ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР
ПРОГРАММИРОВАНИЯ И ВЫСОКИХ ТЕХНОЛОГИЙ

2020г.

LINQ (Language-Integrated Query)

AGENDA

- ✓ Основные принципы LINQ
- ✓ Выражения и деревья выражений. Тип Expression
- ✓ Linq To Object
- ✓ Виды синтаксиса LINQ
- ✓ Наиболее часто-используемые конструкции LINQ

Основные принципы LINQ

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

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 объектов называется последовательностью (**sequence**). Большинство последовательностей **LINQ** имеют тип **IEnumerable<T>**, 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**. Очень многие методы **LINQ** возвращают **IEnumerable<T>**.

Основные принципы LINQ

Существует несколько разновидностей **LINQ**:

- **LINQ to Objects** – применяется для работы с массивами и коллекциями. Используется интерфейс **IEnumerable<T>.**
- **LINQ to Entities** – применяется при обращении к базам данных через технологию **Entity Framework.** Отделяет сущностную объектную модель от физической базы данных, вводя логическое отображение между ними двумя.
- **LINQ to Sql** – технология доступа к данным в **MS SQL Server**. Используется интерфейс **IQueryable<T>.**
- **LINQ to XML** – ориентирован на работу с файлами **XML**.
- **LINQ to DataSet** – применяется при работе с объектом **DataSet**.
- **Parallel LINQ (PLINQ)** – применяется для выполнения параллельных запросов.

Основные принципы LINQ

LINQ включает в себя около 50 стандартых операций запросов, разделяемых на 2 большие группы по способу выполнения запроса: отложенное и немедленное выполнение.

При отложенном выполнении **LINQ-выражение** не выполняется, пока не будет произведена итерация или перебор по выборке.

Фактически **LINQ-запрос** разбивается на три этапа:

- Получение источника данных.
- Создание запроса.
- Выполнение запроса и получение его результатов.

После определения запроса он может выполняться множество раз. И до выполнения запроса источник данных может изменяться.

Основные принципы LINQ

Кроме того, запросы **LINQ** предлагают три основных преимущества перед традиционными **foreach** циклами:

- Они более лаконичны и удобочитаемы, особенно при фильтрации нескольких условий.
- Они предоставляют мощные возможности фильтрации, упорядочивания и группировки с минимумом кода приложения.
- Их можно перенести на другие источники данных с небольшими изменениями или без них.

В целом, чем сложнее операция, которую необходимо выполнить с данными, тем больше преимуществ получаете, используя **LINQ** вместо традиционных методов итераций.

