

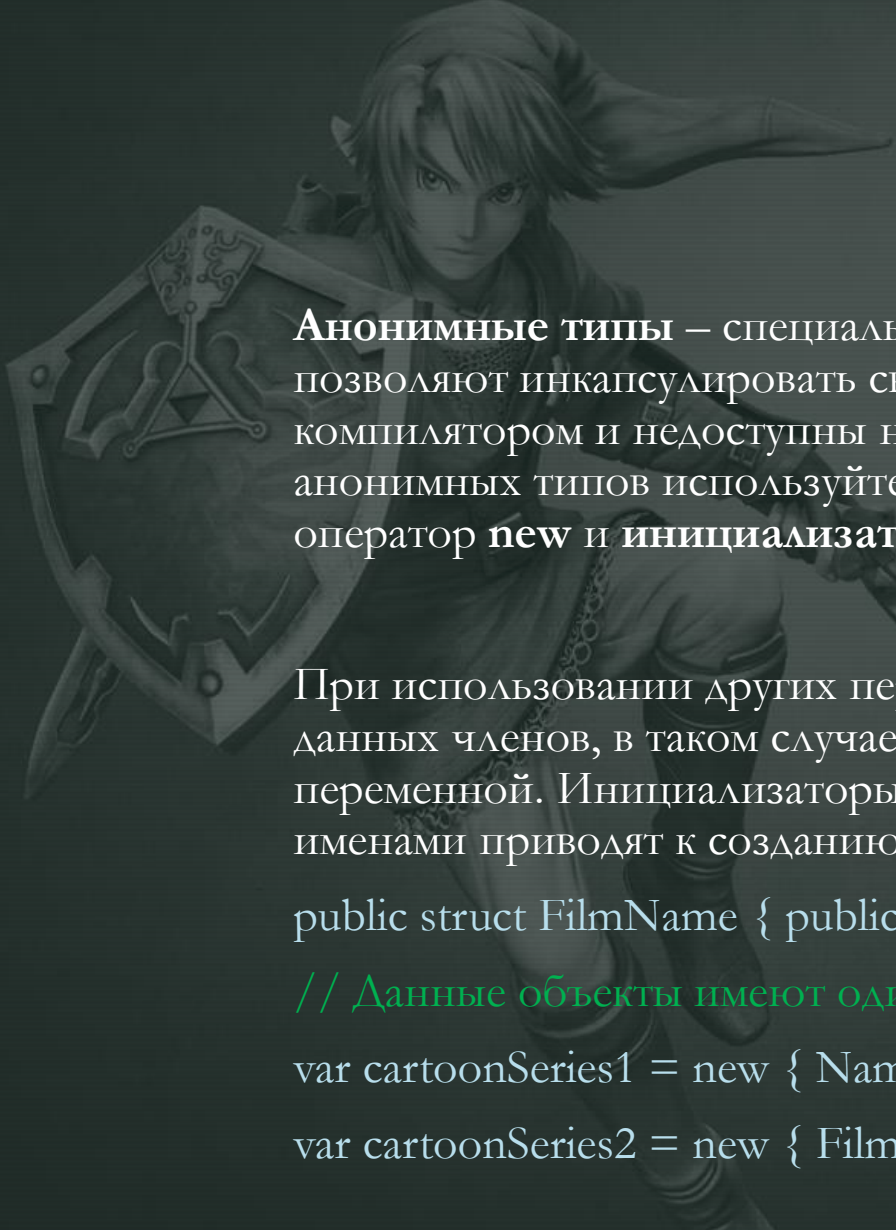
THE LEGEND OF ПИШНИК

Глава 17: Анонимные типы. `Link`
Основы LINQ в C#

Автор презентации – Сагалов Даниил БПИ-196



Анонимные Типы



Анонимные типы – специальные ссылочные типы, создаваемые компилятором, которые позволяют инкапсулировать свойства только для чтения. Имена анонимных типов создаются компилятором и недоступны на уровне исходного кода. Для объявления переменных анонимных типов используйте контекстно-ключевое слово **var**, а для создания самого объекта – оператор **new** и **инициализатор**. Методы или события в анонимных типах недопустимы.

