ЛЕКЦИЯ №8 LINQ ЗАПРОСЫ

Москва, 2020

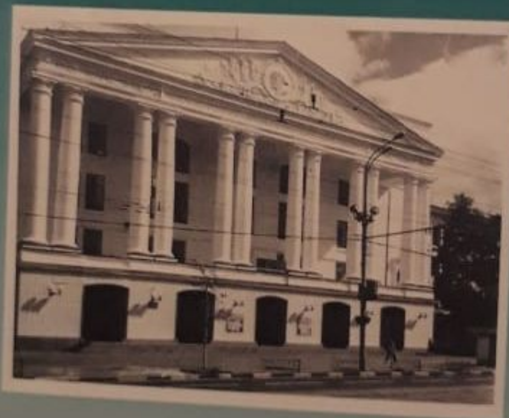
Задача на автомат

1. Служба под Windows
2. Должен быть лог файл.
3. Служба работает с базой данных.
4. Многонитевой программирование. Очередь запросов.
5. Разработка через тестирование.
6. Абстрактные класс и интрефейсы.

LINQ - технология

УДК 004.438
ББК 32.973-018.1
Б 833

МЭИ



Г.А. Бородин, И.Н. Андреева

ОСНОВЫ LINQ

LINQ - технология

Расширение возможностей запросов к данным в синтаксисе языка C#

Источник данных должен поддерживать интерфейс IEnumerable.
Запросы обычно выражаются на специализированном языке запросов.
Для различных типов источников данных, например, SQL для реляционные базы данных и XQuery для XML.
Новый язык запросов для каждого типа данных.
LINQ упрощает эту ситуацию, предлагая
согласованную модель для работы с данными в различных виды источников данных и форматов.
В запросе LINQ вы всегда работаете с объектами.
Поставщик LINQ доступен

LINQ - технология

В выражении Linq запроса указывается источник данных,
фильтрация,
группировка,
сортировка данных

Классификация LINQ

LINQ to Object

LINQ to Object обеспечивает поддержку взаимодействия с объектами .NET в памяти, которые реализуют интерфейс IEnumerable <T>

LINQ to Entities

Он более гибкий, чем LINQ to SQL

LINQ to SQL – исключительно к MSSQL

LINQ - технология

Последовательность – любой объект, который реализует IEnumerable, а элемент – элемент этой последовательности.

Элементами могут быть атомарными, коллекции, фрагменты XML документов.
Запрос не изменяет исходную последовательность, а возвращает новую.

LINQ to Dataset обеспечивает поддержку взаимодействия с кэшем данных в памяти

LINQ to SQL, также известный как DLINQ, обеспечивает поддержку взаимодействия с базой данных MSSQL

LINQ - технология

LINQ to XML, также известный как XLINQ, обеспечивает поддержку взаимодействия с документами XML, т. Е. Для загрузки XML-документы, а также выполнять запросы, такие как чтение, фильтрация, изменение, добавление узла

Операторы LINQ на самом деле представляют собой набор методов расширения. Эти операторы образуют шаблон LINQ. Эти операторы предлагают гибкость для запроса данных, таких как фильтрация данных, сортировка и т. д.

Следующие операторы запроса LINQ мы обсудим:

1. Оператор фильтрации Where
2. Оператор проекции Select
3. Присоединение к оператору JOIN

LINQ - технология

4. Группировка оператора
5. Оператор раздела
6. Агрегация

Присоединение к оператору

Оператор соединения используется для объединения двух или более последовательностей или коллекций на основе некоторого ключа и создания результата

join..in..on.equals

Оператор группировки используется для организации элементов на основе заданного ключа. GroupBy Вернуть последовательность элементов в группах IGroup<key,element>

group…..by <or> group…by..into

<or>

GroupBy(<predicate>)

LINQ - технология

Оператор разделения используется для разделения коллекции или последовательности на две части и возврата оставшейся части (запись), оставленная следствием этих операторов разбиения

Skip Пропустить предоставленное количество записей и вернуться остальные.

