Операции Any, All и Contains

Any

Операция Any возвращает true, если любой из элементов входной последовательности отвечает условию. Эта операция имеет два прототипа, которые описаны ниже:

Первый прототип Any

C#

```
public static bool Any<T> (
       this IEnumerable<T> source);
```

Этот прототип операции Any вернет true, если входная последовательность source содержит любые элементы.

Второй прототип Any

Второй прототип операции Any перечисляет входную последовательность и возвращает true, если хотя бы для одного элемента из входной последовательности вызов делегата метода predicate возвращает true. Перечисление входной последовательности source прекращается, как только predicate вернет true:

C#

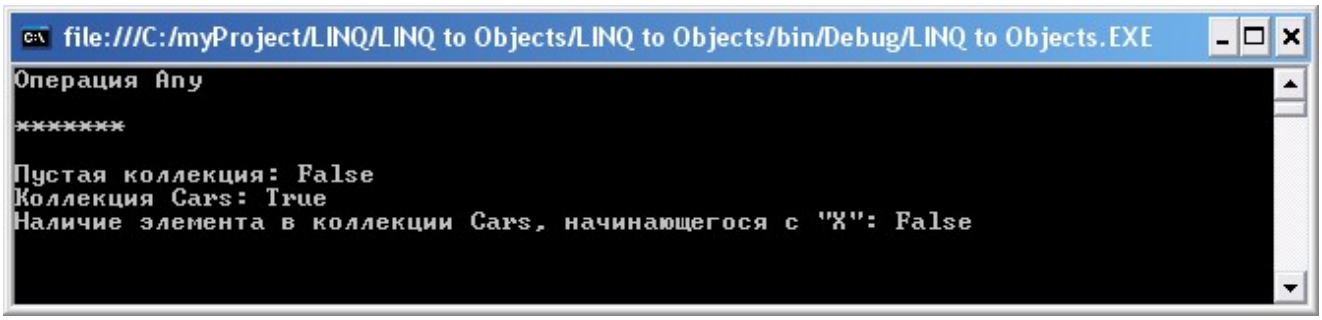
```
public static bool Any<T> (
       this IEnumerable<T> source,
       Func<T, bool> predicate);
```

Ниже показан пример использования обоих прототипов. Сначала воспроизводится пустая последовательность и проверяется с помощью операции Any. Затем проверяется массив Cars:

C#

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

bool anyNull = Enumerable.Empty<string>().Any();
Console.WriteLine("Операция Any \n\n*******\n\nПустая коллекция: " + anyNull);

bool anyCars = cars.Any();
Console.WriteLine("Коллекция Cars: " + anyCars);

// Используем второй прототип
anyCars = cars.Any(s => s.StartsWith("X"));
Console.WriteLine("Наличие элемента в коллекции Cars, начинающегося с \"X\": " + anyCars);
```

Вот результат запуска этого кода:



All

Операция All возвращает true, если каждый элемент входной последовательности отвечает условию. Операция All имеет один прототип, описанный ниже:

C#

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

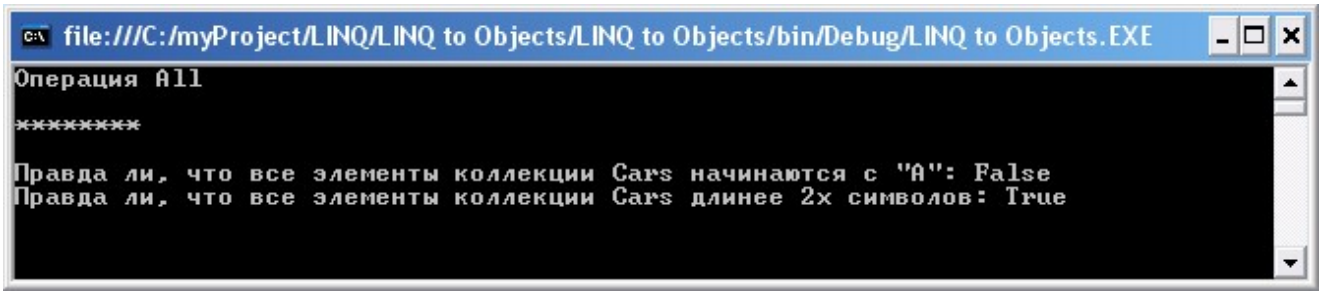
Операция All перечисляет входную последовательность source и возвращает true, если predicate возвращает true для каждого элемента последовательности. После возврата из predicate значения false перечисление прекращается.

Ниже показан пример использования операции All:

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 All\n\n********\n");

bool all = cars.All(s => s.StartsWith("A"));
Console.WriteLine("Правда ли, что все элементы коллекции Cars начинаются с \"A\": " + all);

all = cars.All(s => s.Length > 2);
Console.WriteLine("Правда ли, что все элементы коллекции Cars длинее 2х символов: " + all);
```

Поскольку известно, что не все названия машин в массиве начинаются с "A", первый вызов операции All в этом примере вернет False. При этом второй вызов вернет True, т.к. длина каждого элемента из массива Cars больше 2. Вот результат:



Contains

Операция Contains возвращает true, если любой элемент входной последовательности соответствует указанному значению. Эта операция имеет два прототипа, которые описаны ниже:

Первый прототип Contains

C#

```
public static bool Contains<T>(
    this IEnumerable<T> source,
    T value);
```

Пройди тесты

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

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

Этот прототип операции Contains сначала проверяет входную последовательность на предмет реализации ею интерфейса ICollection<T>, и если она его реализует, то вызывается метод Contains реализации последовательности. Если

последовательность не реализует интерфейс ICollection<T>, входная последовательность source перечисляется с проверкой соответствия каждого элемента указанному значению. Как только найден элемент, который ему соответствует, перечисление прекращается.