При использовании других переменных в качестве членов Вы можете явно не указывать имена данных членов, в таком случае будет использоваться то же имя, что и у переданной переменной. Инициализаторы свойств с одинаковым порядком, одинаковыми типами и именами приводят к созданию объект одного и того же анонимного типа. Примеры:

```
public struct FilmName { public static string Name = “Ben 10” };
```

```
// Данные объекты имеют одинаковый тип
```

```
var cartoonSeries1 = new { Name = “Winx”, Rating = 11 };
```

```
var cartoonSeries2 = new { FilmName.Name, Rating = 10 };
```

В первую очередь анонимные типы применяются при работе с запросами, так как позволяют создавать объекты, содержащие только нужные данные из объектов различных типов.

Два экземпляра анонимных типов считаются равными только в том случае, если равны все их свойства (проверка может быть осуществлена через `Equals()`).

При попытке использования `ToString()` для анонимного типа будет выведен список всех свойств с их значениями в фигурных скобках через запятую:

```
var dataset = new { Id = 0, Content = "Hello World!" };
```

```
Console.WriteLine(dataset);
```

```
// Вывод:
```

```
// { Id = 0, Content = Hello World! }
```

Применение Анонимных Типов

Обзор LINQ



Language Integrated Query (LINQ) – набор технологий, создающих и использующих возможности интеграции запросов в C#.

Вы можете использовать технологии LINQ для превращения запросов в языковые конструкции, применимые аналогично классам методам и событиям. Для выражений запросов используется декларативный синтаксис. Синтаксис запросов применим для фильтрации, упорядочивания и группировки данных с минимальными затратами по объёму кода. Стоит отметить, что одни и те же базовые выражения запросов позволяют получать и преобразовывать различные данные из источников (SQL, ADO .NET, XML, коллекций .NET).

Пример:

```
int[] chicken = { 95, 96, 97, 98, 99, 100 };  
  
// Получаем перечисление всех чётных элементов  
  
IEnumerable<int> result = from evenChicken in chicken  
                           where evenChicken % 2 == 0  
                           select evenChicken;  
  
foreach(int even in result)  
    Console.Write(even + " ");
```

Основные Сведения о LINQ

- Выражения запросов можно использовать с любыми источниками данных, поддерживающими LINQ.
- Все переменные в выражениях запросов строго типизированы, хоть и определяются компилятором автоматически.
- Запрос не будет выполняться до тех пор, пока Вы не обратитесь к переменной запроса.
- При компиляции все запросы преобразуются в соответствующие вызовы методов. Любой запрос, использующий синтаксис запросов может быть заменён эквивалентным с помощью синтаксиса методов.
- Синтаксис запросов может комбинироваться с синтаксисом методов, однако некоторые методы не имеют эквивалентных им запросов.
- Точка с запятой ставится только в самом конце запроса.

Структура LINQ Запросов

```
// Шаг 1: Источник данных
string[] touchingWords = { "somebody", "once", "told", "me",
    "the", "world", "is", "gonna", "roll", "me"};

// Шаг 2: Формирование запроса (с отложенным выполнением)
var theOList = from words in touchingWords
                where words[1] == 'o'
                select words;

// Шаг 3: Выполнение запроса
foreach (string word in theOList)
    Console.WriteLine(word);

// Пример принудительного выполнения того же запроса
// Тип будет определён компилятором как List<string>
var theInstantOList = (from words in touchingWords
                        where words[1] == 'o'
                        select words).ToList();
```

Все операции запросов LINQ состоят из 3 отдельных действий: **получения источника данных, создания запроса и выполнения запроса.**

Источник данных должен реализовывать интерфейс `IEnumerable`, `IEnumerable<T>` или любой производный от них. В контексте запросов такие типы называются запрашиваемыми.

Сама переменная запроса не выполняет никаких действий и не возвращает никаких данных. Она просто хранит сведения, необходимые для предоставления результатов запроса (команды запроса).

Помните, что выполнение запроса откладывается до использования переменной запроса в операторе `foreach`. Благодаря этой особенности Вы, например, можете создать запрос, регулярно обновлять данные и вызывать один и тот же запрос. Такое выполнение называют **отложенным**.