Skip<T> (<число>)

Take Получить предоставленное количество записей и пропустите остальные.

Take<T>(<count>)

Агрегация означает применение агрегатных функций в LINQ. Агрегатная функция - это функция, которая вычисляет запрос и возвращает одно значение.

LINQ - технология

Average	Take the average of a numeric collection.	Average<T>(<param>)
Count	Count the number of elements in a collection.	Count<T>(<param>)
Max	Return the highest value from the collection of numeric values.	Max<T>(<param>)
Min	Return the highest value from the collection of numeric values.	Min<T>(<param>)
Sum	Compute the sum of numeric values in a collection.	Sum<T>(<param>)

LINQ предоставляет различные способы взаимодействия с источниками данных для их запроса. Это облегчает разработчикам SQL
взаимодействовать с различными источниками данных для запроса с использованием C #, предоставляя им синтаксис LINQ Query.

1. Синтаксис метода LINQ..
2. Запрос LINQ.

Анонимные типы

В объектно-ориентированных языках (таких как C #) обычно определяют небольшие классы, которые будут использоваться только один раз. Типичным примером является класс Point, который имеет только два поля - координаты точки. Создание простого класса с идеей использовать его только один раз неудобно и отнимает много времени для программиста, особенно когда стандартные операции для каждого класса: ToString (), Equals () и GetHashCode () должны быть предопределены. В C # есть встроенный способ создания одноразовых типов, называемых анонимными типами. Объекты такого типа создаются почти так же, как и другие объекты в C #. Дело в том, что нам не нужно заранее определять тип данных для переменной. Ключевое слово var указывает компилятору, что тип переменной будет автоматически определяться выражением после знака равенства. На самом деле у нас нет выбора, так как мы не можем указать конкретный тип переменной, потому что она определена как одна из анонимных типов. После этого мы указываем имя для объекта, затем оператор "=" и ключевое слово new. В фигурных скобках мы перечисляем имена и значения свойств анонимного типа

Анонимные типы

Anonymous Types – Example

Here is an example of creating an anonymous type that describes a car:

```
var myCar = new { Color = "Red", Brand = "BMW", Speed = 180 };
Console.WriteLine("My car is a {0} {1}.",
myCar.Color, myCar.Brand);
Console.WriteLine("It runs {0} km/h.", myCar.Speed);
```

Массивы анонимных типов

Анонимные типы, как и обычные, могут использоваться в качестве элементов массивов. Мы можем инициализировать их ключевым словом new, за которым следуют квадратные скобки. Значения элементов массива перечислены так же, как

```
var arr = new[] {
new { X = 3, Y = 5 },
new { X = 1, Y = 2 },
new { X = 0, Y = 7 }
};
foreach (var item in arr)
{
Console.WriteLine(item.ToString());
}
```

Анонимные типы

```
List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6 };
List<int> evenNumbers = list.FindAll(x => (x % 2) == 0);
foreach (var num in evenNumbers)
{
Console.Write("{0} ", num);
}
Console.WriteLine();
The result is:
2 4}
```

Лямбда выражения

Под лямбда-выражением понимается анонимная функ
рую можно использовать для создания таких типов как дел
деревья выражений. С их помощью можно написать
функции, которые вслед за тем можно передавать в другие
качестве аргументов или возвращать из них в качестве знач

Лямбда-выражение определяется как разделённый запя
сок параметров, за которым следует лямбда-операция, а за
ражение или блок операторов. В C# лямбда-операция зап
как =>. Если параметров более одного, то они заключаютс
а если он один – скобки обычно опускаются.

