LINQ

LINQ to

Пачка linq запросов к различным сущностям

- LINQ to Objects: операции с массивами и коллекциями
- LINQ to Entities: используется при обращении к базам данных через технологию Entity Framework
- [Deprecated] LINQ to Sql: технология доступа к данным в MS SQL Server
- LINQ to XML: операции с XML
- Parallel LINQ (PLINQ): параллельные запросы

По сути это набор extension методов для коллекций разного вида и QueryExpression синтаксис для них

Methods

- Select: определяет проекцию выбранных значений
- Where: определяет фильтр выборки
- OrderBy: упорядочивает элементы по возрастанию
- OrderByDescending: упорядочивает элементы по убыванию

- Join: соединяет две коллекции по определенному признаку
- GroupBy: группирует элементы по ключу
- ToLookup: группирует элементы по ключу, при этом все элементы добавляются в словарь
- Concat: объединяет две коллекции
- Zip: объединяет две коллекции в соответствии с определенным условием
- Except: возвращает разность двух коллекцию, то есть те элементы, которые содерятся только в одной коллекции
- Union: объединяет две однородные коллекции
- Intersect: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обоих коллекциях

- Reverse: располагает элементы в обратном порядке
- All: определяет, все ли элементы коллекции удовлятворяют определенному условию
- Any: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию
- Contains: определяет, содержит ли коллекция определенный элемент
- Distinct: удаляет дублирующиеся элементы из коллекции
- Count - количество элементов коллекции, которые удовлетворяют определенному условию
- Sum, Average, Min, Max - арифметические вычисления
- Take, Skip, TakeWhile, SkipWhile - пейджинг в выборке

- First: выбирает первый элемент коллекции
- FirstOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию
- Single: выбирает единственный элемент коллекции, если коллекция содердит больше или меньше одного элемента, то генерируется исключение
- SingleOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию
- ElementAt: выбирает элемент последовательности по определенному индексу
- ElementAtOrDefault: выбирает элемент коллекции по определенному индексу или возвращает значение по умолчанию, если индекс вне допустимого диапазона
- Last: выбирает последний элемент коллекции
- LastOrDefault: выбирает последний элемент коллекции или возвращает значение по умолчанию

```
var list = new List<string> { "First", "Second", "Third", "Fourth", "Fifth"};

IEnumerable<string> query = list.Where(x => x.Contains("i");
int count = query.Count();
string element = query.LastOrDefault();
var ordered = list.OrderBy(x => x).ToList();
```

Пример, как можно писать в функциональном стиле 😃

FizzBuzz - Counting from 1 to 100, and
every time a number is divisible by 3 calling "Fizz",
every time a number is divisible by 5 calling "Buzz" and
every time a number is divisible by 3 and 5, calling "FizzBuzz instead of the number

```
public void DoFizzBuzz()
{
    for (int i = 1; i <= 100; i++)
    {
        bool fizz = i % 3 == 0;
        bool buzz = i % 5 == 0;
        if (fizz && buzz)
            Console.WriteLine ("FizzBuzz");
        else if (fizz)
            Console.WriteLine ("Fizz");
        else if (buzz)
            Console.WriteLine ("Buzz");
        else
            Console.WriteLine (i);
    }
}
```

```
Enumerable.Range(1, 100)
    .Select(x =>
        (x % 15 == 0) ? "FIZZBUZZ"
        : (x % 5 == 0) ? "BUZZ"
        : (x % 3 == 0) ? "FIZZ"
        : x.ToString()
    ).ToList().ForEach(console.WriteLine);
```

Использование метода ForEach(x=> ...) вместо цикла foreach считается не очень

- ухудшает читаемость
- появляются замыкания

```
var list = new List<string> { "First", "Second", "Third", "Fourth", "Fifth"};

// лучше не так
list.ForEach(x=> Console.WriteLine(x));

// Делайте так:
foreach(var x in list)
    Console.WriteLine(x);
```

Deferred, Immediate Query Execution

Запросы делятся на отложенные и неотложенные:

- Отложенные запросы Where, OrderBy, Join, etc не выполняются сразу
- Реально выполнение запроса будет происходить только при вызове неотложенных запросов (Count, SingleOrDefault, First, ToList, etc) или при переборе foreach

```
string[] teams = { "First", "Second", "Third", "Fourth", "Fifth"};

IEnumerable<string> query = teams.Where(x=>x.Contains("i"));

Console.WriteLine(query.Count());  // 3
teams[1] = "Linq";
Console.WriteLine(query.Count());  // 4
```

Можно фильтровать, не выполняя запрос:

```
IEnumerable<User> users = getAllUsers;

if (filter.Type != CredentialType.Undefined)
    users = users.Where(c => c.CredentialType == credentialType);

if (! string.IsNullOrEmpty(filter.Credential))
    users = users.Where(c => c.Credential == filter.Credential);
```

Стандартный и Query Expressions синтакис запросов

Помимо вызова Extension методов можно испольозть Query Expressions синтаксис, который отдаленно напоминает tsql

```
from l in db.Licenses
join il in db.InstanceLicenses on l.Id equals il.LicenseId
where il.InstanceId == instanceId
select l
```