При этом также вводится понятие **принудительного выполнения запроса**. Это относится к запросам LINQ, которые для получения результата должны сами выполнить итерацию по последовательности (например, `Max`, `Count` или `Average`). При этом такие запросы возвращают значение, а не перечислимую коллекцию.

Основные Операции Запросов LINQ

Основные операции LINQ запросов можно поделить на 6 групп: **получение источника данных, фильтрация, сортировка, группировка, соединение и выбор (проекция).**

Получение Источника Данных

Для указания источника в начале запроса используется предложение **from** и переменная диапазона:

from <имя переменной диапазона> **in** <источник>

Переменные диапазона схожи с итерационными переменными `foreach` за исключением того, что фактической итерации не осуществляется. Тип переменных запросов определяется автоматически при работе с универсальными коллекциями, однако для неуниверсальных он должен указываться явно. В дальнейшем будет рассмотрено создание дополнительных переменных диапазона с помощью предложений `let`.

Фильтрация

Фильтры позволяют запросу возвращать только те элементы, которые удовлетворяют указанным условиям. Для фильтрации используется предложения **where**, Вы можете указывать множественные условия с помощью операторов `&&` и `||`.

where <имя переменной диапазона> <условие 1 || условие 2 && условие 3 ...>

Основные Операции Запросов LINQ

Сортировка

Предложения **orderby** позволяют сортировать возвращаемые данные в зависимости от компаратора по умолчанию. Дополнительно может быть явно указан порядок сортировки – `ascending` или `descending`, без явного указания сортировка осуществляется по возрастанию. Можно подряд указать несколько критериев сортировки по уменьшению их приоритета.

orderby <имя переменной диапазона> [`ascending/descending`], ...

Группировка

Для группировки данных на основании указанного ключа воспользуйтесь предложением **group**. Результаты `group` представляют из себя списки из списков. Каждый элемент списка является объектом, имеющим член `Key` и список элементов, сгруппированных по данному ключу. Для итерации запросов, возвращающих последовательность групп используйте вложенный цикл `foreach`. При необходимости обращения к результатам группировки, можно использовать ключевое слово `into` для создания идентификатора

group <имя переменной диапазона> **by** <ключ> [**into** <имя новой переменной диапазона>]

Основные Операции Запросов LINQ

Соединение

Операция соединения позволяют создавать связи между несколькими источниками данных по некоторому ключу, для неё используется предложение **join**.

orderby <имя переменной диапазона> [ascending/descending], ...

Выбор и Проекция

Предложение **select** создаёт результаты запроса, и задаёт форму/тип каждого возвращаемого элемента. Если предложение **select** создаёт что-либо отличное от копии исходного элемента, операция называется **проекцией**. Использование проекций для преобразования данных является одной из крайне полезных функций LINQ.

select <имя переменной диапазона>

Ключевые Слова Запросов

Ключевое Слово	Описание	Ключевое Слово	Описание
from	Задаёт источник данных и переменную диапазона.	let	Создаёт переменную диапазона для хранения результатов выражения, вложенного в выражение запроса.
where	Фильтрует данные источника на основе логических выражений (с && и).	in	Контекстно-ключевое слово в предложении join.
select	Задаёт тип и форму элементов возвращаемой последовательности при выполнении запроса.	on	Контекстно-ключевое слово в предложении join.
group	Группирует результаты запроса по заданному значению ключа.	equals	Контекстно-ключевое слово в предложении join.
into	Идентификатор, используемый в качестве ссылки на результаты join, group или select.	by	Контекстно-ключевое слово в предложении group.
orderby	Сортировка по возрастанию/убыванию на основе функции сравнения по умолчанию.	ascending	Контекстно-ключевое слово в предложении orderby.
join	Соединяет два источника данных посредством сравнения на равенство между двумя заданными критериями сопоставления.	descending	Контекстно-ключевое слово в предложении orderby.

Помните, что **выражение запроса должно начинаться с from**. При этом допустимо использование нескольких from или вложенных from в одном запросе. Как уже говорилось ранее, from определяет строго типизированные источник данных и переменную диапазона. Строгая типизация позволяет обращаться к членам используемых в запросах типов.