Выражения и деревья выражений

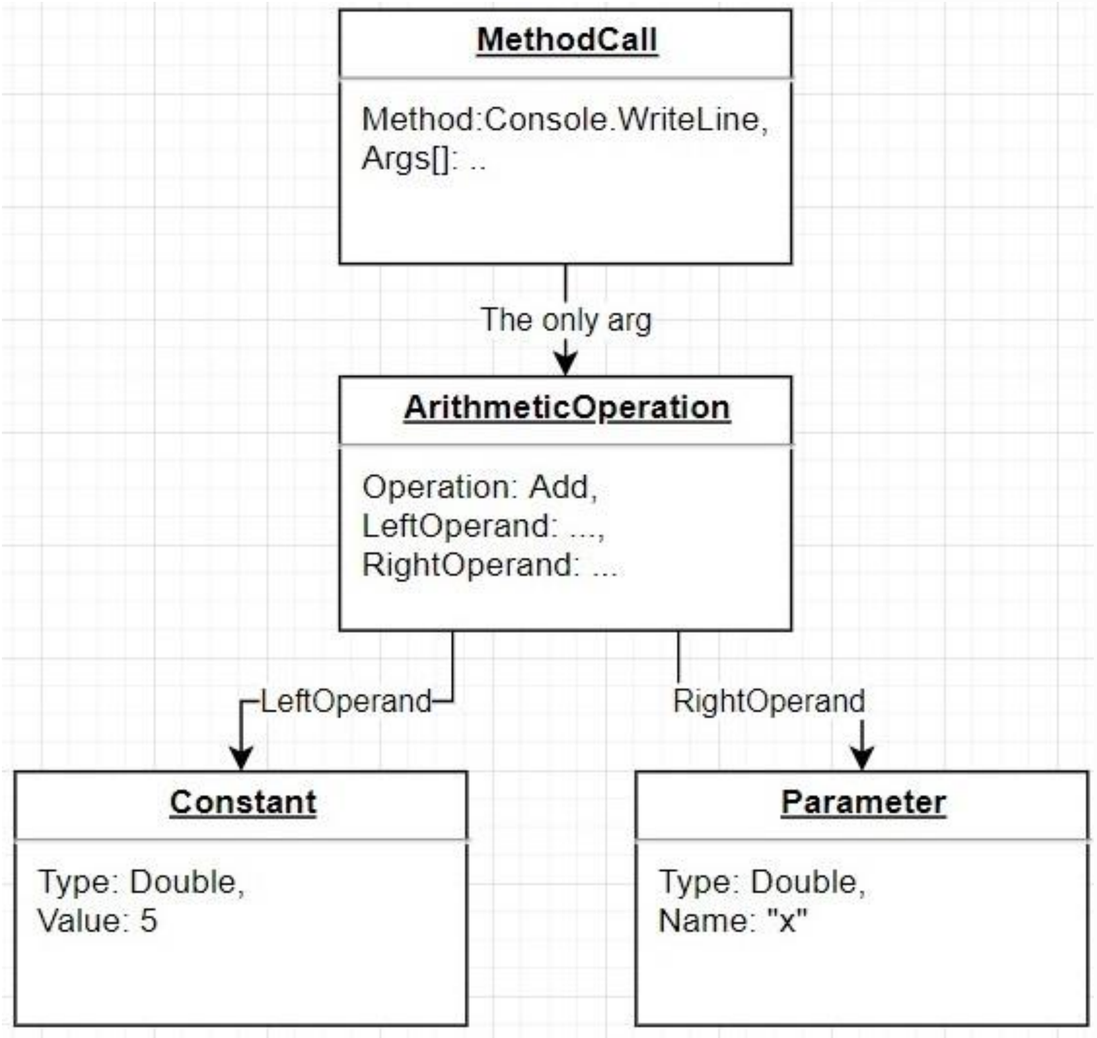
Дерево выражения (**expression tree**) – представление в древовидной форме данных лямбда-выражения операции запроса. Эти представления деревьев выражений могут быть вычислены все сразу, так что единственный запрос может быть построен и выполнен на одном источнике данных, таком как база данных.

В **C#** под деревьями выражений понимают тип **Expression** или любой из его наследников. При обычном раскладе выражения представляются в виде выполняемых инструкций. Тип **Expression** позволяет представить выражение (как правило лямбда-вырожения) как данные, организованные в виде древовидной структуры, к которой пользователь имеет доступ. Древовидный способ организации информации об алгоритме и название класса и дают нам «деревья выражений».

Используются они, в провайдерах баз данных. Применение очевидно - распарсить дерево выражений, понять, что там должно выполниться и составить по этому описанию **SQL**. Деревья выражений также нашли применение в **DLR** (**dynamic language runtime**). Разработчики компилятора используют их при обеспечении совместимости между динамической средой и **.Net**, вместо генерации **MSIL**.

Выражения и деревья выражений

```
(x) => Console.WriteLine(x+5);
```



Выражения и деревья выражений

Тип **Expression** – предоставляет базовый класс, от которого происходят классы, представляющие узлы дерева выражений. Он также содержит static фабричные методы для создания различных типов узлов.

Существуют 2 основных способа создания деревьев выражений:

- Создание деревьев выражений через статические методы класса **Expression**.
- Использование лямбда выражения, компилирующееся в **Expression**.

