Операции Average и Aggregate

LINQ (../../base/level1/info_linq.php) --- LINQ to Objects (../level1/linq_index.php) --- Операции Average и Aggregate

Average

Операция Average возвращает среднее арифметическое числовых значений элементов входной последовательности. Эта операция имеет два прототипа, описанные ниже:

Первый прототип Average

C#

```
public static Result Average(
      this IEnumerable<Numeric> source);
```

Тип Numeric должен быть одним из int, long, double или decimal, либо одним из их допускающих null эквивалентов: int?, long?, double? или decimal?. Если тип Numeric — int или long, то типом Result будет double. Если же тип Numeric — int? или long?, то типом Result будет double?. В противном случае тип Result совпадает с типом Numeric. Первый прототип операции Average перечисляет входную последовательность source элементов типа Numeric, вычисляя их среднее значение.

Второй прототип Average

Второй прототип операции Average перечисляет входную последовательность source элементов и определяет среднее значение членов элементов, возвращаемых функцией selector для каждого элемента входной последовательности.

C#

```
public static Result Average<T>(
      this IEnumerable<T> source,
      Func<T, Numeric> selector);
```

Если любой из аргументов равен null, генерируется исключение ArgumentNullException. Если сумма усредняемых значений превышает емкость long для типов Numeric — int, int?, long и long?, генерируется исключение OverflowException.

Начнем с примера применения первого прототипа, который показан ниже. В этом примере используется операция Range, чтобы создать последовательность целых чисел, а затем подсчитать их среднее значение:

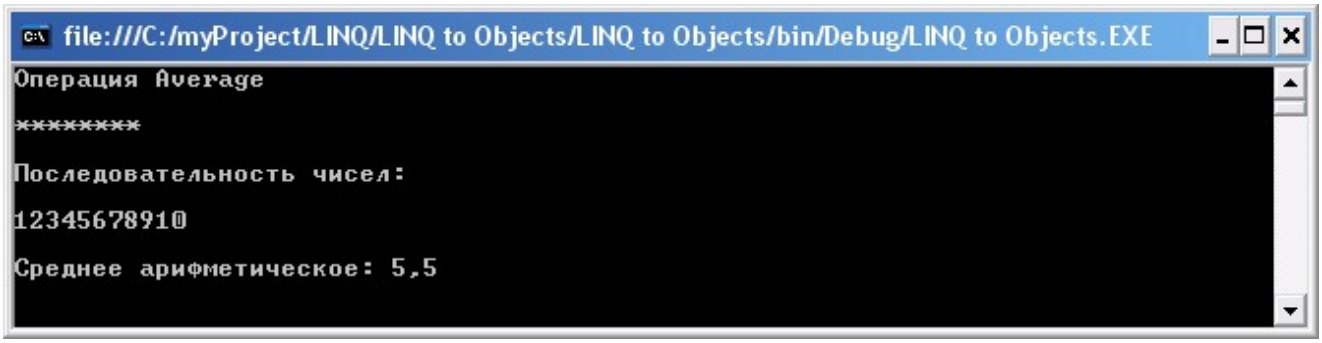
C#

```
Console.WriteLine("Операция Average\n\n********\n");

IEnumerable<int> nums = Enumerable.Range(1, 10);
Console.WriteLine("Последовательность чисел: \n");

foreach (int i in nums)
        Console.Write(i);

double average = nums.Average();
Console.WriteLine("\n\nСреднее арифметическое: " + average);
```



Теперь попробуем воспользоваться вторым прототипом, который обращается к членам элементов. В примере, приведенном ниже, используется общий класс EmployeeOptionEntry (../level2/2_0.php):

