Коллекции стандартной библиотеки, LINQ

Юрий Литвинов
y.litvinov@spbu.ru

11.04.2025

Контейнеры стандартной библиотеки

- System.Array — массив, встроен в язык
- System.Collections — негенериковые коллекции
- System.Collections.Generic — генериковые коллекции
- System.Collections.Concurrent — потокобезопасные коллекции
- System.Collections.Immutable — немутабельные коллекции

Архитектурные соглашения в стандартной библиотеке

- Любая коллекция возвращает энумератор
  - Есть IEnumerable и IEnumerable<T>, второй наследуется от первого
  - С IEnumerable<T> работает LINQ
- Все коллекции реализуют CopyTo — копирование в массив
- Есть Capacity и Count, но не у всех коллекций
- Все индексированные коллекции начинают индексы с нуля
- Все коллекции из System.Collections и System.Collections.Generic не потокобезопасны
- Все коллекции вида «ключ-значение» ожидают, что ключи не будут меняться
  - Но не могут это проверить

Какие коллекции вообще бывают

- Dictionary<TKey,TValue> — словарь на хеш-таблицах
- HashSet<T> — множество на хеш-таблицах, умеет операции
- LinkedList<T> — двусвязный список
  - LinkedListNode<T>
- List<T> — список на массивах
- PriorityQueue<TElement,TPriority> — очередь на куче
  - «array-backed quaternary min-heap»
- Queue<T> — очередь на круговых массивах
- SortedDictionary<TKey,TValue> — словарь на красно-чёрных деревьях
- SortedList<TKey,TValue> — последовательность пар (или словарь) на сортированных списках с двоичным поиском
- SortedSet<T> — множество на красно-чёрных деревьях
- Stack<T> — стек на массивах

Алгоритмическая сложность

Операция	Амортизированная трудоёмкость	Худший случай
Stack<T>.Push	O(1)	O(n)
Queue<T>.Enqueue	O(1)	O(n)
List<T>.Add	O(1)	O(n)
List<T>.Item[Int32]	O(1)	O(1)
List<T>.Enumerator	O(n)	O(n)
HashSet<T>.Add, поиск	O(1)	O(n)
SortedSet<T>.Add	O(log n)	O(n)
Dictionary<T>.Add	O(1)	O(n)
поиск в Dictionary<T>	O(1)	O(1) или O(n)
SortedDictionary<T>.Add	O(log n)	O(n log n)

Вспомогательные интерфейсы и методы

- Object.GetHashCode используется всеми коллекциями на хеш-таблицах
  - Обязан возвращать равные значения для равных (в терминах Equals) объектов
  - Хеш-код нельзя хранить между запусками приложения или передавать
- IEqualityComparer<T> — интерфейс компаратора со встроенным GetHashCode «вне» объекта
  - Коллекции на хеш-таблицах могут принимать его в конструктор, тогда «родной» GetHashCode игнорируется
  - EqualityComparer<T> и его метод Create для быстрой реализации
- IComparer<T> — то же для сортированных коллекций
  - Сравнения уважают культуру
    - См. CultureInfo и InvariantCulture

Вспомогательные интерфейсы и методы (2)

- IEquatable<T> — определяет равенство «изнутри» элемента
  - Используется в Contains, IndexOf и т.п.
  - Если нет, используется Object.Equals
- IComparable<T> — то же для сравнения объектов
  - Есть негенериковая версия, IComparable
  - Если не реализован, компаратор надо передавать явно
- IReadOnlyCollection<T> и его потомки — часть ICollection<T> без методов изменения
  - Никто не мешает коллекцию, которую передали по этому интерфейсу, менять извне
  - AsReadOnly<T> в CollectionExtensions

Как писать свои коллекции

- Не делайте этого, всё написано до вас
- Collection<T> — базовый класс для реализации своих архитектурно правильных коллекций
  - Переопределением protected-методов InsertItem, RemoveItem, ClearItems, SetItem
  - Внутри по умолчанию List<T>, можно использовать «из коробки»

Language Integrated Query

- LINQ — набор методов-расширений для операций над разными структурами данных
- Прежде всего для работы с базами данных, но хорошо работает и с коллекциями (LINQ to Objects)
  - Предназначался прежде всего, чтобы запросы к БД не надо было писать строками в коде
- Имеет свой набор ключевых слов, делающих запрос похожим на SQL
- Архитектурно интересно устроен
  - Запрос не исполняется, пока не нужен результат (ленивость)
  - Запрос может транслироваться перед исполнением (например, в SQL)
- Живёт в System.Linq
- В Java ближайший аналог (но не такой навороченный) — Stream API

