МЕТОДЫ РАСШИРЕНИЯ

В .NET, начиная с версии 3.0, появился механизм, позволяющий использовать методы расширения. Методы расширения позволяют "добавлять" методы в существующие типы без создания нового производного типа, перекомпиляции или иного изменения исходного типа. Методы расширения являются особым видом статического метода, но они вызываются так, как если бы они были методами экземпляра. Для кода, написанного на языке C#, нет видимого различия между вызовом метода расширения и вызовом методов, фактически определенных в типе.

Наиболее распространенными методами расширения являются стандартные операторы запроса LINQ, добавляющие функции запроса в существующие типы, реализующие интерфейсы System.Collections.IEnumerable и System.Collections.Generic.IEnumerable<T>.

Рассмотрим некоторые методы расширения более подробно.

Метод расширения Where

Метод расширения Where действует аналогично предложению where и позволят проводить фильтрацию данных некоторого источника.

Рассмотрим пример, с которого мы начали знакомство с интегрированными запросами, но реализуем его с помощью метода расширения Where.

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         //источник данных - массив
         int[] number = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
         //обращаемся к методу расширения Where
         var lowNums = number.Where(n => n <= 5);
         //обрабатываем результат
         foreach (var x in lowNums)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
1 2 3 4 5 0
```

В качестве источника данных был выбран одномерный массив. При обращении к методу расширения мы указали массив, а затем обратились к методу Where также, как к и любому экземплярному методу. Единственное исключение – в качестве аргумента методу расширения мы предали лямбда-выражение: n => n <= 5.

Лямбда-выражение – это анонимная функция, которая содержит выражения и операторы. Во всех лямбда-выражениях используется лямбда-оператор =>, который читается как "переходит

в". Левая часть лямбда-оператора определяет параметры лямбда-выражения, а правая часть содержит выражение, или блок оператора.

Наше лямбда-выражение читается как "n переходит в n меньше или равное 5". Это выражение, передаваясь методу Where в качестве аргумента, позволят реализовать запрос вида:

```
from n in number
where n <= 5
select n;
```

Задание

Возьмите в качестве источника данных обобщенный список List<> и примените к нему тот же метод расширения с заданным лямбда-выражением. Объясните полученный результат.

Оператор, стоящий в лямбда-выражении справа от оператора => может быть и более сложным, например:

1. ***n <= 5 || n>=8*** говорит о том, что n определяется как значение «не больше 5, или не меньше 8».
2. ***n>5 && n<8*** говорит о том, что n определяется как значение «больше 5 и меньше 8».

Рассмотрим следующий пример:

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace Example
{
   class Student
   {
      public string Name;
      public int ID;
      public Student(string name, int id)
      {
         Name = name;
         ID = id;
      }
   }
   class Program
   {
      static void Main()
      {
         List<Student> list = new List<Student>{
                 new Student("Иванов", 1),
                 new Student("Петров", 8),
                 new Student("Сидоров", 6),
                 new Student("Ткачев", 3),
                 new Student("Смирнов", 9),
                 new Student("Цукерман", 2)};
         var students =
                 list.Where(n => n.Name[0] == 'С' && n.ID > 6);
```

```
            foreach (var x in students)
            {
               Console.WriteLine("{0} {1}", x.Name, x.ID);
            }
         }
      }
   }
```

Результат работы программы:

```
Смирнов 9
```

В данном случае, метод расширения применяется к обобщенному списку, который используется для хранения ссылок на экземпляры класса Student, а лямбда-выражение метода Where позволяет обращаться к различным полям данного класса.

Задание

Измените лямбда-выражение так, чтобы метод Where позволил отобрать студентов, ID которых четно.

Лямбда-выражение может содержать несколько параметров. Следующий пример позволит нам вывести на экран имена чисел от 0 до 9, у которых длина имени меньше, чем его порядковый номер в массиве.

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         string[] digits = {"ноль","один", "два",
                            "три", "четыре", "пять",
                            "шесть", "семь", "восемь","девять"};
         var shortDigits = digits.Where((n, i) => n.Length < i);
         foreach (var x in shortDigits)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
пять шесть семь восемь девять
```

Следующий пример позволят получить информацию о файлах диска c, размер которых больше 10Кбайт:

```
using System;
using System.Linq;
using System.IO;
namespace Example
{
```

```
        class Program
        {
           static void Main()
           {
              DirectoryInfo directory =new DirectoryInfo( @"c:\");
              var files = directory.GetFiles().Where(n=>n.Length>10240);
              foreach (var x in files)
              {
                 Console.WriteLine("{0} {1}", x.Name, x.Length);
              }
           }
        }
     }
```

Результат работы программы:

```
NTDETECT.COM   47564
ntldr          251152
pagefiles.sys  2145386496
```

Задание

Измените лямбда-выражение так, чтобы метод Where позволил получить информацию о файлах, созданных сегодня.

Метод расширения Select

Данный метод может использоваться для изменения значений в заданном источнике данных. Например, следующая программа показывает, как создать массив, элементы которого на 1 превосходят элементы исходного массива.

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         int[] number = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
         var nums = number.Select(n => n+1);
         foreach (var x in nums)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
2 3 4 5 6 7 8 9 1
```

Допускается комбинированное использование расширяющих методов. Следующий пример выводит на экран только четные элементы массива, увеличенные на 1:

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         int[] number = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
         var nums = number.Where(n=>n%2==0).Select(n => n+1);
         foreach (var x in nums)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
3 5 7 9 1
```

Задание

Объясните, изменится ли в результате выполнения данной программы массив number, и почему.

Замечание

Не стоит использовать слишком большое число комбинаций методов расширения в одном запросе, так как это делает текст программы трудночитаемым.

Методы расширения OrderBy и OrderByDescending

Метод расширения OrderBy, также как и предложение orderby в интегрированном запросе, позволяет отсортировать данные, извлеченные из источника. Так следующий пример позволит нам отсортировать файлы по имени:

```
using System;
using System.Linq;
using System.IO;
namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo directory = new DirectoryInfo(@"c:\");
         var files = directory.GetFiles().OrderBy(n => n.Name);
         foreach (var x in files)
         {
            Console.WriteLine("{0} {1}", x.Name);
         }
      }
   }
}
```

Результат работы программы:

```
letter.doc
pucture.bmp
отчет.doc
реферат.doc
список.txt
```

Как видим, метод расширения OrderBy сортирует файлы в алфавитном порядке, по возрастанию. Для того, чтобы сортировка данных производилась по убыванию значения некоторого ключа можно использовать метод OrderByDescending. Например:

```
using System;
using System.Linq;
using System.IO;

namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo directory = new DirectoryInfo(@"c:\");
         var files =
               directory.GetFiles().OrderByDescending (n=>n.Name);
         foreach (var x in files)
         {
            Console.WriteLine("{0} {1}", x.Name);
         }
      }
   }
}
```

Результат работы программы:

```
список.txt
реферат.doc
отчет.doc
pucture.bmp
letter.doc
```

Допускается комбинированное использование методов расширения. Так следующий пример позволит нам вывести на экран информацию о файлах, отсортированных по расширению, размер которых превышает 20 Кбайт.

```
using System;
using System.Linq;
using System.IO;

namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo directory = new DirectoryInfo(@"d:\");
```

```
            var files = directory.GetFiles()
                           .Where(n => n.Length > 20480)
                           .OrderBy(n => n.Extension);
            foreach (var x in files)
            {
               Console.WriteLine("{0} {1}", x.Name, x.Length);
            }
         }
      }
   }
```

Результат работы программы:

```
pucture.bmp
отчет.doc
реферат.doc
```

Задание

Используя методы расширения, выведите на экран информацию о файлах с расширением txt, упорядоченную в алфавитном порядке по имени файла.

Методы расширения Take и TakeWhile

Метод Take возвращает заданное количество элементов источника данных, начиная с нулевого. Метод TakeWhile выполняет почти то же самое, что и метод Take, но выбор заканчивается не после фиксированного числа элементов, а тогда, когда перестает выполняться условие, определенное в лямбда-выражении. Например:

```
using System;
using System.Linq;
using System.IO;
namespace Example
{
   class Program
   {
      static void Main()
      {
         int[] numbers = { 1,3,5,7,9,8,14,15};
         var y = numbers.Take(3);
         Console.Write("y: ");
         foreach (var i in y)
         {
            Console.Write("{0} ", i);
         }
         Console.WriteLine();
         Console.Write("z: ");
         var z = numbers.TakeWhile(n => n %2 == 1);
         foreach (var i in z)
         {
            Console.Write("{0} ", i);
         }
      }
   }
}
```

Результат работы программы:

```
y: 1 3 5
z: 1 3 5 7 9
```

Задание

Измените лямбда-выражение в методе TakeWhile так, чтобы на экран выводились элементы массива до тех пор, пока они являются цифрами.

Следующий пример позволит нам получить информацию о трех файлах, объем которых наименьший:

```
using System;
using System.Linq;
using System.IO;
namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo directory = new DirectoryInfo(@"d:\");
         var files = directory.GetFiles()
                              .OrderBy(n => n.Length)
                              .Take(3);
         foreach (var x in files)
         {
            Console.WriteLine("{0} {1}", x.Name, x.Length);
         }
      }
   }
}
```

Результат работы программы:

```
letter.doc
список.txt
отчет.doc
```

Задание

Используя методы расширения, выведите на экран наименьший по объему файл, измененный сегодня.

Методы расширения Skip и SkipWhile

Метод Skip, в отличие от метода Take, возвращает все элементы источника данных, пропуская заданное количество начальных элементов. Метод SkipWhile возвращает все элементы источника данных, начиная с того, для которого условие в лямбда-выражении перестало существовать. Например:

```
using System;
using System.Linq;
using System.IO;
namespace Example
{
```

```
   class Program
   {
      static void Main()
      {
         int[] numbers = { 1, 3, 5, 7, 9, 8, 14, 15 };
         var y = numbers.Skip(3);
         Console.Write("y: ");
         foreach (var i in y)
         {
            Console.Write("{0} ", i);
         }
         Console.WriteLine();
         Console.Write("z: ");
         var z = numbers.SkipWhile(n => n % 2 == 1);
         foreach (var i in z)
         {
            Console.Write("{0} ", i);
         }
      }
   }
}
```

Результат работы программы:

```
y: 7 9 8 14 15
z: 8 14 15
```

Следующий пример позволит нам получить информацию обо всех файлах, кроме первых трех с наименьшем размером.

```
using System;
using System.Linq;
using System.IO;

namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo directory = new DirectoryInfo(@"d:\");
         var files = directory.GetFiles()
                              .OrderBy(n => n.Length)
                              .Skip(3);
         foreach (var x in files)
         {
            Console.WriteLine("{0} {1}", x.Name, x.Length);
         }
      }
   }
}
```

Результат работы программы:

```
реферат.doc
pucture.bmp
```

Задание

Выведите на экран информацию о файлах, за исключением тех, которые были созданы ранее заданной даты.

Самостоятельная работа №7

1. Самостоятельно изучите следующие методы расширения: Count, First, GroupBy, GroupJoin, Join, Last, Max, Min, Revers, Sum. Продемонстрируйте на примерах работу с данными методами, используя в качестве источников данных различные структуры данных.
2. Используя дополнительную литературу, изучите возможность разработки собственных методов расширения.