

Лекція 2.

ОСНОВИ LINQ

Література

1. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/getting-started-with-linq>
2. Anatomy of the Lambda Expression - <http://www.tutorialsteacher.com/linq/linq-lambda-expression>
3. Полное руководство по языку программирования C# 6.0 и платформе .NET 4.6, - <http://metanit.com/sharp/tutorial/15.1.php>

1. Основи LINQ

LINQ (Language-Integrated Query) - SQL-подібна мова запитів до джерела даних, розроблена Microsoft для платформи .Net Framework.

LINQ - наприкінці листопада 2007 р. разом з Visual Studio 2008.

Джерела даних:

- об'єкти, які реалізують інтерфейс IEnumerable (колекції, масиви);
- набори даних DataSet;
- документи XML.

Застосовується один підхід до вибірки даних.

Відлагодження та створення запитів - утиліта LINQPad

1. 1. LINQ за типами джерел даних

Запити LINQ за джерелами даних:

- **LINQ to Objects** - робота з масивами і колекціями;
- **LINQ to Entities** - створення запитів до БД за технологією Entity Framework;
- **LINQ to Sql** - створення запитів до БД MS SQL Server;
- **LINQ to XML** - створення запитів до файлів XML;
- **LINQ to DataSet** – робота з об'єктом DataSet;
- **Parallel LINQ (PLINQ)** - паралельні запити.

1.1.1. LINQ to Objects

Реалізація функціонального програмування за допомогою SQL-подібного синтаксису

Схема

from *змінна* **in** *набір_об'єктів*

<оператори, які формують вибірку>

select *змінна*;

Вирази LINQ мають строгий тип

2.1.2. Приклад. Фільтрація і впорядкування

```
string[] teams =  
{ "Баварія", "Боруссія", "Динамо",  
  "Манчестер Юнайтед", "ПСЖ", "Барселона" };
```

```
var selectedTeams = new List<string>();  
  
foreach(string s in teams)  
{  
    if (s.ToUpper().StartsWith("Б"))  
        selectedTeams.Add(s);  
}
```

```
selectedTeams.Dump();
```

```
string[] teams =  
{ "Баварія", "Боруссія", "Динамо",  
  "Манчестер Юнайтед", "ПСЖ", "Барселона" };
```

```
var selectedTeams = from t in teams // визначаємо кожне  
                    where t.ToUpper().StartsWith("Б")  
                    orderby t // упорядкування за зростанням  
                    select t; // вибираємо об'єкт
```

```
foreach (string s in selectedTeams)  
    Console.WriteLine(s);
```

Results λ SQL IL Tree

▲ List<String> (3 items) ▶
Баварія
Боруссія
Барселона

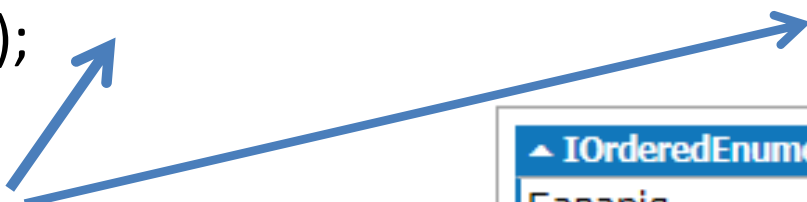
2.1.3. Методи розширення LINQ

```
string[] teams = { "Баварія", "Боруссія", "Динамо",  
"Манчестер Юнайтед", "ПСЖ", "Барселона" };
```

```
var selectedTeams1  
= from t in teams where t.ToUpper().StartsWith("Б") orderby t select t;  
selectedTeams1.Dump();
```

```
var selectedTeams2 =  
teams.Where(t=>t.ToUpper().StartsWith("Б")).OrderBy(t => t);  
selectedTeams2.Dump();
```

λ – вирази, делегати



▲ IEnumerable<String> (3 items) ▶
Баварія
Барселона
Боруссія

▲ IEnumerable<String> (3 items) ▶
Баварія
Барселона
Боруссія

2.1.4. Список методів розширення (1)

В інтерфейсі `IEnumerable` визначені методи:

- **Select** - проекція вибраних значень
- **Where** - фільтр елементів
- **OrderBy** - впорядкування елементів за зростанням
- **OrderByDescending** - впорядкування за спаданням
- **ThenBy** - додаткові критерії для впорядкування елементів за зростанням
- **ThenByDescending** - додаткові критерії для впорядкування елементів за спаданням
- **Join** - з'єднання двох колекцій за певною ознакою

Список методів розширення (2)

- **GroupBy**: групує елемент за ключем
- **ToLookup**: групує елементи за ключем, причому всі елементи додаються в словник
- **GroupJoin**: виконує одночасно з'єднання колекцій і групування елементів за ключем
- **Reverse**: розміщує елементи в зворотньому порядку
- **All**: перевіряє, чи всі елементи колекції задовільняють певну умову
- **Any**: перевіряє, чи є хочаб один елемент колекції, який задовільняє певну умову
- **Contains**: перевіряє, чи містить колекція вказаний елемент

Список методів розширення (3)

- **Distinct** – видаляє, елементи які повторюються, з колекції
- **Except** – віднімання (елементи, які містяться лише в одній колекції)
- **Union** - об'єднує дві однорідні колекції
- **Intersect** – перетин (елементи, які є в обох колекціях)
- **Count** - підраховує кількість елементів колекції, які задовольняють певній умові
- **Sum** - підраховує суму вказаних числових значень в колекції

Список методів розширення (4)

- **Average**: підраховує середнє числових значень в колекції
- **Min**: знаходить мінімальне значення
- **Max**: знаходить максимальне значення
- **Take**: вибирає певну кількість елементів
- **Skip**: пропускає певну кількість елементів
- **TakeWhile**: послідовно повертає елементи колекції, доки умова істинна
- **SkipWhile**: пропускає елементи в послідовності, доки вони задовольняють заданій умові, а потім повертає елементи, які залишились.
- **Concat**: об'єднує дві колекції

Список методів розширення (5)

- **Zip** - об'єднує дві колекції відповідно до визначеної умови
- **First** - вибирає перший елемент колекції
- **FirstOrDefault** - вибирає перший елемент колекції або повертає значення за замовчуванням
- **Single** - вибирає єдиний елемент колекції, якщо колекція містить більше або менше одного елемента, то сприймається як помилка і генерується виняткова ситуація
- **SingleOrDefault** - вибирає перший елемент або повертає значення за замовчуванням
- **ElementAt** - вибирає елемент за індексом

Список методів розширення (6)

- **ElementAtOrDefault** - вибирає елемент колекції за індексом або повертає значення за замовчуванням, якщо індекс за межами діапазону
- **Last** - вибирає останній елемент колекції
- **LastOrDefault** - вибирає останній елемент колекції або повертає значення за замовчуванням

2.1.3. Приклади

LINQ to Entity Framework. Фільтрація

```
var queryLondonCust = from cust in customers  
                        where cust.City == "London"  
                        select cust;
```

```
where cust.City=="London" && cust.Name == "Devon"
```

```
where cust.City == "London" || cust.City == "Paris"
```

LINQ. Групування

```
var queryCustomersByCity = from cust in customers  
                           group cust by cust.City;
```

```
var custQuery = from cust in customers  
               group cust by cust.City into custGroup  
               where custGroup.Count() > 2  
               orderby custGroup.Key  
               select custGroup;
```

LINQ. Впорядкування

```
var queryLondonCust = from cust in customers  
                        where cust.City == "London"  
                        orderby cust.Name ascending  
                        select cust;
```

```
where cust.City=="London" && cust.Name == "Devon«
```

```
where cust.City == "London" || cust.City == "Paris"
```


2. Lambda-вирази

C# 3.0(.NET 3.5) - анонсовано LINQ та засобів для інтерпретації лямбда-виразів.

Лямбда-вираз - це найкомпактніший спосіб запису анонімного методу за допомогою спеціального синтаксису.