При использовании вложенных последовательностей вы также можете использовать и вложенные from (например, в ситуации, когда вы оперируете со списком разработчиков и у каждого из них есть список оценок активности в GitHub).

При некоторых сценариях from позволяет соединять данные из нескольких различных независимых источников, что было бы невозможно с join.

Предложение from

Пример (from)

Ссылка: 5

```
public class Developer {
```

Ссылка: 6

```
    public int Id { get; private set; }
```

```
    public readonly List<int> rating;
```

ссылка: 1

```
    public Developer(int id, params int[] ratings) {
```

```
        Id = id;
```

```
        rating = new List<int>();
```

```
        rating.AddRange(ratings);
```

```
    }
```

```
}
```

Ссылка: 4

```
public class Request {
```

Ссылка: 2

```
    public int Id { get; private set; }
```

Ссылка: 2

```
    public string Text { get; private set; }
```

ссылка: 1

```
    public Request(int id, string text) {
```

```
        Id = id;
```

```
        Text = text;
```

```
    }
```

```
}
```

```
public static void Main() {
    Random rnd = new Random();
    List<Developer> developers = new List<Developer>();
    List<Request> requests = new List<Request>();

    for (int i = 0; i < 10; ++i) {
        int[] points = new int[10];
        for (int j = 0; j < 10; ++j)
            points[j] = rnd.Next(1, 11);
        developers.Add(new Developer(i, points));
    }

    for (int i = 0; i < 10; ++i) {
        int[] points = new int[10];
        for (int j = 0; j < 10; ++j)
            points[j] = rnd.Next(1, 11);
        requests.Add(new Request(i, ((char)('a' + i)).ToString()));
    }

    // Пример 1 - выбор всех разработчиков с ID, кратным 3
    var fromExample1 = from dev in developers
                       where dev.Id % 3 == 0
                       select dev;

    foreach (Developer dev in fromExample1)
        Console.WriteLine($"Developer Id: {dev.Id}");

    // Пример 2 - вложенный from для выбора рейтингов
    // выше 5 для всех разработчиков
    var fromExample2 = from dev in developers
                       from ratings in dev.rating
                       where ratings > 5
                       select ratings;

    foreach (int rating in fromExample2)
        Console.WriteLine(rating);

    // Пример 3 - использование from с несколькими источниками
    // и формирование объектов анонимного типа по критерию
    var fromExample3 = from dev in developers
                       from req in requests
                       where req.Id == dev.Id && dev.Id % 2 == 0
                       select new { Id = dev.Id, Rating = dev.rating, Request = req.Text };
    foreach (var anonymous in fromExample3)
        Console.WriteLine(anonymous);
}
```

Как уже было описано на предыдущих слайдах, **orderby** задаёт сортировку возвращаемой последовательности по возрастанию (по умолчанию или при указании **ascending**) или по убыванию (**descending**).

При использовании **orderby** стоит учитывать, что Вы можете перечислить через запятую несколько критериев в порядке убывания их приоритета, для каждого при необходимости указывая порядок возрастания/убывания сортировки явно.

Для того, чтобы задать настраиваемую функцию сравнения, воспользуйтесь синтаксисом методов (**OrderBy()** и **ThenBy()**).

Применение **orderby** к элементам, не поддерживающим сортировку приводит к исключению.

Предложение **orderby**

Пример (orderby)

```
public class Product {  
    Ссылка: 3  
    public string Name { get; private set; }  
    Ссылка: 3  
    public decimal Price { get; private set; }  
    Ссылка: 3  
    public string Type { get; private set; }  
  
    Ссылка: 7  
    public Product(string name, decimal price, string type = "None") {  
        Name = name;  
        Price = price;  
        Type = type;  
    }  
    Ссылка: 0  
    public override string ToString() {  
        return $"Product Name: {Name}, Price: ${Price}" +  
            $"{(Type == "None" ? "" : $", Type: {Type}")}";  
    }  
}  
  
class Program  
{  
    Ссылка: 0  
    static void Main() {  
        List<Product> storeStock = new List<Product>() {  
            new Product("Nokia 3310", 6400),  
            new Product("Apple iPhone", 120000, "Flagship"),  
            new Product("Samsung Galaxy", 80000, "S Series"),  
            new Product("Apple iPhone", 42500, "SE"),  
            new Product("Samsung Galaxy", 100000, "Note Series"),  
            new Product("Xiaomi Mi", 60000, "Mix"),  
            new Product("Xiaomi Mi", 48000, "Flagship")  
        };  
  
        // Сортировка товара по лексикографическому убыванию  
        // имён с возрастанием цен в подкатегориях  
        var sortedStock = from product in storeStock  
                           orderby product.Name descending,  
                                product.Price ascending  
                           select product;  
        foreach (var product in sortedStock)  
            Console.WriteLine(product);  
    }  
}
```