C#

Пройди тесты

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

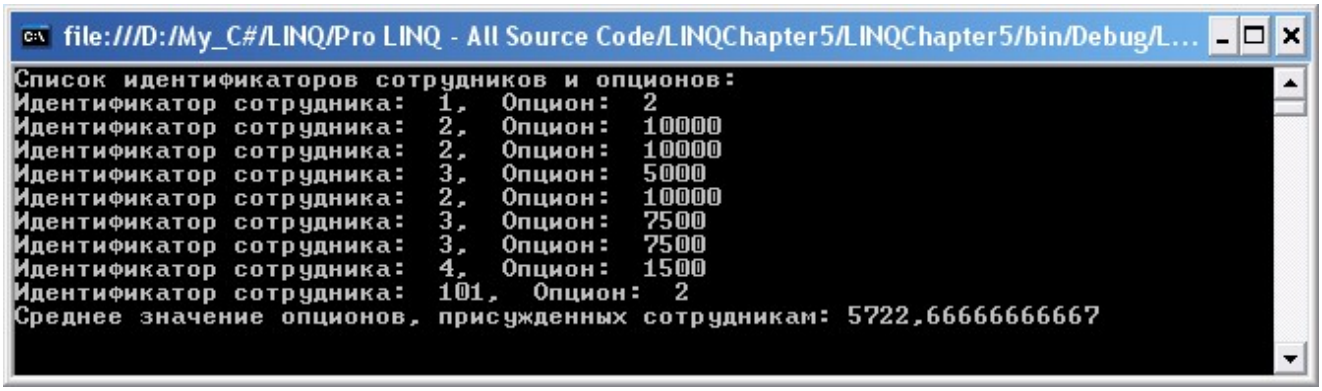
.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

```
IEnumerable<EmployeeOptionEntry> options =
        EmployeeOptionEntry.GetEmployeeOptionEntries();

      Console.WriteLine("Список идентификаторов сотрудников и опционов:");
      foreach (EmployeeOptionEntry eo in options)
          Console.WriteLine("Идентификатор сотрудника:  {0},  Опцион:  {1}", eo.id, eo.opti
onsCount);

      // Теперь получим среднее значение опционов
      double optionAverage = options.Average(o => o.optionsCount);
      Console.WriteLine("Среднее значение опционов, присужденных сотрудникам: {0}", optionA
verage);
```

Сначала извлекаются объекты EmployeeOptionEntry. Затем выполняется перечисление последовательности объектов с отображением каждого из них. В конце вычисляется и отображается среднее значение опционов. Результат работы этого кода выглядит следующим образом:



Aggregate

Операция Aggregate выполняет указанную пользователем функцию на каждом элементе входной последовательности, передавая значение, возвращенное этой функцией для предыдущего элемента, и возвращая ее значение для последнего элемента. Эта операция имеет два прототипа, которые описаны ниже:

Первый прототип Aggregate

C#

```
public static T Aggregate<T>(
    this IEnumerable<T> source,
    Func<T, T, T> func);
```

В этой версии прототипа операция Aggregate перечисляет каждый элемент входной последовательности source, вызывая делегат метода func на каждом из них, передавая значение, возвращенное им же на предыдущем элементе, и, наконец, помещая значение, возвращенное func, во внутренний аккумулятор, который затем передается на обработку следующего элемента. Первый элемент сам передается в качестве входного значения делегату метода func.

Второй прототип Aggregate

Второй прототип операции Aggregate ведет себя так же, как и первая версия, за исключением того, что принимает также начальное значение, служащее входным при первоначальном вызове делегата метода func, вместо самого первого элемента.

C#

```
public static U Aggregate<T, U> (
     this IEnumerable<T> source,
     U seed,
     Func<U, T, U> func);
```

Если любой из аргументов равен null, генерируется исключение ArgumentNullException. Если последовательность source пуста, генерируется исключение InvalidOperationException, но только для первого прототипа Aggregate, когда не указано никакого начального значения.