Приклад

Анонімний метод перевіряє, чи студент підліток чи ні:

```
delegate(Student s) { return s.Age > 12 && s.Age < 20; };
```

```
public class Program
{
    delegate bool IsTeenAger(Student stud);
    public static void Main() {
        IsTeenAger isTeenAger =
            delegate(Student s) { return s.Age > 12 && s.Age < 20; }
        Student stud = new Student() { Age = 25 };
        Console.WriteLine(isTeenAger(stud));}}
}
```

```
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; } }
}
```

```
public class Program
{
    delegate bool IsTeenAger(Student stud);
    public static void Main()
    {
        IsTeenAger isTeenAger = s => s.Age > 12 && s.Age < 20;
        Student stud = new Student() { Age = 25 };
        Console.WriteLine(isTeenAger(stud));
    }
}

public class Student{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }}
```

The Lambda expression evolves from anonymous method by first removing the delegate keyword and parameter type and adding a lambda operator =>.

```
delegate(Student s) { return s.Age > 12 && s.Age < 20; };
```

© TutorialsTeacher.com

↓
1 - Remove Delegate and Parameter Type and add lamda operator =>

```
delegate(Student s) => { return s.Age > 12 && s.Age < 20; };
```

↓
(s) => { return s.Age > 12 && s.Age < 20; };

Lambda Expression from Anonymous Method

Also, we can remove parenthesis (), if we have only one parameter.

`(s) => { return s.Age > 12 && s.Age < 20; }`



2 - Remove curly bracket, return and semicolon

© TutorialsTeacher.com

`(s) => s.Age > 12 && s.Age < 20;`



3 - Remove Parenthesis around parameter if there is only one parameter

`s => s.Age > 12 && s.Age < 20;`

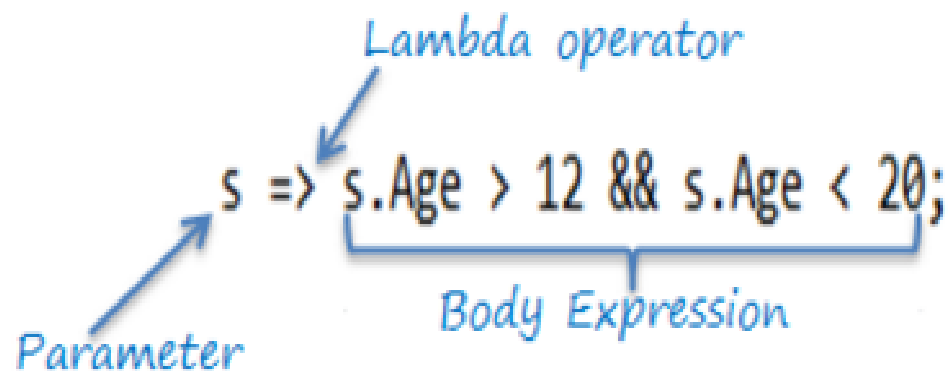
Lambda Expression from Anonymous Method

Lambda Expression `s => s.Age > 12 && s.Age < 20`

`s` is a parameter

`=>` is the lambda operator and

`s.Age > 12 && s.Age < 20` is the body expression



Lambda Expression Structure in C#

Lambda-вирази з багатьма параметрами

Приклад.

Описуємо декілька параметрів в lambda-виразі

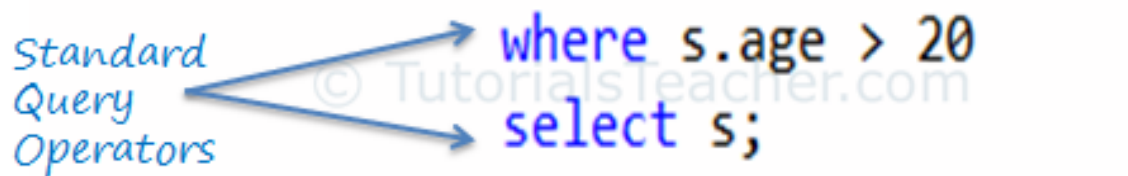
(s, youngAge) => s.Age >= youngage;

(Student s,int youngAge) => s.Age >= youngage;

LINQ and Lambda Expression

```
var students = from s in studentList
                where s.age > 20
                select s;
```

