Операции Count, LongCount и Sum

Count

Операция Count возвращает количество элементов во входной последовательности. Эта операция имеет два прототипа, описанные ниже:

Первый прототип Count

C#

```
public static int Count<T>(
      this IEnumerable<T> source);
```

Этот прототип операции Count возвращает общее количество элементов во входной последовательности, проверяя сначала, реализует ли она интерфейс ICollection<T>, и если да, то получает счетчик последовательности через реализацию этого интерфейса. Если же входная последовательность source не реализует интерфейс ICollection<T>, операция Count перечисляет всю эту последовательность, подсчитывая количество элементов.

Второй прототип Count

Второй прототип операции Count перечисляет входную последовательность source и подсчитывает все элементы, которые заставляют делегат метода predicate вернуть true:

C#

```
public static int Count<T>(
      this IEnumerable<T> source,
      Func<T, bool> predicate);
```


Если любой из аргументов равен null, генерируется исключение ArgumentNullException. Если значение count превышает Int32.MaxValue, генерируется исключение OverflowException.

Код ниже начинается с использования первого прототипа. Сколько элементов содержится в последовательности cars? Затем используется второй прототип, подсчитывая количество машин, начинающихся с "A":

C#

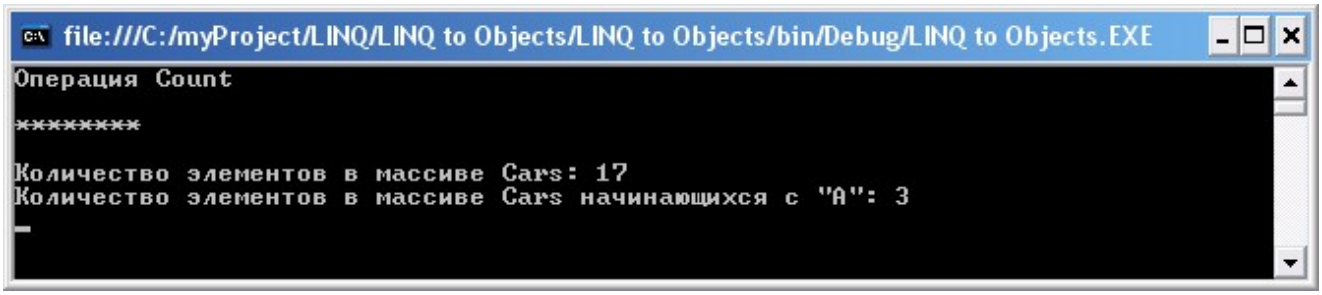
```
string[] cars = { "Alfa Romeo", "Aston Martin", "Audi", "Nissan", "Chevrolet",  "Chrysler",
"Dodge", "BMW",
                            "Ferrari",  "Bentley", "Ford", "Lexus", "Mercedes", "Toyota",
"Volvo", "Subaru", "Жигули :)"};

Console.WriteLine("Операция Count\n\n********\n");

int count = cars.Count();
Console.WriteLine("Количество элементов в массиве Cars: " + count);

count = cars.Count(s => s.StartsWith("A"));
Console.WriteLine("Количество элементов в массиве Cars начинающихся с \"A\": " + count);
```

Результат работы этого кода:



LongCount

Операция LongCount возвращает количество элементов входной последовательности как значение типа long. Эта операция имеет два прототипа, полностью идентичных прототипам операции Count, за тем лишь исключением, что они возвращают число элементов, имеющих тип long, а не int.

Давайте рассмотрим пример использования операции LongCount. Можно было бы просто повторить тот же пример, который сопровождал описание операции Count, изменив соответствующие места на тип long, но это не слишком наглядно. Поскольку нереально описать достаточно длинную статическую последовательность, для которой понадобилась бы операция LongCount, она генерируется с помощью стандартной операции запроса.

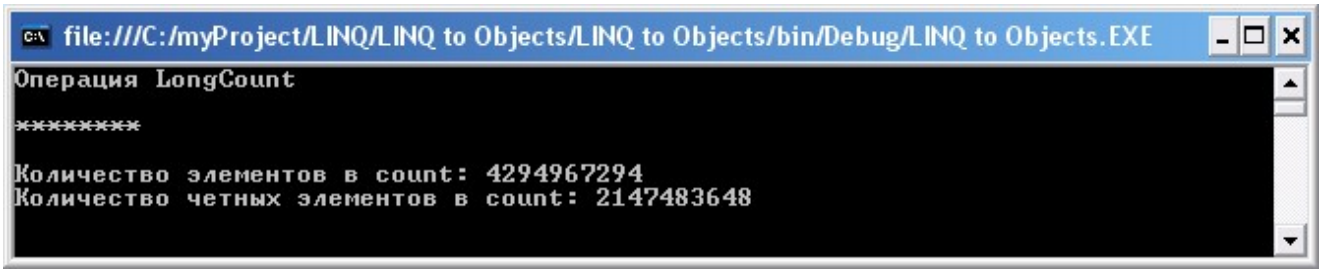
C#

```
long count = Enumerable.Range(0, int.MaxValue)
                .Concat(Enumerable.Range(0, int.MaxValue)).LongCount();
Console.WriteLine("Количество элементов в count: " + count);

// Используем второй прототип
count = Enumerable.Range(0, int.MaxValue)
       .Concat(Enumerable.Range(0, int.MaxValue))
       .LongCount(s => s % 2 == 0);
Console.WriteLine("Количество четных элементов в count: " + count);
```

Как видите, были сгенерированы две последовательности с использованием операции Range (../level2/2_12.php) и соединены вместе с помощью операции Concat (../level2/2_4.php).