Синтаксически обобщённо лямбда-выражение на C# выгл

```
(<параметр1>[, <параметр2>]) => <выражение>
```

Когда компилятору бывает трудно или невозможно
тип выходных параметров, их можно указать явно:

```
(int x, string s) => s.Length > x
```

Или, когда требуется бо́льшая сложность, может п
блок операторов, заключаемый в фигурные скобки:

```
(<параметр1>[, <параметр2>]) => {
    <оператор1>;
    <оператор2>;
    [<оператор3>;]
return(<тип_возврата_лямбда-выражения>); }
```

Например:

```
(s, i) => { i = 3; return s.Length > i; }
s => { int i = 3; bool b = (s.Length > i); return b; }
(string s) => { int i = 3; bool b = (s.Length > i); retur
```

LINQ - технология

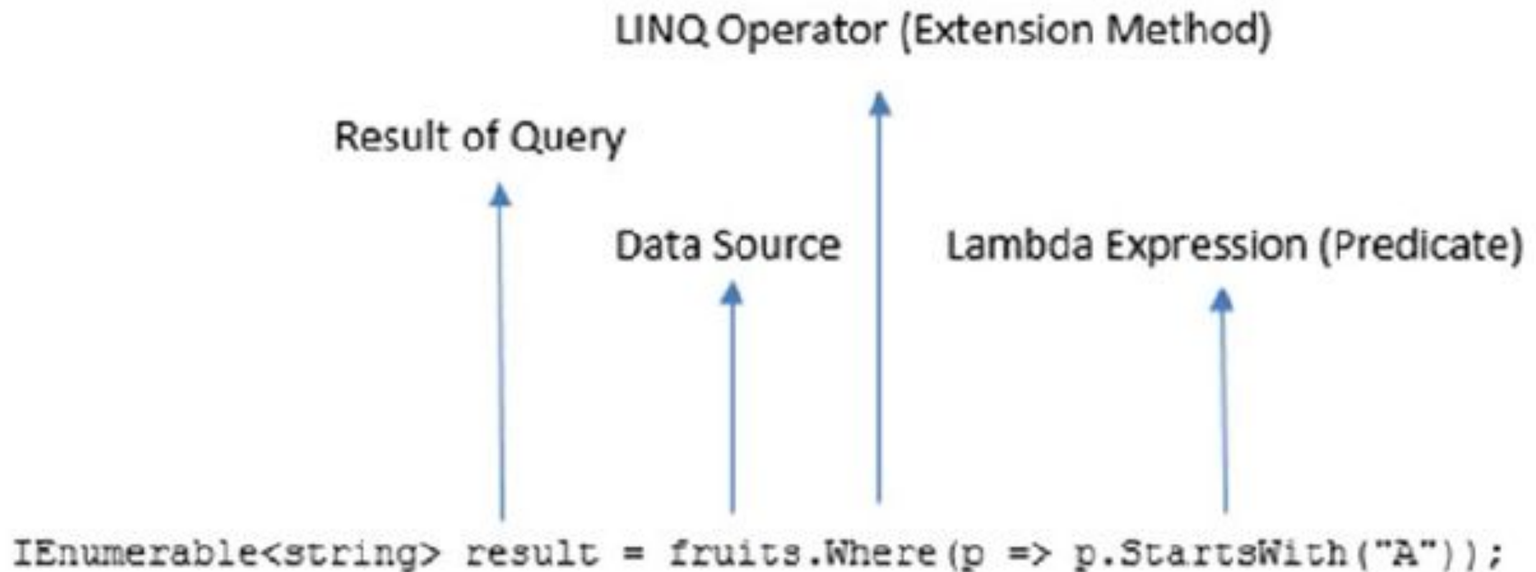
LINQ предоставляет синтаксис метода для взаимодействия с различными источниками данных для их запроса. В основном, он использует методы расширения для запроса данных.

```
result=DataSource.Operator(<lambda expression>);

OR

result=DataSource.Operator(<lambda expression>).Operator(<optional>);
```

где результат должен иметь тип возвращаемых данных. Вы также можете использовать тип var, если вы не уверены в возвращаемом типе данных.

LINQ - технология

Получить фрукты, название которых начинается с «А». Поэтому мы делаем запрос на фрукты (источник данных) чтобы получить требуемый результат.



