LINQ

Основы LINQ

LINQ

- **LINQ** (Language-Integrated Query) представляет простой и удобный язык запросов к источнику данных. В качестве источника данных может выступать объект, реализующий интерфейс IEnumerable (например, стандартные коллекции, массивы) Но вне зависимости от типа источника LINQ позволяет применить ко всем один и тот же подход для выборки данных.

Разновидности **LINQ**

- **LINQ to Objects**: применяется для работы с массивами и коллекциями
- **LINQ to Entities**: используется при обращении к базам данных через технологию Entity Framework
- **LINQ to Sql**: технология доступа к данным в MS SQL Server
- **LINQ to XML**: применяется при работе с файлами XML
- **LINQ to DataSet**: применяется при работе с объектом DataSet
- **Parallel LINQ (PLINQ)**: используется для выполнения параллельной запросов

LINQ to Objects

- В чем же удобство LINQ? Посмотрим на простейшем примере. Выберем из массива строки, начинающиеся на определенную букву и отсортируем полученный список:

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

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

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

LINQ to Objects

- Теперь проведем те же действия с помощью LINQ:

```
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);
```

- Итак, код стал меньше и проще. В принципе все выражение можно было бы записать в одну строку:

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

Простейший запрос LINQ

- Простейшее определение запроса LINQ выглядит следующим образом:

```
from переменная in набор_объектов
select переменная;
```

- Итак, что делает этот запрос LINQ? Выражение from t in teams проходит по всем элементам массива teams и определяет каждый элемент как t. Используя переменную t мы можем проводить над ней разные операции

Примененные операторы в запросе LINQ

- С помощью оператора **where** проводится фильтрация объектов, и если объект соответствует критерию (в данном случае начальная буква должна быть "Б"), то этот объект передается дальше.

- Оператор **orderby** упорядочивает по возрастанию, то есть сортирует выбранные объекты.

- Оператор **select** передает выбранные значения в результирующую выборку.

Методы расширения LINQ

- Кроме стандартного синтаксиса **from .. in .. select** для создания запроса LINQ мы можем применять специальные методы расширения. Как правило, эти методы реализуют ту же функциональность, что и операторы LINQ типа **where** или **orderby**.

```
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);
```

Фильтрация выборки

- Для выбора элементов из некоторого набора по условию используется метод **Where**. Например, выберем все четные элементы, которые больше 10.

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

IEnumerable<int> evens = from i in numbers
                         where i%2==0 && i>10
                         select i;
foreach (int i in evens)
    Console.WriteLine(i);
```

Фильтрация выборки

- Тот же запрос с помощью метода расширения:

```
int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
IEnumerable<int> evens = numbers.Where(i => i % 2 == 0 && i > 10);
```

Сортировка

- Для сортировки набора данных по возрастанию используется оператор **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);
```

Сортировка

- По умолчанию оператор **orderby** производит сортировку по возрастанию. Однако с помощью ключевых слов **ascending** (сортировка по возрастанию) и **descending** (сортировка по убыванию) можно явным образом указать направление сортировки:

```
var sortedUsers = from u in users
                  orderby u.Name descending
                  select u;
```

Сортировка

- Вместо оператора **orderby** можно использовать методы расширения **OrderBy или OrderByDescending**:

```
int[] numbers = { 3, 12, 4, 10, 34, 20, 55, -66, 77, 88, 4 };
IEnumerable<int> sortedNumbers = numbers.OrderBy(i=>i);

List<User> users = new List<User>()
{
    new User { Name = "Tom", Age = 33 },
    new User { Name = "Bob", Age = 30 },
    new User { Name = "Tom", Age = 21 },
    new User { Name = "Sam", Age = 43 }
};
var sortedUsers = users.OrderBy(u=>u.Name);
```

```
var sortedUsers = users.OrderByDescending(u=>u.Name);
```

Разность множеств

- С помощью метода **Except** можно получить разность двух множеств:

```
string[] soft = { "Microsoft", "Google", "Apple"};
string[] hard = { "Apple", "IBM", "Samsung"};

// разность множеств
var result = soft.Except(hard);

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

Пересечение множеств

- Для получения пересечения множеств, то есть общих для обоих наборов элементов, применяется метод **Intersect**:

```
string[] soft = { "Microsoft", "Google", "Apple"};
string[] hard = { "Apple", "IBM", "Samsung"};

// пересечение множеств
var result = soft.Intersect(hard);

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

Объединение множеств

- Для объединения двух множеств используется метод **Union**. Его результатом является новый набор, в котором имеются элементы, как из одного, так и из второго множества. Повторяющиеся элементы добавляются в результат только один раз:

```
string[] soft = { "Microsoft", "Google", "Apple"};
string[] hard = { "Apple", "IBM", "Samsung"};

// объединение множеств
var result = soft.Union(hard);

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

Удаление дубликатов

- Для удаления дублей в наборе используется метод **Distinct**:

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

- Последовательное применение методов **Concat** и **Distinct** будет подобно действию метода **Union**.

Агрегатные операции

- К агрегатным операциям относят различные операции над выборкой, например, получение числа элементов, получение минимального, максимального и среднего значения в выборке, а также суммирование значений.
- **Count**
- **Sum**
- **Min**
- **Max**
- **Average**

Получение размера выборки. Метод Count

- Для получения числа элементов в выборке используется метод **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);
```

Получение суммы

- Для получения суммы значений применяется метод **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);
```

Максимальное, минимальное и среднее значения

- Для нахождения минимального значения применяется метод **Min()**, для получения максимального - метод **Max()**, а для нахождения среднего значения - метод **Average()**. Их действие похоже на методы **Sum** и **Count**:

```
int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
int min1 = numbers.Min();

int max1 = numbers.Max();

double avr1 = numbers.Average();
```

Группировка

- Для группировки данных по определенным параметрам применяется оператор **group by** или метод **GroupBy()**:

```
var phoneGroups = from phone in phones
                  group phone by phone.Company;
```

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

Методы Skip и Take

- Метод **Skip()** пропускает определенное количество элементов.
- метод **Take()** извлекает определенное число элементов.
- Извлечем три первых элемента:

```
int[] numbers = { -3, -2, -1, 0, 1, 2, 3 };
var result = numbers.Take(3);

foreach (int i in result)
    Console.WriteLine(i);
```

- Выберем все элементы, кроме первых трех:

```
var result = numbers.Skip(3);
```

Методы TakeWhile и SkipWhile

- Метод **TakeWhile** выбирает цепочку элементов, начиная с первого элемента, пока они удовлетворяют определенному условию:

```
string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };
foreach (var t in teams.TakeWhile(x=>x.StartsWith("Б")))
    Console.WriteLine(t);
```

- В подобном русле действует метод **SkipWhile**. Он пропускает цепочку элементов, начиная с первого элемента, пока они удовлетворяют определенному условию:

```
string[] teams = { "Бавария", "Боруссия", "Реал Мадрид", "Манчестер Сити", "ПСЖ", "Барселона" };
foreach (var t in teams.SkipWhile(x=>x.StartsWith("Б")))
    Console.WriteLine(t);
```

Методы All, Any и Contains

- Метод **All** проверяет, соответствуют ли все элементы условию. Например, узнаем, у всех ли пользователей возраст превышает 20:

```
List<User> users = new List<User>()
{
    new User { Name = "Tom", Age = 23 },
    new User { Name = "Bill", Age = 35 }
};

bool result1 = users.All(u => u.Age > 20); // true
if (result1)
    Console.WriteLine("У всех пользователей возраст больше 20");
else
    Console.WriteLine("Есть пользователи с возрастом меньше 20");
```

Методы All, Any и Contains

- Метод **Any** действует подобным образом, только позволяет узнать, соответствует ли хотя бы один элемент коллекции определенному условию:

```
bool result1 = users.Any(u => u.Age < 20); //false
if (result1)
    Console.WriteLine("Есть пользователи с возрастом меньше 20");
else
    Console.WriteLine("У всех пользователей возраст больше 20");
```

Методы All, Any и Contains

- Метод **Contains** проверяет содержит ли последовательность указанный элемент:

```
string[] fruits = { "apple", "banana", "mango", "orange", "passionfruit", "grape" };

string fruit = "mango";

bool result = fruits.Contains(fruit); //true
if (result)
    Console.WriteLine("Манго есть");
else
    Console.WriteLine("Манго нет");
```