Указанное значение сравнивается с каждым элементом с использованием класса проверки эквивалентности по умолчанию EqualityComparer<K>.Default.

Второй прототип Contains

Второй прототип подобен первому, за исключением возможности указания объекта IEqualityComparer<T>. Если используется этот прототип, каждый элемент в последовательности сравнивается с переданным значением с применением переданного объекта проверки эквивалентности:

C#

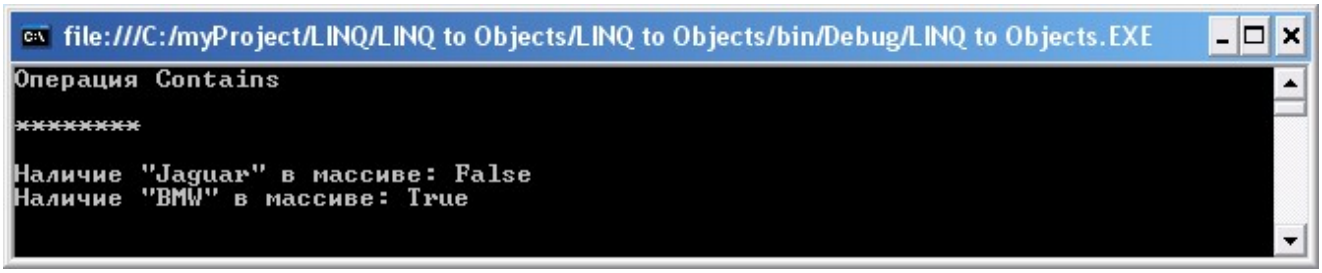
```
public static bool Contains<T>(
     this IEnumerable<T> source,
     T value,
     IEqualityCoinparer<T> comparer);
```

Для демонстрации работы первого прототипа, рассмотрим следующий пример:

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 Contains\n\n********\n");

bool contains = cars.Contains("Jaguar");
Console.WriteLine("Наличие \"Jaguar\" в массиве: " + contains);

contains = cars.Contains("BMW");
Console.WriteLine("Наличие \"BMW\" в массиве: " + contains);
```

В данном пример проверяется наличие слов "Jaguar" и "BMW" в исходном массиве Cars с помощью первого прототипа операции Cars. Результат выглядит следующим образом:



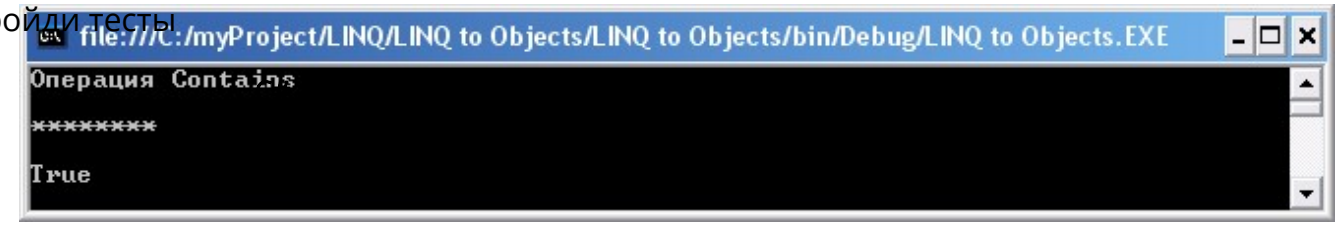
При демонстрации примера использования второго прототипа Contains задействован общий класс MyStringifiedNumberComparer. Массив чисел в строковом формате проверяется на предмет наличия числа, формально не эквивалентного ни одному элементу массива, но поскольку применяется собственный класс проверки эквивалентности, соответствующий элемент будет найден. Код приведен ниже:

C#

```
string[] nums = {"1", "5","10","28" };
bool contains = nums.Contains("000028",
                new MyStringifiedNumberComparer());
Console.WriteLine(contains);

...

public class MyStringifiedNumberComparer : IEqualityComparer<string>
  {
    public bool Equals(string x, string y)
    {
      return (Int32.Parse(x) == Int32.Parse(y));
    }

    public int GetHashCode(string obj)
    {
      return Int32.Parse(obj).ToString().GetHashCode();
    }
  }
```

Поскольку ищется элемент со значением "000028" и используется объект проверки эквивалентности, который перед сравнением преобразует это строковое значение наряду со всеми элементами последовательности в целое число, и т.к. последовательность содержит элемент "28", переменная contains должна быть равна true. Взглянем на результат:

Пройди тесты



Назад (3_6.php)	6	7	8	Вперед (3_8.php)

Пройди тесты

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

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