Пример

```
string sentence = "the quick brown fox jumps over the lazy dog";
string[] words = sentence.Split(' ');

var query = words.
    GroupBy(w => w.Length, w => w.ToUpper()).
    Select(g => new { Length = g.Key, Words = g }).
    OrderBy(o => o.Length);

foreach (var obj in query)
{
    Console.WriteLine($"Words of length {obj.Length}:");
    foreach (string word in obj.Words)
        Console.WriteLine(word);
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/

Или то же самое на встроенном языке запросов

```
string sentence = "the quick brown fox jumps over the lazy dog";
string[] words = sentence.Split(' ');

var query = from word in words
    group word.ToUpper() by word.Length into gr
    orderby gr.Key
    select new { Length = gr.Key, Words = gr };

foreach (var obj in query)
{
    Console.WriteLine($"Words of length {obj.Length}:");
    foreach (string word in obj.Words)
        Console.WriteLine(word);
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/

Выборка и преобразование

Операция	Описание
Where	Возвращает коллекцию, элементы которой удовлетворяют предикату (filter)
Select	Возвращает коллекцию, полученную применением лямбда-функции к каждому элементу исходной (map)
SelectMany	Возвращает коллекцию, полученную применением лямбда-функции, возвращающей коллекцию, к каждому элементу исходной, и склеивает результирующие коллекции в одну (collect)
Zip	Превращает две последовательности в последовательность пар (zip)

SelectMany, пример

```
List<string> phrases = ["an apple a day", "the quick brown fox"];

var query = phrases.SelectMany(phrases => phrases.Split(' '));

foreach (string s in query)
{
    Console.WriteLine(s);
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/projection-operations

Zip, пример

```
IEnumerable<int> numbers = [1, 2, 3, 4, 5, 6, 7];
IEnumerable<char> letters = ['A', 'B', 'C', 'D', 'E', 'F'];
IEnumerable<string> emoji = [ ":)", ":D", ";)", ":(", ":/", "=^_^="];

foreach ((int number, char letter, string em)
    in numbers.Zip(letters, emoji))
{
    Console.WriteLine(
        $"Number: {number} is zipped with letter: '{letter}' and emoji: {em}");
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/projection-operations

Операции над множествами

Операция	Описание
Distinct	Возвращает коллекцию без дубликатов (с точки зрения Equals или переданного компаратора)
DistinctBy	Distinct, но можно указать лямбда-функцию — проекцию, которая должна возвращать ключ
Except	Возвращает элементы первой коллекции, которых нет во второй
ExceptBy	Except, но с проекцией, отображающей элементы первой коллекции во вторую
Intersect	Возвращает коллекцию элементов, которые встречаются в обеих исходных коллекциях
IntersectBy	Intersect с проекцией
Union	Возвращает коллекцию элементов, которые встречаются в одной из исходных коллекций
UnionBy	Union с проекцией

DistinctBy, пример

```
string[] words = ["the", "quick", "brown", "fox", "jumped",
    "over", "the", "lazy", "dog"];

foreach (var word in words.DistinctBy(p => p.Length))
{
    Console.WriteLine(word);
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/set-operations

ExceptBy, пример

```
string[] words = ["the", "quick", "brown", "fox", "jumped",
    "over", "the", "lazy", "dog"];
int[] forbiddenLengths = [3, 4];

foreach (var word in words.ExceptBy(forbiddenLengths,
    w => w.Length))
{
    Console.WriteLine(word);
}
```

IntersectBy, пример

```
// Student — класс со свойствами FirstName и LastName
// Teacher — с аналогичными свойствами First и Last
// Найдём всех студентов, кто не является учителем
foreach (Student person in
    students.IntersectBy(
        teachers.Select(t => (t.First, t.Last)),
            s => (s.FirstName, s.LastName)))
{
    Console.WriteLine($"{person.FirstName} {person.LastName}");
}
```

Сортировка

Операция	Описание
OrderBy	Сортировка по возрастанию по ключу
OrderByDescending	Сортировка по убыванию по ключу
ThenBy	Сортировка по вторичному признаку (возможно, потому что методы сортировки возвращают IOrderedEnumerable<TElement>)
ThenByDescending	Сортировка по вторичному признаку по убыванию
Reverse	Развернуть коллекцию

ThenBy, пример

