Лекция № 4

I T E A

IT Education Academy

WWW.ITEA.UA

C# Base

Урок № 4

LINQ

План урока

- Анонимные типы
- LINQ
- Стандартные операции запросов. Фильтрация. Сортировка.
- Группировка
- Выборка сложных объектов
- Динамические типы

Анонимные типы

Анонимные типы позволяют создать объект с некоторым набором свойств без определения класса.

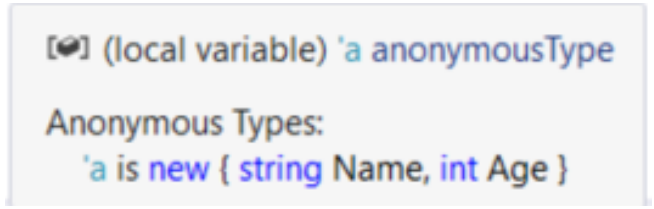
Анонимный тип определяется с помощью ключевого слова var и инициализатора объектов.

Имя типа создается компилятором и не доступно на уровне исходного кода.

Анонимный тип объявляется с помощью следующей общей формы:

new { имя_A = значение_A, имя_B = значение_B, ... }

```
var anonymousType = new { Name = "Alexs", Age = 30 };
```



anonymousType - это объект анонимного типа, у которого определены два свойства Name и Age. Ети свойства можно использовать как обычные свойства класса.

```
string name = anonymousType.Name;
int age = anonymousType.Age;
```

Анонимные типы

Зачем нужны анонимные типы?

Иногда возникает задача использовать один тип в одном узком контексте или даже один раз. Создание класса для подобного типа может быть избыточным. Если нам захочется добавить свойство, то мы сразу же на месте анонимного объекта это можем сделать. В случае с классом придется изменять еще и класс, который может больше нигде не использоваться.

```
public class Person
{
    public string Name { get; set; }

    public int Age { get; set; }
}
```

```
Person person = new Person() { Name = "Alexs", Age = 30 };
```

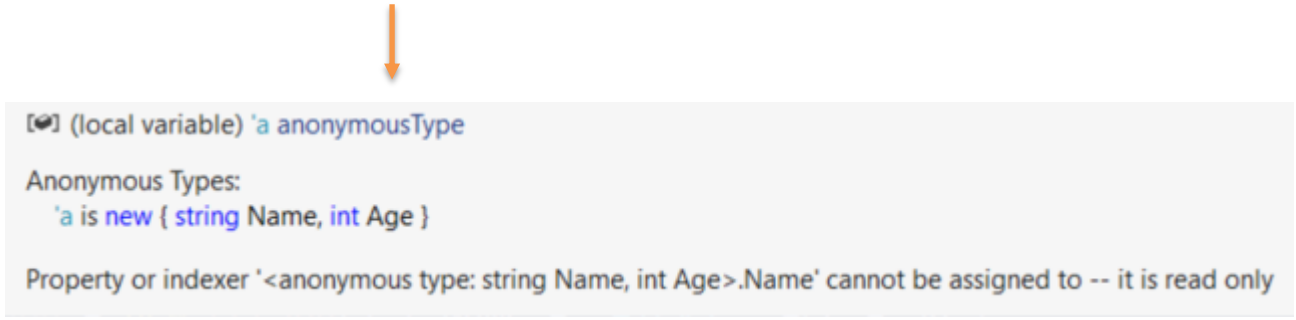
аналогично

```
var anonymousType = new { Name = "Alexs", Age = 30 };
```

Анонимные типы

Свойства анонимного объекта доступны для установки только в инициализаторе.

```
var anonymousType = new { Name = "Alexs", Age = 30 };
anonymousType.Name = "Ivan";
```



- Для исполняющей среды CLR анонимные типы будут также, как и классы, представлять ссылочный тип.
- Объекты анонимного типа нельзя преобразовать к какому-нибудь другому типу, например, классу, даже если он имеет подобный набор свойств.

Анонимные типы

В анонимных типах могут быть вложенные анонимные типы

```
var anonymousNestedType = new
{
    Name = "Alexs",
    Age = 30,
    Salary = new { Advance = 2500 }
};
```

Анонимный тип Salary вложенный в анонимный тип anonymousNestedType

Доступ к свойствам:

```
int advance = anonymousNestedType.Salary.Advance;
string name = anonymousNestedType.Name;
```

Типичная ситуация использования анонимного типа - получение результата выборки из базы данных: объекты используются только для получения выборки, часто больше нигде не используются, и классы для них создавать было бы излишне

LINQ

LINQ (Language-Integrated Query) представляет простой и удобный язык запросов к источнику данных, проект Microsoft по добавлению синтаксиса языка запросов, напоминающего SQL, в языки программирования платформы .NET Framework.

Все операции запроса LINQ состоят из трех различных действий.

- Получение источника данных.
- Создание запроса.
- Выполнение запроса

LINQ – представляет стандартные, легко изучаемые шаблоны для создания запросов и обновления данных. Технология может быть расширена для поддержки потенциально любого типа хранилища данных.