LINQ - технология

также можем дополнительно применить другой оператор (метод расширения) по тому же запросу, чтобы подсчитать количество фруктов

```
int fruitsLength = fruits.Where(p => p.StartsWith("A")).Count();
```

LINQ - технология

2.2. Обобщённый синтаксис оператора запроса

Обобщённый синтаксис оператора запросов имеет следующий вид.

• Запрос с декларативным синтаксисом:

from <переменная диапазона> **in** <источник данных>

[**where** <предикат>]

[**orderby** <выражение> [**ascending**|**descending**]

[**let** <идентификатор> = <выражение>|(<запрос>)]

select|[**group** <переменная диапазона> **by** <выражение>]

[**into** <временный идентификатор> <запрос>]

Между первым разделом from и последним разделом select или group может содержаться один или несколько необязательных разделов where, orderby, join, let или, даже, дополнительных разделов from.

LINQ - технология

2.3.1. Раздел from

Выражение LINQ запроса с декларативным синтаксисом должно начинаться с раздела from и оканчиваться разделом select или group.

Идентификатор, следующий сразу за ключевым словом from, называют переменной диапазона (итерации) [1].

```
var query = from p in dc.Publishers
            select p;
```

query – переменная запроса;

p – переменная диапазона;

dc.Publishers – источник данных (таблица из базы данных Pubs).

Переменная диапазона автоматически строго типизируется на основе типа данных элементов источника данных.

Выражение LINQ запроса может содержать несколько разделов from, причём дополнительные разделы from часто используют для доступа к внутренним коллекциям в едином источнике данных и, по сути, используют как подзапросы. Подзапрос является всего лишь ещё одним последовательным оператором на языке C# и не требует специального синтаксиса.

Раздел from для LINQ запроса на основе методов фактически со-
... <источника данных> – dc.Publisher.

LINQ - технология

2.3.2. Раздел select

Раздел select задаёт тип значений, получаемых при выполнении запроса, и форму его представления.

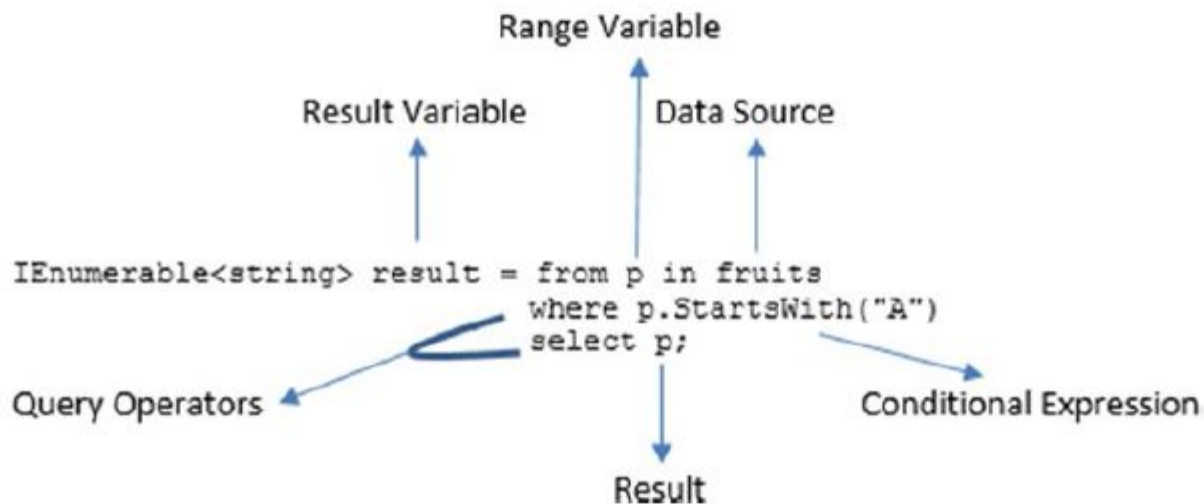
Часто раздел select просто создаёт последовательность с тем же типом объектов, что и у объектов, содержащихся в источнике данных, а выбор свойств объектов производится в процессе итерации. При использовании запроса на основе методов раздел select можно в этом случае опустить, а при декларативном он должен быть записан.