```
string[] fruits = { "grape", "passionfruit", "banana", "mango",
    "orange", "raspberry", "apple", "blueberry" };

// Сортируем по длине и затем равные по длине — по алфавиту
IEnumerable<string> query =
    fruits.OrderBy(fruit => fruit.Length).ThenBy(fruit => fruit);

foreach (string fruit in query)
{
    Console.WriteLine(fruit);
}
```

Квантификаторы

Операция	Описание
All	Возвращает true, если для всех элементов коллекции выполнен предикат
Any	Возвращает true, если для хотя бы одного элемента коллекции выполнен предикат
Contains	Возвращает true, коллекция содержит указанный элемент

Разбиение коллекций

Операция	Описание
Skip	Возвращает коллекцию без первых n элементов
Take	Возвращает коллекцию из первых n элементов
SkipWhile	Возвращает коллекцию, пропустив первые элементы, для которых выполняется предикат
TakeWhile	Возвращает коллекцию, из элементов до первого, для которого не выполняется предикат
Chunk	Делит коллекцию на куски указанного размера

TakeWhile, SkipWhile, пример

```
foreach (int number in Enumerable.Range(0, 8).TakeWhile(n => n < 5))
{
    Console.WriteLine(number);
}

foreach (int number in Enumerable.Range(0, 8).SkipWhile(n => n < 5))
{
    Console.WriteLine(number);
}
```

Chunk, пример

```
int chunkNumber = 1;
foreach (int[] chunk in Enumerable.Range(0, 8).Chunk(3))
{
    Console.WriteLine($"Chunk {chunkNumber++}:");
    foreach (int item in chunk)
    {
        Console.WriteLine($"    {item}");
    }

    Console.WriteLine();
}
```

Преобразование типов

Операция	Описание
Cast	Преобразует все элементы коллекции к указанному виду
OfType	Оставляет в коллекции только значения, которые можно преобразовать к указанному типу
ToArray	Превращает коллекцию в массив
ToDictionary	Превращает коллекцию в словарь, используя данную лямбду для вычисления ключа элемента
ToList	Превращает коллекцию в список

OfType, пример

```
var fruits = new System.Collections.ArrayList(4);
fruits.Add("Mango");
fruits.Add("Orange");
fruits.Add("Apple");
fruits.Add(3.0);
fruits.Add("Banana");

// Apply OfType() to the ArrayList.
IEnumerable<string> query1 = fruits.OfType<string>();

Console.WriteLine("Elements of type 'string' are:");
foreach (string fruit in query1)
{
    Console.WriteLine(fruit);
}
```

Cast, пример

```
var fruits = new System.Collections.ArrayList();
fruits.Add("mango");
fruits.Add("apple");
fruits.Add("lemon");

IEnumerable<string> query =
    fruits.Cast<string>()
    .OrderBy(fruit => fruit)
    .Select(fruit => fruit);

foreach (string fruit in query)
{
    Console.WriteLine(fruit);
}
```

© https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.cast?view=net-8.0

Join

Операция	Описание
Join	Zip с проекциями, которые определяют, по каким ключам выполнять JOIN
GroupJoin	Join, но с последующим преобразованием результата

Join, пример

```
var query = students.Join(departments,
    student => student.DepartmentID, department => department.ID,
    (student, department) => new {
        Name = $"{student.FirstName} {student.LastName}",
        DepartmentName = department.Name });

foreach (var item in query)
{
    Console.WriteLine($"{item.Name} - {item.DepartmentName}");
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/join-operations

GroupJoin, пример

```
IEnumerable<IEnumerable<Student>> studentGroups =
    departments.GroupJoin(students,
        department => department.ID,
        student => student.DepartmentID,
    (department, studentGroup) => studentGroup);

foreach (IEnumerable<Student> studentGroup in studentGroups)
{
    Console.WriteLine("Group");
    foreach (Student student in studentGroup)
    {
        Console.WriteLine($"  - {student.FirstName}, {student.LastName}");
    }
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/join-operations

Группировка данных

Операция	Описание
GroupBy	Раскладывает элементы коллекции по группам (IGrouping<TKey,TElement>) по данной функции-проекции
ToLookup	То же, что GroupBy, но выкладывает элементы в Lookup<TKey,TElement>

GroupBy, пример

```
List<int> numbers = [35, 44, 200, 84, 3987, 4, 199, 329, 446, 208];