Предложения select и group

Любой LINQ запрос должен заканчиваться либо предложением `group`, либо предложением `select`.

Как известно, предложение `select` задаёт тип значений, которые будут получены при выполнении запроса, причём ключевое слово `new` позволяет создавать объекты анонимных типов.

Предложение `group` позволяет вернуть последовательность объектов **`IGrouping<out TKey, out TElement>`**, содержащую 0+ элементов, соответствующих значению ключа `TKey`. Ключ группы всегда можно получить через свойство **`Key`**. В итоге полученная последовательность имеет тип **`IEnumerable<IGrouping<TKey, TElement>>`** (из-за чего и возникает необходимость во вложенном цикле при её переборе). После предложения `group` для дальнейшего действия может создаваться идентификатор с помощью `into`.

В качестве ключей группировки могут выступать различные критерии, включая создаваемые анонимные типы.

Пример (group)

```
static void Main() {
    string[] animals = {"elephant", "goat", "hippo", "crocodile", "goose", "cat",
        "monkey", "cuckoo", "chameleon", "horse", "donkey", "booby", "duck", "kakapo"};

    var alphabetSortedAnimals = from gp in animals
                                group gp by gp[0] into gps
                                where gps.Key == 'b' || gps.Key == 'c' || gps.Key == 'h' ||
                                    gps.Key == 'g'
                                select gps;

    /*          ВЫВОД:
     *
     * Animals starting with g: goat goose
     * Animals starting with h: hippo horse
     * Animals starting with c: crocodile cat cuckoo chameleon
     * Animals starting with b: booby
     *
     */

    foreach (IGrouping<char,string> letterGroups in alphabetSortedAnimals)
    {
        Console.WriteLine($"Animals starting with {letterGroups.Key}: ");
        foreach (string animal in letterGroups)
            Console.WriteLine(animal + " ");
        Console.WriteLine();
    }
}
```

Методы Расширений LINQ: Сортировка

Метод	Описание
OrderBy	Сортировка значений по возрастанию.
OrderByDescending	Сортировка значений по убыванию.
ThenBy	Дополнительная сортировка по возрастанию.
ThenByDescending	Дополнительная сортировка по убыванию.
Reverse	Замена порядка коллекции на обратный.

Методы Расширений LINQ: Операции над Множествами

Метод	Описание
Distinct	Удаляет повторяющиеся значения коллекции.
Except	Возвращает разность множеств, т. е. элементы одной коллекции, отсутствующие во второй.
Intersect	Возвращает пересечение двух множеств, т. е. элементы, присутствующие одновременно в обеих коллекциях.
Union	Возвращает коллекцию, состоящую из уникальных элементов каждой из двух коллекций.

Методы Расширений LINQ: Фильтрация. Проекция

Метод	Описание
OfType	Выбирает все значения, которые можно привести к указанному типу.
Where	Выбирает все значения, удовлетворяющие указанному предикату.

Метод	Описание
Select	Проецирует значения с использованием функции преобразования.
SelectMany	Проецирует последовательности значений, основанных на функции преобразования, а затем выстраивает их в одну последовательность (аналогично нескольким from)

Методы Расширений LINQ: Поиск. Объединение

Метод	Описание
All	Проверяет, все ли элементы последовательности удовлетворяют указанному условию.
Any	Проверяет, удовлетворяет ли хотя бы один элемент указанному условию.
Contains	Проверяет, содержит ли данная последовательность указанный элемент.

Метод	Описание
Concat	Объединяет две последовательности в одну.