Standard Query Operators

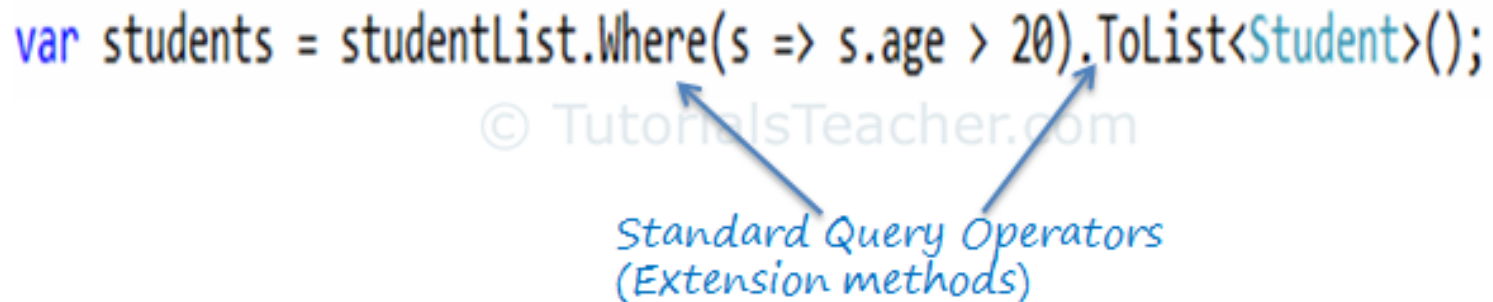


Standard Query Operators in Query Syntax

Standard Query Operators in Method Syntax:

```
var students = studentList.Where(s => s.age > 20).ToList<Student>();
```

Standard Query Operators
(Extension methods)



Standard Query Operators in Method Syntax

Принципи роботи IQueryable і LINQ-провайдерів даних

LINQ надають однотипні засоби для роботи як з колекціями об'єктів в пам'яті, так і з об'єктами в БД чи іншому віддаленому джерелі.

Наступний код майже ідентичний:

```
List<Apple> appleList;
```

```
DbSet<Apple> appleDbSet;
```

```
var applesFromList =
```

```
    appleList.Where(apple => apple.Color == "red").Take(10);
```

```
var applesFromDb =
```

```
    appleDbSet.Where(apple => apple.Color == "red").Take(10);
```

Інтерфейс IQueryable<T>

Для списку типу List<Apple> за допомогою foreach записи будуть відфільтровані за вказаною умовою, після чого будуть взяті перші 10 з них.

Для колекції типу DbSet<Apple> синтаксичне дерево запиту буде передано LINQ-провайдеру, який трансліює його в SQL-запит до бази даних, виконує, а потім формує для 10 знайдених записів об'єкти C# і та повертає.

Інтерфейс IQueryable <T>, призначений для створення LINQ-провайдерів до зовнішніх джерел даних.

Інтерфейси IEnumerable<T> та IQueryable<T>

В LINQ to Objects реалізовані методи розширень Where (), Select (), First (), Count () та інші інтерфейсу IEnumerable <T> з простору імен System.Collections.

Підхід не може застосовуватись для даних в БД або на віддаленому сервісі, оскільки вимагає попереднього завантаження всього набіру даних в додаток.

Набір методів-розширень до БД і даних віддалених сервісів описані в інтерфейсі IQueryable <T> і реалізовані в LINQ to SQL, LINQ to Entities та LINQ to OData Services з простору імен - System.Linq.

Інтерфейси IEnumerable<T> та IQueryable<T>

- IQueryable <T> - нащадок Enumerable <T>, успадковує методи IEnumerable <T>.
- List <T> реалізує методи IEnumerable <T>, DbSet <T> з Entity Framework - IQueryable <T>, тому запити з яблуками виконуються по-різному.

Інтерфейс IQueryable<T>

- Реалізації методів-розширень IQueryable <T> не містять логіки обробки даних - вони лише формують синтаксичну структуру з описом запиту, «нарощуючи» її при кожному новому виклику методу в ланцюжку.
- Коли викликається агрегатний метод (Count () і т.п.) або при використанні запиту в циклі foreach, запит передається провайдеру, інкапсульованому всередині конкретної реалізації IQueryable <T>, який перетворює запит в мову джерела даних і виконує його.

Для Entity Framework такою мовою є SQL,
для .Net-драйвера для MongoDB - пошуковий json-об'єкт і
т.д.

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries>

<http://qaru.site/questions/18913/what-is-the-difference-between-iqueryablet-and-ienumerablet>

Deferred vs Immediate Query Execution in LINQ

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries>

<http://qaru.site/questions/18913/what-is-the-difference-between-iqueryablet-and-ienumerablet>

<https://www.dotnetcurry.com/linq/750/deferred-vs-immediate-query-execution-linq>