IEnumerable<IGrouping<int, int>> query = numbers
    .GroupBy(number => number % 2);

foreach (var group in query)
{
    Console.WriteLine(group.Key == 0 ?
        "\nEven numbers:" : "\nOdd numbers:");
    foreach (int i in group)
    {
        Console.WriteLine(i);
    }
}
```

© https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/grouping-data

Агрегатные операции

Операция	Описание
Aggregate	Свёртка. Применяет функцию последовательно к каждому элементу коллекции и состоянию, меняя состояние
Average	Считает среднее для числовых коллекций
Min, Max	Считает минимум/максимум для числовых коллекций или для любых коллекций, если передать лямбду, отображающую элемент в число
MinBy, MaxBy	То же, что Min/Max, но с лямбдой-проекцией и опциональным компаратором для произвольного типа ключа
Sum	Считает сумму для числовых коллекций или для любых коллекций, если передать лямбду, отображающую элемент в число

Типы операций

- Немедленные (Immediate) — требуют материализации коллекции
  - Все агрегатные операции
  - ToArray, ToList
- Отложенные (ленивые, Deferred) — исполняются только когда нужен результат
  - Можно понимать, как энумератор, «надеваемый» на исходный
  - Могут возвращать разные результаты в разные моменты времени
- Отложенные потоковые — возвращают результат для каждого элемента исходной коллекции, если надо (например, Select, Where)
- Отложенные непотоковые — требуют просмотра всей коллекции для возврата результата (например, Order, GroupBy)

Встроенный язык написания запросов

```
// Specify the data source.
int[] scores = [97, 92, 81, 60];

// Define the query expression.
IEnumerable<int> scoreQuery =
    from score in scores
    where score > 80
    select score;

// Execute the query.
foreach (var i in scoreQuery)
    Console.Write($"{i} ");
```

- Синтаксический сахар над методами, любой запрос можно записать в «методной» форме
- Можно комбинировать ключевые слова и методы

Части запроса

- Начинается с from
  - Объявляются источники данных и переменные, по которым на них будут ссылаться в запросе
- Заканчивается на select или group
- Внутри — where, orderby, join, let или ещё from
- into — сохранить промежуточные результаты

Пример

```
// percentileQuery is an IEnumerable<IGrouping<int, Country>>
var percentileQuery =
    from country in countries
    let percentile = (int)country.Population / 10_000_000
    group country by percentile into countryGroup
    where countryGroup.Key >= 20
    orderby countryGroup.Key
    select countryGroup;

// grouping is an IGrouping<int, Country>
foreach (var grouping in percentileQuery)
{
    Console.WriteLine(grouping.Key);
    foreach (var country in grouping)
    {
        Console.WriteLine(country.Name + ":" + country.Population);
    }
}
```

Ключевые слова

Ключевое слово	Описание
from	Объявление источника данных и «range variable» (прямого аналога переменной в foreach). Транслируется в имя коллекции и параметры у лямбд
where	Транслируется в Where
select	Транслируется в Select
group	Транслируется в GroupBy вместе с by
into	Транслируется во временную переменную, хранящую результат join, group или select
orderby	Транслируется в OrderBy или OrderByDescending (по умолчанию OrderBy)
join	Транслируется в Join или GroupJoin
let	Объявляет ещё одну «range variable» для итерирования по промежуточным результатам
in	Указывает вторую коллекцию для join
on	Указывает условие «склейки» элементов коллекций в join
equals	Говорит, что должно быть равно в join ... on
by	Указывает проекцию для GroupBy
ascending	Для orderby, указывает порядок сортировки
descending	Для orderby, указывает порядок сортировки

join, пример

```
var innerJoinQuery =
    from category in categories
    join prod in products on category.ID equals prod.CategoryID
    select new { ProductName = prod.Name, Category = category.Name };
```

let и два from, пример

```
string[] strings = [
    "A penny saved is a penny earned.",
    "The early bird catches the worm.",
    "The pen is mightier than the sword." ];

// Split the sentence into an array of words
// and select those whose first letter is a vowel.
var earlyBirdQuery = from sentence in strings
    let words = sentence.Split(' ')
    from word in words
    let w = word.ToLower()
    where w[0] == 'a' || w[0] == 'e' || w[0] == 'i' || w[0] == 'o' || w[0] == 'u'
    select word;

// Execute the query.
foreach (var v in earlyBirdQuery)
    Console.WriteLine($"\"{v}\" starts with a vowel");
```