# StartWith

string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };

var selectedTeams = from t in teams // определяем каждый объект из teams как t

where t.ToUpper().StartsWith("Б") //фильтрация по критерию

orderby t // упорядочиваем по возрастанию

select t; // выбираем объект

foreach (string s in selectedTeams)

Console.WriteLine(s);

string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };

var selectedTeams = teams.Where(t => t.ToUpper().StartsWith("Б")).OrderBy(t => t);

foreach (string s in selectedTeams)

Console.WriteLine(s);

# All methods

### Список используемых методов расширения LINQ

* **Select**: определяет проекцию выбранных значений
* **Where**: определяет фильтр выборки
* **OrderBy**: упорядочивает элементы по возрастанию
* **OrderByDescending**: упорядочивает элементы по убыванию
* **ThenBy**: задает дополнительные критерии для упорядочивания элементов возрастанию
* **ThenByDescending**: задает дополнительные критерии для упорядочивания элементов 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
* **GroupJoin**: выполняет одновременно соединение коллекций и группировку элементов по ключу
* **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
* **Except**: возвращает разность двух коллекцию, то есть те элементы, которые содератся только в одной коллекции
* **Union**: объединяет две однородные коллекции
* **Intersect**: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обоих коллекциях
* **Count**: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию
* **Sum**: подсчитывает сумму числовых значений в коллекции
* **Average**: подсчитывает cреднее значение числовых значений в коллекции
* **Min**: находит минимальное значение
* **Max**: находит максимальное значение
* **Take**: выбирает определенное количество элементов
* **Skip**: пропускает определенное количество элементов
* **TakeWhile**: возвращает цепочку элементов последовательности, до тех пор, пока условие истинно
* **SkipWhile**: пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы
* **Concat**: объединяет две коллекции
* **Zip**: объединяет две коллекции в соответствии с определенным условием
* **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

# SelectMany

var r = users.SelectMany(u => u.Languages,

(u, l) => new { User = u, Lang = l })

.Where(u => u.Lang == "английский" && u.User.Age < 28)

.Select(u => u.User);

# Select

List<User> users = new List<User>();

users.Add(new User { Name = "Sam", Age = 43 });

users.Add(new User { Name = "Tom", Age = 33 });

var names = from u in users select u.Name;

# Select anonymous type

var items = users.Select(u => new

{

FirstName = u.Name,

DateOfBirth = DateTime.Now.Year - u.Age

});

# Let

List<User> users = new List<User>()

{

new User { Name = "Sam", Age = 43 },

new User { Name = "Tom", Age = 33 }

};

var people = from u in users

let name = "Mr. " + u.Name

select new

{

Name = name,

Age = u.Age

};

# OrderBy

int[] numbers = { 3, 12, 4, 10, 34, 20, 55, -66, 77, 88, 4 };

var orderedNumbers = from i in numbers

orderby i

select i;

foreach (int i in orderedNumbers)

Console.WriteLine(i);

# OrderBy by class property

var sortedUsers = from u in users

orderby u.Name

select u;

# Two list selecting

List<User> users = new List<User>()

{

new User { Name = "Sam", Age = 43 },

new User { Name = "Tom", Age = 33 }

};

List<Phone> phones = new List<Phone>()

{

new Phone {Name="Lumia 630", Company="Microsoft" },

new Phone {Name="iPhone 6", Company="Apple"},

};

var people = from user in users

from phone in phones

select new { Name = user.Name, Phone = phone.Name };

foreach (var p in people)

Console.WriteLine("{0} - {1}", p.Name, p.Phone);

};

# ThenBy

var result = users.OrderBy(u => u.Name).ThenBy(u => u.Age).ThenBy(u => u.Name.Length);

var result = from user in users

orderby user.Name, user.Age, user.Name.Length

select user;

# Except

string[] soft = { "Microsoft", "Google", "Apple" };

string[] hard = { "Apple", "IBM", "Samsung" };

// разность множеств

var result = soft.Except(hard);

# Intersect

string[] soft = { "Microsoft", "Google", "Apple" };

string[] hard = { "Apple", "IBM", "Samsung" };

// пересечение множеств

var result = soft.Intersect(hard);

# Union|Concat|Distinct

string[] soft = { "Microsoft", "Google", "Apple" };

string[] hard = { "Apple", "IBM", "Samsung" };

// объединение множеств

var result = soft.Union(hard);

var result = soft.Concat(hard).Distinct();

# Aggragate

nt[] numbers = { 1, 2, 3, 4, 5 };

int query = numbers.Aggregate((x, y) => x - y);

int query = 1 - 2 - 3 - 4 – 5

# Count

int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };

int size = (from i in numbers where i % 2 == 0 && i > 10 select i).Count();

Console.WriteLine(size);

# Sum

int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };

List<User> users = new List<User>()