Начнем с примера применения первого прототипа, который приведен ниже. В этом примере вычисляется факториал числа 5. Факториал — это произведение всех положительных целых чисел, меньших или равных заданному. Итак, 5! будет равно 1*2*3*4*5. Похоже, что для этого вычисления можно использовать операции Range и Aggregate:

C#

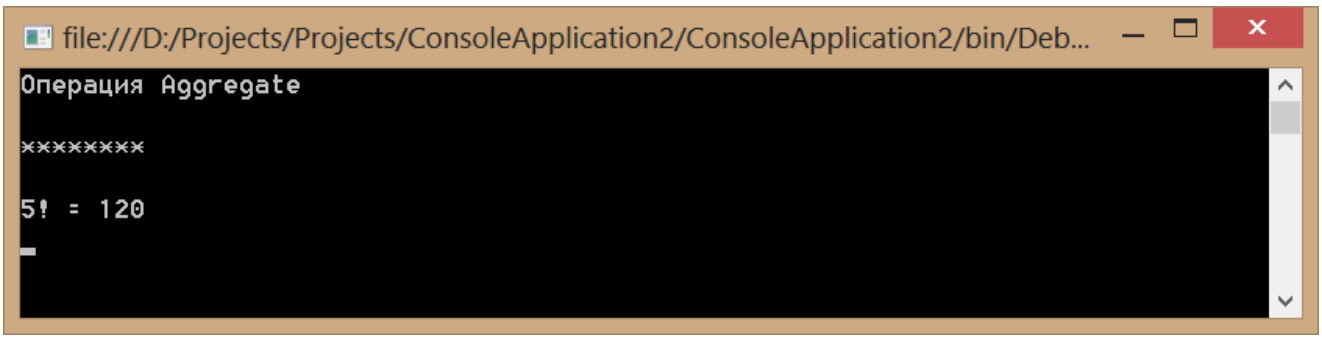
```
Console.WriteLine("Операция Aggregate\n\n********\n");

int N = 5;
int agg = Enumerable
         .Range(1, N)
         .Aggregate((av, e) => av * e);
Console.WriteLine("{0}! = {1}",N,agg);
```

Пройди тесты

В этом коде генерируется последовательность, содержащая целые числа от 1 до 5, для чего используется операция Range. Затем вызывается операция Aggregate, которой передается лямбда-выражение, умножающее переданное агрегатное

значение на сам переданный элемент. Результат выглядит следующим образом:



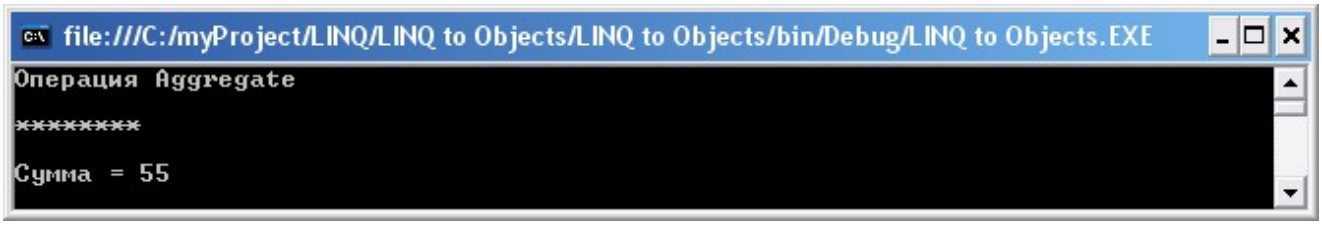
Применяя эту версию операции Aggregate, следует проявлять осторожность, чтобы первый элемент не был обработан дважды, поскольку он передается в качестве входного для первого элемента. В предыдущем примере первый вызов лямбда-выражения func должен получить 1 и 1. Поскольку два этих значения только перемножаются, и оба они равны 1, никакого вредного побочного эффекта нет. Но если бы нужно было складывать два значения, то в итоговую сумму первый элемент был бы включен дважды.

Для примера использования второго прототипа написана собственная версия операции Sum:

C#

```
int agg = Enumerable
                        .Range(1, 10)
                        .Aggregate(0, (s, i) => s + i);
Console.WriteLine("Сумма = " + agg);
```

Обратите внимание, что в качестве начального значения при вызове операции Aggregate был передан 0. Ниже показан результат:



Пройди тесты

Назад (3_9.php)	C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)

Пройди тесты

C# тест (легкий) (https://professorweb.ru/test/c-sharp-test.html)

.NET тест (средний) (https://professorweb.ru/test/asp-test.html)