LINQ

LINQ to Objects: применяется для работы с массивами и коллекциями

LINQ to Entities: используется при обращении к базам данных через технологию Entity Framework

LINQ to Sql: технология доступа к данным в MS SQL Server

LINQ to XML: применяется при работе с файлами XML

LINQ to DataSet: применяется при работе с объектом DataSet

Parallel LINQ (PLINQ): используется для выполнения параллельных запросов

Стандартные операции запросов

Выражение запроса должно начинаться предложением **from** и оканчиваться предложением **select** или **group**.

- **From** – задает источник данных.
- **Group** – используется для получения последовательности групп, организованной на основе указанного ключа.
- **Select** – для получения всех других типов последовательностей.
- **Into** – можно использовать в предложении select или group для создания временного идентификатора, в котором хранится запрос.
- **Orderby** – сортирует результаты в порядке возрастания или убывания.
- **Where** – используется для фильтрации элементов из источника данных по одному или нескольким выражениям предиката.

Простейший пример LINQ

Локальная последовательность, так как она представляет локальную коллекцию объектов в памяти.

```
string[] names = { "Tom", "Dick", "Harry" };
```

Запрос — это выражение, которое при перечислении трансформирует последовательности с помощью операций запросов. Простейший запрос состоит из одной входной последовательности и одной операции. Например, применим операцию Where к простому массиву для извлечения элементов, длина которых составляет четыре символа:

```
IEnumerable<string> filteredNames = Enumerable.Where(names, n => n.Length >= 4);
```

Лямбда-выражение

Данный запрос записан с помощью **текучего синтаксиса**.

Текучий синтаксис является наиболее гибким и фундаментальным.

Простейший пример LINQ

C# также предлагает другой синтаксис для написания запросов, который называется синтаксисом выражений запросов.

Предыдущий запрос, записанный в виде выражения запроса:

```
IEnumerable<string> filteredNames = from n in names
                                    where n.Length >= 4
                                    select n;
```

Текучий синтаксис и синтаксис запросов дополняют друг друга.

- Любой LINQ - запрос, трансформируется в последовательность вызовов расширяющих методов.
- Запросы, оперирующие на локальных последовательностях, называются
- локальными запросами или запросами LINQ to Objects.

Анонимные типы в LINQ

Основное применение анонимных типов — это, конечно, LINQ. Фактически, они и были созданы для него (вообще говоря, все нововведения C# 3.0 были сделаны для LINQ, за исключением, пожалуй, частичных методов).

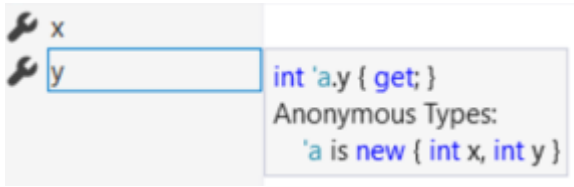
Анонимные типы обычно используются в предложении select выражения запроса для возврата подмножества свойств из каждого объекта в исходной последовательности.

```
List<int> collection = new List<int>() { 1, 2, 3, 4, 5 };

var query = from element in collection
            select new { x = element, y = element * 2 };
```

query – анонимный тип

```
foreach (var item in query)
{
    item.;
}
```



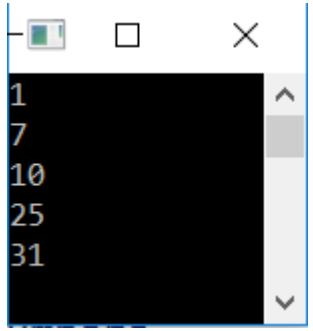
Сортировка LINQ

Для сортировки набора данных по возрастанию используется оператор **orderby**. Оператор orderby принимает критерий сортировки. По умолчанию оператор orderby производит сортировку по возрастанию.

```
List<int> numbers = new List<int>() { 1, 10, 7, 31, 25 };

var orderNumbers = from element in numbers
                   orderby element
                   select element;

foreach (var item in orderNumbers)
{
    Console.WriteLine(item);
}
```



С помощью ключевых слов **ascending** (сортировка по возрастанию) и **descending** (сортировка по убыванию) можно явным образом указать направление сортировки.

Сортировка LINQ

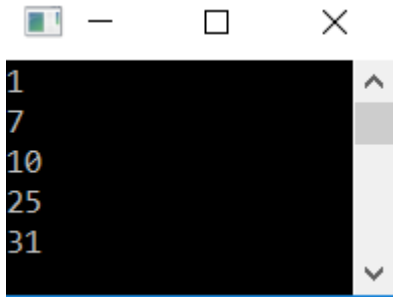
Вместо оператора orderby можно использовать методы расширения OrderBy.