LINQ - технология

1. Дан массив вывести из него все различные слова имеющие 6 букв в верхнем регистре и в алфавитном порядке. Решить на основе LINQ запросов.
2. Дан массив целых чисел. Выбрать числа большие чем 4 и запистаь из в список Y отсортировав по возрастанию. Решить на основе LINQ запросов.
3. Дан список имен из списка выбрать все имена, начинающиеся на некоторую букву. Решить на основе LINQ запросов.
4. Дан список сотрудников предприятия. Вывести всех некурящих сотрудников для повышения им зарплаты.
5. Написать LINQ запрос для вывода всех студентов, получивших двойки.

   Создать класс студент и поле «Оценки» типа список.

LINQ - технология



При работе с запросами LINQ у него всегда есть три шага или действия:

1. Получить источник данных
2. Создайте запрос
3. Выполнить запрос

Немедленное выполнение запроса

Немедленное выполнение запроса - это выполнение во время написания запроса. Это заставляет запрос LINQ. выполнить и немедленно возвращает результаты. Выполняя методы ToList <T> или ToArray <T> для запроса, вы можете принудительно выполнить его немедленно.

Сколько ноуnбуков имеющий ч/б дисплей и сколько цветных.

```
SELECT color, Count(*)
  FROM laptop
GROUP BY color
ч.б   10
Цв    20
Id, hd, ram, color
1    10   20 ч.б
2    20   10 ч.б
3    30    20  Цв
```

LINQ - технология

Запрос SelectMany включает в себя различные свойства, которые не определены ни в одном классе, и может получить результат запроса путем доступа к этим свойствам анонимного типа. Этот тип запроса называется анонимным

Listing 6-8. SelectMany Operator

```
var result = (from p in persons
              where p.Name.Length > 4
              select new
              {
                  PersonID = p.ID,
                  PersonName = p.Name,
                       PersonAddress=p.Address
              });
```

```
foreach (var item in result)
{
    Console.WriteLine(item.PersonID + "\t" + item.PersonName );
}
```

Группирование

Оператор группировки используется для организации последовательности элементов в группах как IGroup <ключ, элемент>.

Разделение

Оператор разделения используется для разделения коллекции или последовательности на две части и возврата оставшейся оставленный следствием одного из этих операторов разбиения. Он содержит операторы Take и Skip.

```
var sortedDogs = dogs.OrderByDescending(x => x.Age);
```

Агрегация

Агрегатная функция используется для вычисления запроса и возврата одного значения. .

Using Lambda Expressions with Anonymous Types

We can create **collections of anonymous types** from a collection with some elements by **using lambda expressions**. Let's take the collection **dogs**, containing elements of type **Dog**, and create new collection consisting of elements of an anonymous type, having two properties – age and the initial letter of the dog's name:

```
var newDogsList = dogs.Select(
   x => new { Age = x.Age, FirstLetter = x.Name[0] });
foreach (var item in newDogsList)
{
   Console.WriteLine(item);
}
```

The result is:

```
{ Age = 4, FirstLetter = R }
{ Age = 0, FirstLetter = S }
{ Age = 3, FirstLetter = S }
```

Statements in Lambda Expressions

Lambda functions can also have a **body**. So far we have used lambda functions with only one statement. Now we will pay more attention to lambda functions that have a body. Let's return to the example with the even numbers. Suppose we want to print to the console the values of all numbers, to which our lambda function is applied to and to return the result if they are even or not. We can do it the following way:

```
List<int> list = new List<int>() { 20, 1, 4, 8, 9, 44 };
// Process each argument with code statements
var evenNumbers = list.FindAll((i) =>
{
  Console.WriteLine("Value of i is: {0}", i);
  return (i % 2) == 0;
});
```