{

    new User { Name = "Tom", Age = 23 },

    new User { Name = "Sam", Age = 43 },

    new User { Name = "Bill", Age = 35 }

};

int sum1 = numbers.Sum();

decimal sum2 = users.Sum(n => n.Age);

# Skip and Take

int[] numbers = { -3, -2, -1, 0, 1, 2, 3 };

var result = numbers.Take(3);

foreach (int i in result)

Console.WriteLine(i);

var result = numbers.Skip(3);

# TakeWhile

string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };

foreach (var t in teams.TakeWhile(x => x.StartsWith("Б")))

Console.WriteLine(t);

# SkipWhile

string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };

foreach (var t in teams.SkipWhile(x => x.StartsWith("Б")))

Console.WriteLine(t);

# Group By

var phoneGroups = from phone in phones

group phone by phone.Company;

foreach (IGrouping<string, Phone> g in phoneGroups)

{

Console.WriteLine(g.Key);

foreach (var t in g)

Console.WriteLine(t.Name);

Console.WriteLine();

}

var phoneGroups = phones.GroupBy(p => p.Company)

                        .Select(g => new { Name = g.Key, Count = g.Count() });

# Вложенный запрос

var phoneGroups = phones.GroupBy(p => p.Company)

                        .Select(g => new

                        {

                            Name = g.Key,

                            Count = g.Count(),

                            Phones = g.Select(p =>p)

                        });

# Join

var result = players.Join(teams, // второй набор

p => p.Team, // свойство-селектор объекта из первого набора

t => t.Name, // свойство-селектор объекта из второго набора

(p, t) => new { Name = p.Name, Team = p.Team, Country = t.Country }); // результат

var result = from pl in players

join t in teams on pl.Team equals t.Name

select new { Name = pl.Name, Team = pl.Team, Country = t.Country };

# GroupJoin

var result2 = teams.GroupJoin(

players, // второй набор

t => t.Name, // свойство-селектор объекта из первого набора

pl => pl.Team, // свойство-селектор объекта из второго набора

(team, pls) => new // результирующий объект

{

Name = team.Name,

Country = team.Country,

Players = pls.Select(p => p.Name)

});

foreach (var team in result2)

{

Console.WriteLine(team.Name);

foreach (string player in team.Players)

{

Console.WriteLine(player);

}

Console.WriteLine();

}

# Zip

var result2 = players.Zip(teams,

(player, team) => new

{

Name = player.Name,

Team = team.Name,

Country = team.Country

});

foreach (var player in result2)

{

Console.WriteLine("{0} - {1} ({2})", player.Name, player.Team, player.Country);

Console.WriteLine();

}

# All|Any

List<User> users = new List<User>()

{

new User { Name = "Tom", Age = 23 },

new User { Name = "Sam", Age = 43 },

new User { Name = "Bill", Age = 35 }

};

bool result1 = users.All(u => u.Age > 20); // true

bool result1 = users.Any(u => u.Age < 20); //false хотя бы один

# Отложенное и немедленное выполнение

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

// выполнение LINQ-запроса

foreach (string s in selectedTeams)

Console.WriteLine(s);

То есть фактическое выполнение запроса происходит не в строке определения: var selectedTeams = from t..., а при переборе в цикле foreach.

После определения запроса он может выполняться множество раз. И до выполнения запроса источник данных может изменяться. Чтобы более наглядно увидеть это, мы можем изменить какой-либо элемент до перебора выборки:

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

// изменение массива после определения LINQ-запроса

teams[1] = "Ювентус";

// выполнение LINQ-запроса

Немедленное выполнение запроса

С помощью ряда методов мы можем применить немедленное выполнение запроса.Это методы, которые возвращают одно атомарное значение или один элемент.Например, Count(), Average(), First() / FirstOrDefault(), Min(), Max() и т.д.Например, метод Count() возвращает числовое значение, которое представляет количество элементов в полученной последовательности.А метод First() возвращает первый элемент последовательности. Но чтобы выполнить эти методы, вначале надо получить саму последовательность, то есть результат запроса, и пройтись по ней циклом foreach, который вызывается неявно внутри структуры запроса.

Рассмотрим пример с методом Count(), который возвращает число элементов последовательности:

string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };

// определение и выполнение LINQ-запроса

int i = (from t in teams

where t.ToUpper().StartsWith("Б")

orderby t

select t).Count();

Console.WriteLine(i); //3

teams[1] = "Ювентус";

Console.WriteLine(i); //3

--------

string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };

// выполнение LINQ-запроса

var selectedTeams = (from t in teams

where t.ToUpper().StartsWith("Б")

orderby t

select t).ToList<string>();

// изменение массива никак не затронет список selectedTeams

teams[1] = "Ювентус";

foreach (string s in selectedTeams)

Console.WriteLine(s);