемых ею языков C# и VB, стало ясно, что одной из наиболее проблемных областей для разработчиков остается доступ к данным из разных источников. В частности, доступ к базе данных и манипуляции XML часто в лучшем случае запутаны, а в худшем — проблематичны.

Проблемы, связанные с базами данных, многочисленны. **Первая сложность** в том, что нельзя программно взаимодействовать с базой данных на уровне естественного языка. Это приводит к синтаксическим ошибкам, которые не проявляются вплоть до момента запуска. Неправильные ссылки на поля базы данных тоже не обнаруживаются. Это может пагубно отразиться на программе, особенно если произойдет во время выполнения кода обработки ошибок. Нет ничего хуже, чем сбой механизма обработки ошибок из-за синтаксически неверного кода, который никогда не тестировался. Иногда это неизбежно из-за непредсказуемого поведения ошибки. Наличие кода базы данных, который не проверяется во время компиляции, определенно может привести к этой проблеме.

Вторая проблема связана с неудобством, которое вызвано различными типами данных используемыми определенным доменом данных, например, разница между типами базы данных или типами XML и типами данных в языке, на котором написана программа. В частности, серьезные сложности могут вызывать типы времени и даты.

Разбор итерация и манипулирование XML-разметкой могут быть достаточно утомительными. Часто фрагмент XML - это все, что нужно, но из-за требований интерфейса W3C DOM XML API объект XmlDocument должен быть обязательно создан, чтобы выполнять различные операции над фрагментом XML.

Вместо того чтобы просто добавить больше классов и методов для постепенного восполнения этих недостатков, в Microsoft решили пойти на один шаг дальше в абстрагировании основ запросов данных из этих конкретных доменов данных. В результате появился **LINQ** - технология Microsoft, предназначенная для поддержки запросов к данным всех типов на уровне языка. Эти типы включают массивы и коллекции в памяти, базы данных, документы XML и многое другое.

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)



.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

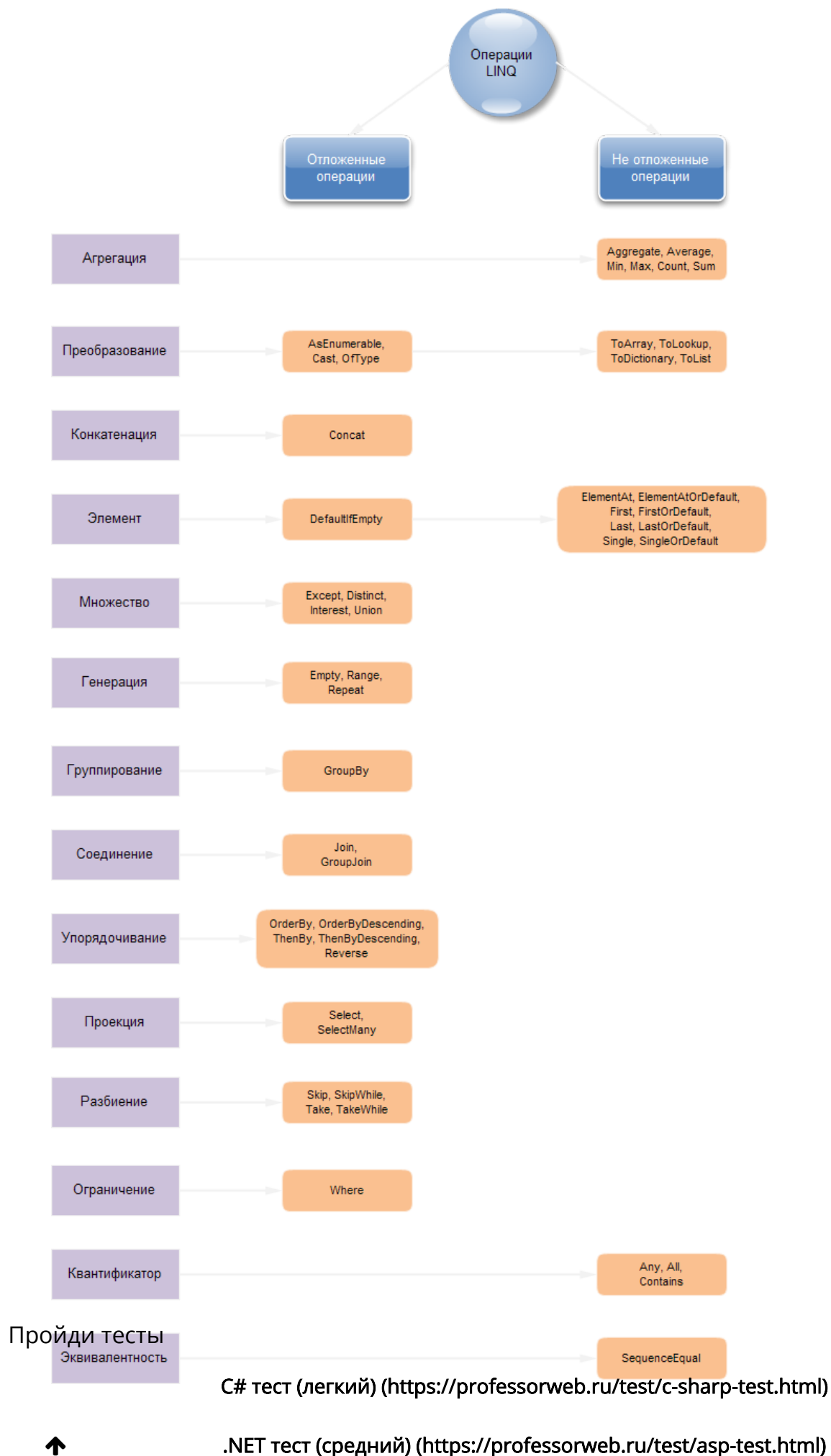
LINQ включает в себя около 50 стандартных операций запросов, разделяемых на 2 большие группы - **отложенные операции** (выполняются не во время инициализации, а только при их вызове) и **не отложенные операции** (выполняются сразу). На рисунке ниже наглядно показана "градация" операций LINQ:

Пройди тесты

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)



.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)



По большей части LINQ ориентирован на **запросы** — будь то запросы, возвращающие набор подходящих объектов, единственный объект или подмножество полей из объекта либо набора объектов. В LINQ этот возвращенный набор называется *последовательностью (sequence)*. Большинство последовательностей LINQ имеют тип `IEnumerable<T>` (`../csharp/charp_theory/level12/12_19.php`), где `T` — тип данных объектов, находящихся в последовательности. Например, если есть последовательность целых чисел, они должны храниться в переменной типа `IEnumerable<int>`. Вы увидите, что `IEnumerable<T>` буквально господствует в LINQ и очень многие методы LINQ возвращают `IEnumerable<T>`.

Может показаться, что LINQ — это нечто, связанное только с запросами, поскольку расшифровывается как **язык интегрированных запросов (Language Integrated Query)**. Однако не думайте о нем лишь в этом контексте. Предпочтительнее воспринимать LINQ как *механизм итерации данных (data iteration engine)*, но возможно в Microsoft не захотели обозначать эту технологию аббревиатурой DIE ("умереть").

Приходилось ли вам когда-нибудь вызывать метод, возвращающий данные в структуре, которую затем приходилось преобразовывать в еще одну структуру данных, прежде чем передать другому методу? Предположим, например, что вызывается метод `A`, и этот метод возвращает массив типа `string`, содержащий числовые значения в виде строк. Затем нужно вызвать метод `B`, но метод `B` требует массива целых чисел. Обычно приходится организовывать цикл для прохода по массиву строк и заполнения вновь сконструированного массива целых чисел. Давайте рассмотрим краткий пример мощи Microsoft LINQ.

Предположим, что имеется массив строк, которые приняты от метода `A`, как показано ниже:

C#

```
string[] numbers = { "40", "2012", "176", "5" };

// Преобразуем массив строк в массив типа int используя LINQ
int[] nums = numbers.Select(s => Int32.Parse(s)).ToArray();

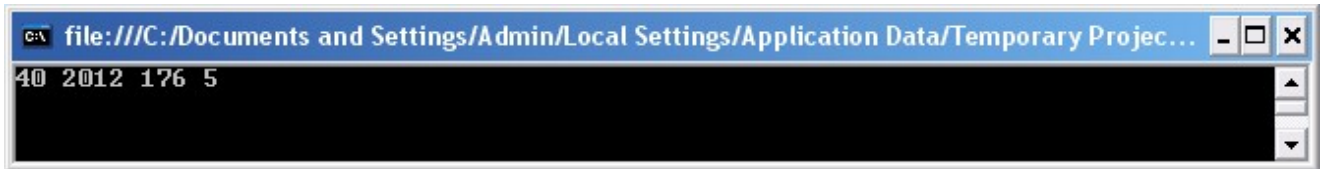
foreach (int n in nums)
    Console.Write(n + " ");
```

Пройдите тест. Что может быть проще?

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)



.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

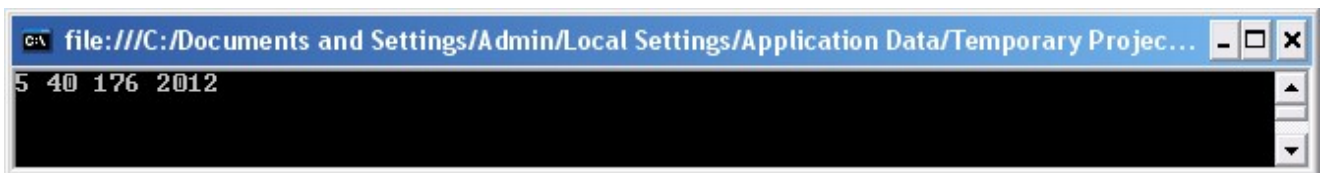


```
C:\ file:///C:/Documents and Settings/Admin/Local Settings/Application Data/Temporary Projec...  
40 2012 176 5
```