The result from the above code is:

```
Value of i is: 20
Value of i is: 1
Value of i is: 4
Value of i is: 8
Value of i is: 9
Value of i is: 44
```

Results of LINQ Queries

To **choose the output data** for the query, we can **use the keyword `select`**. The result is an object of an existing class or an anonymous type. The result can also be a property of the objects, the query runs through or the objects themselves. The **`select`** statement and everything following it is placed **always at the end of the query**. The four keywords: **`from`**, **`in`**, **`where`** and **`select`**, are completely enough to create a simple LINQ query. Here is an example:

```
List<int> numbers = new List<int>() {
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10
};
var evenNumbers =
  from num in numbers
  where num % 2 == 0
  select num;
foreach (var item in evenNumbers)
{
  Console.Write(item + " ");
}
```

The result is:

```
2 4 6 8 10
```

Sorting Data with LINQ

Sorting with LINQ queries is done through the **keyword orderby**. The conditions, used for sorting the elements, are placed after it. For each condition the order of arrangement can be indicated: ascending (using the keyword **ascending**) and descending (with the keyword **descending**), as by default the elements are ordered in ascending order. If we want to sort an array of strings by their length in descending order, for example, we can write the following query:

```
string[] words = { "cherry", "apple", "blueberry" };
var wordsSortedByLength =
   from word in words
   orderby word.Length descending
   select word;
foreach (var word in wordsSortedByLength)
{
   Console.WriteLine(word);
}
```

The result is:

```
blueberry
cherry
apple
```

LINQ

Для группировки результатов по некоторым критериям следует использовать группу ключевых слов. Шаблон выглядит следующим образом:
group [variable name] by [grouping condition] into [group name]

Результатом группировки является новая коллекция специального типа, которую можно использовать в дальнейшем в запросе. Однако после группировки запрос перестает работать с его начальной переменной. Это означает, что в операторе select мы можем использовать только группу. Пример группировки:

LINQ

```
int[] numbers =
  { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0, 10, 11, 12, 13 };
int divisor = 5;

var numberGroups =
  from number in numbers
  group number by number % divisor into group
  select new { Remainder = group.Key, Numbers = group };

foreach (var group in numberGroups)
{
  Console.WriteLine(
  "Numbers with a remainder of {0} when divided by {1}:",
    group.Remainder, divisor);
  foreach (var number in group.Numbers)
  {
    Console.WriteLine(number);
  }
}
```

The result is:

```
Numbers with a remainder of 0 when divided by 5:
5
0
10
Numbers with a remainder of 4 when divided by 5:
```

LINQ

В запросе для каждого числа вычисляется делитель числа%, а для каждого другого результата создается новый.
Далее в запросе оператор select работает со списком созданных групп, и для каждой группы создает анонимный тип с двумя свойствами: Remainder и Numbers. Свойству Remainder присваивается ключ группы (в нашем случае остаток от деления на делитель числа). А свойству Numbers присваивается группа сбора, которая содержит все элементы в группе. Обратите внимание, что выбор выполняется только по списку групп. Номер переменной не может быть использован там. Далее в примере двух вложенных операторов foreach печатаются остатки (группы) и числа, у которых есть остаток (расположенный в группе).

```
List<int> list = new List<int>();
list.AddRange(Enumerable.Range(1, 100000));
var elements = list.Where(e => e > 20000);
var element = list.Where(e => e > 20000).First();
```

LINQ, HashSet

```
HashSet<Guid> set = new HashSet<Guid>();
for (int i = 0; i < 50000; i++)
{
set.Add(Guid.NewGuid()); // Add random GUID
}

Guid keyForSearching = new Guid();
for (int i = 0; i < 50000; i++)
{
// Use HashSet.Contains(…)
bool found = set.Contains(keyForSearching);
}
bool found = set.Where(g => g==keyForSearching).Count() > 0;
```

LINQ, упражнения

Напишите класс student со следующими свойствами: имя, фамилия и возраст. Напишите метод, который для данного массива student находит тех, чье имя совпадает заданному в массиве. Используйте LINQ