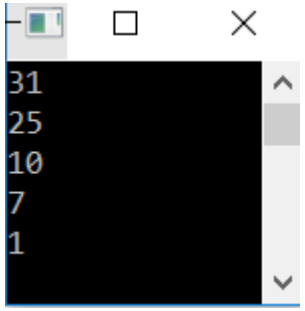
Метод OrderBy сортирует по возрастанию. Для сортировки по убыванию используется метод OrderByDescending.

```
List<int> numbers = new List<int>() { 1, 10, 7, 31, 25 };

var orderNumbers = numbers.OrderBy(t=>t);

var orderNumbersDescending = numbers.OrderByDescending(t => t);
```





Группировка

Для группировки данных по определенным параметрам применяется оператор group by или метод GroupBy().

```
List<Student> studentsList = new List<Student>()
{
    new Student(){Name = "Ivan", Course = "C# Base"},
    new Student(){Name = "Ivan", Course = "C# Advanced"},
    new Student(){Name = "Ivan", Course = "SQL"}
};
```

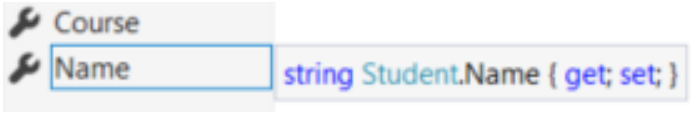
```
class Student
{
    public string Name { get; set; }
    public string Course { get; set; }
}
```

```
var groups = from student in studentsList
             group student by student.Name;
```

или так

```
var groups = studentsList.GroupBy(t => t.Name);
```

```
foreach (var group in groups)
{
    foreach (var item in group)
    {
        item.
    }
}
```



Простейший пример LINQ

Иногда возникает необходимость произвести в запросах LINQ какие-то дополнительные промежуточные вычисления. Для этих целей мы можем задать в запросах свои переменные с помощью оператора let.

Оператор let можно представить, как локальную переменную видимую только внутри выражения запроса.

```
var people = from u in users
             let name = "Mr. " + u.Name
             select new
             {
                Name = name,
                Age = u.Age
             };
```

```
List<User> users = new List<User>()
{
   new User { Name = "Sam", Age = 43 },
   new User { Name = "Tom", Age = 33 }
};
```

Динамические типы

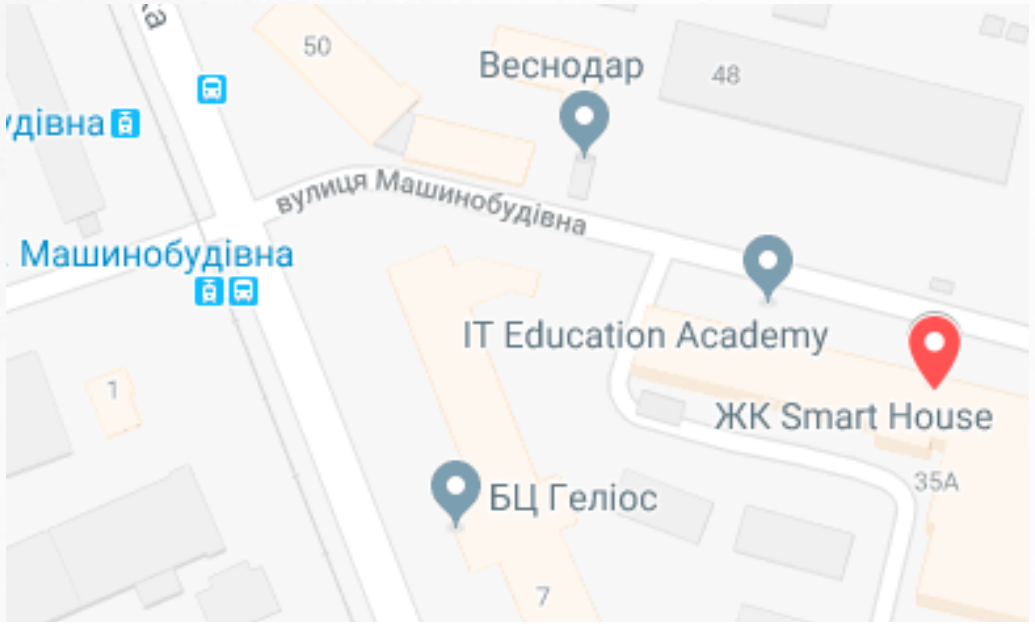
В версии .NET 4.0 язык C# стал поддерживать ключевое слово **dynamic**.

```
static void Main()
{
   dynamic variable = 100;
   variable = "Alexs";
   variable = DateTime.Now;
}
```

На самом высоком уровне dynamic можно рассматривать как специализированную форму System.Object, в том смысле, что типу данных **dynamic** может быть присвоено любое значение.

Динамические данные не типизированы статически. Для компилятора C# это выглядит так, что элемент данных, объявленный с ключевым словом dynamic, может получить какое угодно начальное значение, и на протяжении времени его существования это значение может быть заменено новым (и возможно, не связанным с первоначальным). Разработчику не нужно следить за тем, откуда объект получает свое значение.


Презентация окончена.
Спасибо за внимание!



I T E A

КОНТАКТНЫЕ ДАННЫЕ

ITEA

ЖК “Smart House”, ул. Машиностроительная, 41 (м.Берестейская)

ЖК «Корона» улица Срибнокильская,1 м. Позняки

+38 (044) 599-01-79
facebook.com/Itea
info@itea.ua
itea.ua