Перед запуском примера наберитесь терпения — он выполняется долго, возможно, даже несколько минут. Например, на машине с четырехядерным процессором и 4 Гбайт памяти для этого потребовалось около минуты. В конце концов, будут сгенерированы две последовательности, каждая по 2 147 483 647 элементов. Ниже показан результат:



Sum

Операция Sum возвращает сумму числовых значений, содержащихся в элементах последовательности. Эта операция имеет два прототипа, описанные ниже:

Первый прототип Sum

C#

```
public static Numeric Sum(
    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?.

Первый прототип операции Sum возвращает сумму всех элементов входной последовательности source. Пустая последовательность приведет к возврату нуля. Операция Sum не включает значения null в результат для числовых типов, допускающих null.

Второй прототип Sum

Второй прототип Sum ведет себя так же, как и первый, но суммирует только те значения входной последовательности, которые выбраны делегатом метода selector.

C#

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

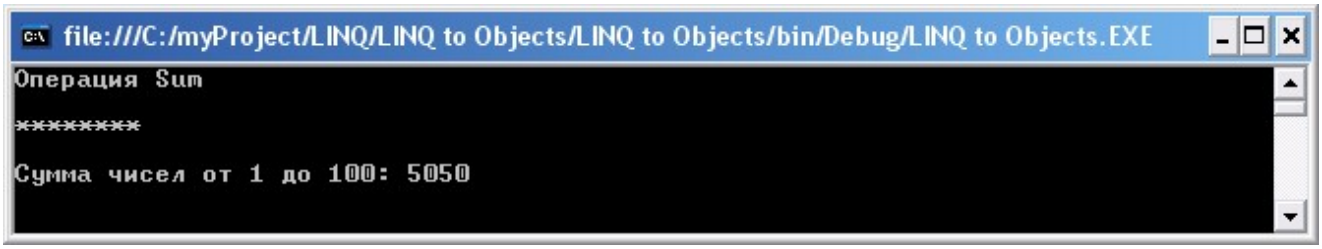
Если любой из аргументов равен null, генерируется исключение ArgumentNullException. Если сумма оказывается слишком большой, чтобы уместиться в тип Numeric, и если тип Numeric отличается от decimal или decimal?, генерируется исключение OverflowException. Если же типом Numeric является decimal или decimal?, возвращается положительная или отрицательная бесконечность.

Ниже показан пример использования первого прототипа операции Sum:

C#

```
IEnumerable<int> ints = Enumerable.Range(1, 100);

int sum = ints.Sum();
Console.WriteLine("Сумма чисел от 1 до 100: " + sum);
```



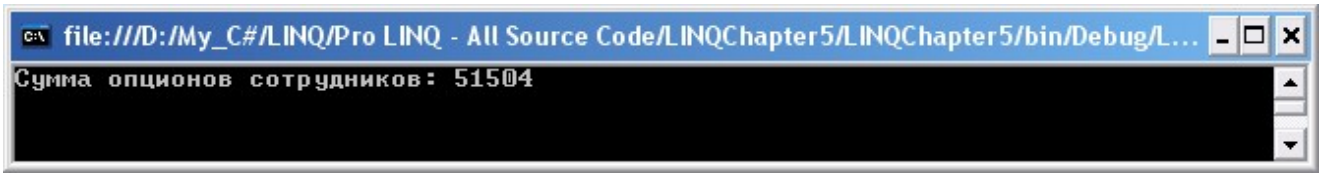
Теперь попробуем второй прототип, как показано ниже. В этом примере будет использован общий класс EmployeeOptionEntry (../level2/2_0.php), и будут суммироваться опционы для всех сотрудников:

C#

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

long optionsSum = options.Sum(o => o.optionsCount);
Console.WriteLine("Сумма опционов сотрудников: {0}", optionsSum);
```

Вместо того чтобы пытаться суммировать весь элемент, что не имело бы смысла в данном примере, поскольку речь идет об объекте сотрудника, можно использовать элемент selector второго прототипа, чтобы извлечь только интересующие члены — в данном случае optionsCount. Результат выполнения этого кода показан ниже:



Назад (3_7.php)	7	8	9	Вперед (3_9.php)

Пройди тесты

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

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