Методы Расширений LINQ: Секционирование

Метод	Описание
Skip	Пропускает элементы последовательности до указанной позиции.
SkipWhile	Пропускает элементы до тех пор, пока не будет удовлетворено условие предиката.
Take	Возвращает указанное число элементов с начала последовательности.
TakeWhile	Возвращает элементы пока выполняется условие предиката.

Методы Расширений LINQ: Соединение. Группировка

Метод	Описание
Join	Соединяет две последовательности на основании функций селектора ключа и извлекает пары значений.
GroupJoin	Соединяет две последовательности на основании функций селектора ключа и группирует полученные при сопоставлении данные для каждого элемента.

Метод	Описание
GroupBy	Группирует элементы с общим атрибутом. Каждая группа представлена объектом <code>IGrouping<TKey,TElement></code> .
ToLookup	Вставляет элементы в <code>Lookup<TKey,TElement></code> (словарь «один ко многим») в зависимости от функции выбора ключа.

Методы Расширений LINQ: Создание. Сравнение

Метод	Описание
DefaultIfEmpty	Заменяет пустую последовательность на одноэлементную со значением по умолчанию.
Empty	Возвращает пустую последовательность.
Range	Создаёт коллекцию, содержащую последовательность чисел.
Repeat	Создаёт коллекцию, содержащую одно значение, повторяющееся n раз.

Метод	Описание
SequenceEqual	Определяет, равны ли две последовательности, сравнивая их элементы попарно.

Методы Расширений LINQ: Операции с Элементами

Метод	Описание
ElementAt	Возвращает элемент коллекции с указанным индексом.
ElementAtOrDefault	Возвращает элемент коллекции с указанным индексом или значение по умолчанию, если индекс выходит за пределы допустимого диапазона.
First	Возвращает первый элемент коллекции или первый элемент, удовлетворяющий условию.
FirstOrDefault	Возвращает первый элемент коллекции или первый элемент, удовлетворяющий условию. Если такого элемента не существует, возвращает значение по умолчанию.
Last	Возвращает последний элемент коллекции или последний элемент, удовлетворяющий условию.
LastOrDefault	Возвращает последний элемент коллекции или последний элемент, удовлетворяющий условию. Если такого элемента не существует, возвращает значение по умолчанию.
Single	Возвращает единственный элемент коллекции или единственный элемент, удовлетворяющий условию. Создает исключение InvalidOperationException , если нет ни одного элемента для возврата или таких элементов несколько.
SingleOrDefault	Возвращает единственный элемент коллекции или единственный элемент, удовлетворяющий условию. Возвращает значение по умолчанию, если нет элементов для возврата. Создает исключение InvalidOperationException , если существует несколько элементов для возврата.

Методы Расширений LINQ: Преобразования

Метод	Описание
AsEnumerable	Возвращает входное значение как IEnumerable<T>.
AsQueryable	Преобразует IEnumerable/IEnumerable<T> в IQueryable/IQueryable<T>.
Cast	Приводит элементы коллекции к указанному типу.
ToArray	Преобразует коллекцию в массив. Приводит к принудительному выполнению запроса.
ToDictionary	Помещает элементы в Dictionary<TKey,TValue> в зависимости от функции выбора ключа. Приводит к принудительному выполнению запроса.
ToList	Преобразует коллекцию в List<T>. Приводит к принудительному выполнению запроса.
ToLookup	Помещает элементы в Lookup<TKey,TElement> (словарь «один ко многим») в зависимости от функции выбора ключа. Приводит к принудительному выполнению запроса.

Методы Расширений LINQ: Агрегирование

*Данные методы вычисляют одно значение по коллекции. В связи с этим, их использование приводит к принудительному выполнению запросов.

Метод	Описание
Aggregate	Выполняет пользовательскую операцию агрегирования над коллекцией.
Average	Вычисляет среднее значение коллекции значений.
Count	Подсчитывает количество элементов коллекции, имеет перегрузку для подсчёта количества элементов коллекции, удовлетворяющих указанному предикату.
LongCount	Подсчитывает количество элементов большой коллекции, имеет перегрузку для подсчёта количества элементов коллекции, удовлетворяющих указанному предикату.
Max	Определяет максимальное значение коллекции.
Min	Определяет минимальное значение коллекции.
Sum	Вычисляет сумму значений коллекции.