Возможно, вы подумали, что просто в строках отброшены ведущие пробелы. Но убедит ли вас, если отсортировать результат? Если бы это были по-прежнему строки, то 5 окажется в конце, а 176 — в начале. Ниже приведен код, который выполняет преобразование и сортирует вывод:

C#

```
string[] numbers = { "40", "2012", "176", "5" };  
  
// Преобразуем массив строк в массив типа int и сортируем по возрастанию используя LINQ  
int[] nums = numbers.Select(s => Int32.Parse(s)).OrderBy(s => s).ToArray();
```



```
C:\ file:///C:/Documents and Settings/Admin/Local Settings/Application Data/Temporary Projec...  
5 40 176 2012
```

Не правда ли гладко? Вы можете возразить, что все это прекрасно, но пример был очень прост. Давайте рассмотрим более сложный пример. Предположим, что есть некоторый код, содержащий класс `Employee`. В этом классе имеется метод, возвращающий всех сотрудников. Также предположим, что есть другой код, включающий класс `Contact`, с определенным в нем методом, публикующим контакты. Пусть необходимо опубликовать всех сотрудников в виде контактов.

Задача выглядит достаточно простой, но здесь таится ловушка. Общий метод `Employee`, который извлекает всех сотрудников, возвращает их в списке `ArrayList` хранящем объекты `Employee`, а метод `Contact`, публикующий контакты, требует массива объектов типа `Contact`. Ниже показан обычный для такого случая код:

C#

Пройди тесты

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)



.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

```
public class Employee
{
    public int id;
    public string firstName;
    public string lastName;

    public static ArrayList GetEmployees()
    {
        ArrayList al = new ArrayList();
        // В реальном коде коллекция заполнялась бы из базы данных
        al.Add(new Employee { id = 1, firstName = "Alexandr", lastName = "Erohin" });
        al.Add(new Employee { id = 2, firstName = "Alexey", lastName = "Volkov" });
        al.Add(new Employee { id = 3, firstName = "Dmitry", lastName = "Moiseenko" });
        return (al);
    }
}

public class Contact
{
    public int id;
    public string Name;

    public static void PublishContacts(Contact[] contacts)
    {
        foreach (Contact c in contacts)
            Console.WriteLine("Contact Id: {0} Contact: {1}", c.id, c.Name);
    }
}
```

Как видите, метод `GetEmployees` возвращает `ArrayList`. Метод `PublishContacts` требует передачи ему массива объектов `Contact`. Ранее это всегда требовало итерации по списку `ArrayList`, который возвращен методом `GetEmployee` и создания нового массива типа `Contact` для передачи методу `PublishContacts()`. Как можно видеть ниже, LINQ значительно облегчает решение этой задачи:

C#

Пройди тесты

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)



.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

```

class Program
{
    static void Main()
    {
        ArrayList al = Employee.GetEmployees();
        Contact[] contacts = al
            .Cast<Employee>()
            .Select(e => new Contact
            {
                id = e.id,
                Name = string.Format("{0} {1}", e.firstName, e.lastName)
            })
            .ToArray<Contact>();

        Contact.PublishContacts(contacts);
        Console.ReadLine();
    }
}

```



Чтобы преобразовать коллекцию ArrayList объектов Employee в массив объектов Contact, сначала выполняется приведение ArrayList объектов Employee к последовательности IEnumerable<Employee> с использованием стандартной операции запросов Cast. Это необходимо, потому что использован унаследованный класс коллекции ArrayList. С синтаксической точки зрения в коллекции ArrayList хранятся объекты класса System.Object, а не объекты типа класса Employee. Поэтому они должны быть приведены к объектам Employee. Если бы метод GetEmployees возвращал обобщенную коллекцию List, необходимости в этом не было бы. Однако на момент написания унаследованного кода этот тип коллекции не был доступным.

Затем на возвращенной последовательности объектов Employee вызывается операция Select, и в лямбда-выражении — коде, переданном внутрь вызова метода Select, создается и инициализируется экземпляр объекта Contact с использованием для этого средства инициализации объектов C#, чтобы присвоить значения входного элемента Employee вновь сконструированному выходному элементу Contact. Лямбда-выражение (lambda expression) (http://csharp/charp_theory/level10/10_6.php) — это средство C#, которое является сокращенным способом указания анонимных методов. И, наконец, последовательность вновь сконструированных объектов Contact преобразуется в массив объектов Contact с применением операции ToArray, потому что этого требует метод PublishContacts. Разве не изящно?

1	2	3	Вперед (1_2.php)
---	---	---	------------------



Лучший чат для C# программистов (<https://t.me/professorweb>)

Professor Web (/)

Наш любимый хостинг (/)

Пройди тесты

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)



.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)