Компилятор может генерировать для лямбда-выражения операции две вещи – **IL-код** и **дерево выражения**.

Если операция объявлена для приема делегата метода, будет сгенерирован **IL-код**. Если же операция объявлена для приема выражения делегата, будет создано **дерево выражения**.

Linq To Object

LINQ to Objects – название, данное API-интерфейсу **IEnumerable<T>** для стандартных операций запросов. **LINQ to Objects** позволяет выполнять запросы к находящимся в памяти коллекциям данных.

IEnumerable<T> – интерфейс, реализуемый всеми классами обобщенных коллекций **C#**. Этот интерфейс позволяет выполнять перечисление элементов коллекций.

Последовательность – это термин для обозначения коллекции, реализующей интерфейс **IEnumerable<T>**. Если есть переменная типа **IEnumerable<T>**, то можно сказать, что имеется последовательность элементов типа **T**.

Большинство стандартных операций запросов представляют собой расширяющие методы в статическом классе **System.Linq.Enumerable** и прототипированы с **IEnumerable<T>** в качестве первого аргумента.

Виды синтаксиса LINQ

LINQ поддерживает два вида синтаксиса:

- Синтаксис запросов:
  Такой синтаксис всегда должен начинаться с **from in** и заканчиваться **select** или **group**.

```
int[] MyArray = { -1, 2, 3, -4 };
var res = from n in MyArray
          where n > 0
          select n; // 2, 3
```

- Синтаксис методов:
  Все стандартные методы реализованы как расширяющие, большинство из них принимает в параметр лямбда-выражение или делегат.

```
int[] MyArray = { -5, 2, 3, -11 };
var res = MyArray.Where(n => n > 0); // 2, 3
```

Виды синтаксиса LINQ

- Методы можно применять один за другим.
- Оба вида синтаксиса можно использовать вместе.

```
int[] MyArray = { -1, 2, 6, 8, 3, -4 };
var res = from n in MyArray.Where(x => x < 5) // -1, 2, 3, -4
          where n > 0
          select n; // 2, 3
```

Поскольку записывать запросы **LINQ** можно с использованием либо синтаксиса запросов, либо синтаксиса методов, может возникнуть вопрос о том, какой из них следует выбрать. В большинстве случаев это вопрос персональных предпочтений, если только необходимые операции запроса поддерживаются синтаксисом методов. Не все операции поддерживаются им, поэтому когда применяются неподдерживаемые операции, следует обратиться к синтаксису с вызовом методов через точечную нотацию.

Часто-используемые конструкции LINQ

Конструкции **where** – используется для отбора из последовательности тех элементов, которые удовлетворяют некоторому условию.

Конструкции **join** – используется для соединения двух наборов данных по некоторому признаку.

Конструкции **order by** – используется для сортировки элеметов последовательности.

Конструкции **group** – используется для группировки элемента последовательности по некоторому признаку.

Конструкции **select** – используется для обеспечения вывода элемента последовательности в видоизмененном виде.

Задание

Дана последовательность-> [11,-20,-5,4,5,8,-1,9,2,0,-11,15,3,-3,4,0,20]

1. Извлечь из нее все положительные числа, сохранив их исходный порядок следования.
2. Извлечь из нее все нечетные числа, сохранив их исходный порядок следования и удалив все вхождения повторяющихся элементов, кроме первых.
3. Извлечь из нее все четные отрицательные числа, поменяв порядок извлеченных чисел на обратный.
4. Извлечь из нее все положительные двузначные числа, отсортировав их по возрастанию.
5. Создайте две коллекции: Cars (содержит тип Car, который имеет поля: Id, Name, Age) и Buyers (содержит модель Buyers с полями: Id, Name, CarId). Выведите на консоль информацию об одном покупателе и информацию о машине, которую он купил. Правило выбора покупателя говорит о том, что его имя должно быть первым при сортировке по возрастанию. Один покупатель может купить одну машину.
6. Один покупатель может купить много машину.