

Операция GroupBy

[LINQ \(../base/level1/info_linq.php\)](#) --- [LINQ to Objects \(../level1/linq_index.php\)](#) --- Операция GroupBy

Операции группирования помогают объединять вместе элементы последовательности по общему ключу. Операция GroupBy используется для группирования элементов входной последовательности.

Все прототипы операции GroupBy возвращают последовательность элементов **IGrouping<K, T>**. Здесь IGrouping<K, T> - интерфейс, который определен, как показано ниже:

C#

```
public interface IGrouping<K, T> : IEnumerable<T>
{
    K Key { get; }
}
```

Таким образом, IGrouping — это последовательность элементов типа T с ключами типа K. Существуют четыре прототипа, которые описаны ниже:

Первый прототип GroupBy

C#

```
public static IEnumerable<IGrouping<K, T>> GroupBy<T, K>(
    this IEnumerable<T> source,
    Func<T, K> keySelector);
```

Этот прототип операции GroupBy возвращает объект, который при перечислении перебирает входную последовательность source, вызывает метод keySelector, собирает каждый элемент с его ключом и выдает последовательность экземпляров IGrouping<K, E>, где каждый элемент IGrouping<K, E> представляет собой последовательность элементов с одинаковым значением ключа. Значения ключа сравниваются с использованием компаратора эквивалентности по умолчанию — **EqualityComparer.Default**. Говоря иначе, возвращаемое значение метода GroupBy — **Proйди тесты** **C# тест (легкий)** (<https://professorweb.ru/test/c-sharp-test.html>) **.NET тест (средний)** (<https://professorweb.ru/test/asp-test.html>)

это последовательность объектов `IGrouping`, каждый из которых содержит ключ и последовательность элементов из входной последовательности, имеющих тот же ключ.

Порядок экземпляров `IGrouping` будет тем же, что и вхождения ключей в последовательности `source`, и каждый элемент в последовательности `IGrouping` будет расположен в том порядке, в котором элементы находились в последовательности `source`.

Второй прототип GroupBy

C#

```
public static IEnumerable<IGrouping<K, T>> GroupBy<T, K> (
    this IEnumerable<T> source,
    Func<T, K> keySelector,
    IEqualityComparer<K> comparer);
```

Этот прототип операции `GroupBy` похож на первый, за исключением того, что вместо использования компаратора эквивалентности по умолчанию, `EqualityComparer.Default`, указывается собственный компаратор.

Третий прототип GroupBy

C#

```
public static IEnumerable<IGrouping<K, T>> GroupBy<T, K>(
    this IEnumerable<T> source,
    Func<T, K> keySelector,
    Func<T, E> elementSelector);
```

Этот прототип операции `GroupBy` подобен первому, за исключением того, что вместо помещения в выходную последовательность `IGrouping` всего исходного элемента целиком, можно указать, какая часть входного элемента должна попасть на выход, используя для этого `elementSelector`.

Четвертый прототип GroupBy

C#

Пройди тесты

```
public static IEnumerable<IGrouping<K, T>> GroupBy<T, K>(
    this IEnumerable<T> source,
    Func<T, K> keySelector,
    Func<T, E> elementSelector,
    IEqualityComparer<K> comparer);
```

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

Этот прототип операции GroupBy является комбинацией второго и третьего, так что можно указывать компаратор в аргументе comparer и выводить элементы типа, отличного от входных элементов, используя аргумент elementSelector.

В примере применения первого прототипа GroupBy будет использоваться общий класс EmployeeOptionEntry (2_0.php). В коде, приведенном ниже, записи EmployeeOptionEntry группируются по id и затем отображаются:

C#

```
EmployeeOptionEntry[] empOptions = EmployeeOptionEntry.GetEmployeeOptionEntries();
IEnumerable<IGrouping<int, EmployeeOptionEntry>> outerSequence =
    empOptions.GroupBy(o => o.id);

// Сначала перечисление по внешней последовательности IGrouping
foreach (IGrouping<int, EmployeeOptionEntry> keyGroupSequence in outerSequence)
{
    Console.WriteLine("Записи опционов для сотрудника: " + keyGroupSequence.Key);

    // Теперь перечисление по сгруппированной последовательности элементов EmployeeOptionEntry.
    foreach (EmployeeOptionEntry element in keyGroupSequence)
        Console.WriteLine("id={0} : optionsCount={1} : dateAwarded={2:d}",
            element.id, element.optionsCount, element.dateAwarded);
}
```

Обратите внимание, что перечисление выполняется по внешней последовательности с именем outerSequence, где каждый элемент — это объект, реализующий IGrouping, который содержит ключ, и последовательность элементов EmployeeOptionEntry, имеющих одинаковые ключи. Ниже показаны результаты:

```
file:///D:/My_C#/LINQ/Pro LINQ - All Source Code/LINQChapter4/LINQChapter4/bin/Debug/L...
Записи опционов для сотрудника: 1
id=1 : optionsCount=2 : dateAwarded=31.12.1999
Записи опционов для сотрудника: 2
id=2 : optionsCount=10000 : dateAwarded=30.06.1992
id=2 : optionsCount=10000 : dateAwarded=01.01.1994
id=2 : optionsCount=10000 : dateAwarded=01.04.2003
Записи опционов для сотрудника: 3
id=3 : optionsCount=5000 : dateAwarded=30.09.1997
id=3 : optionsCount=7500 : dateAwarded=30.09.1998
id=3 : optionsCount=7500 : dateAwarded=30.09.1998
Записи опционов для сотрудника: 4
id=4 : optionsCount=1500 : dateAwarded=31.12.1997
Записи опционов для сотрудника: 101
id=101 : optionsCount=2 : dateAwarded=31.12.1998
```

Пройди тесты

Для примера использования LINQ (http://professorweb.ru/base/level2/2_0.php) предположим, что каждый сотрудник, id которого меньше 100, является одним из основателей компании. NET тест (средний) (https://professorweb.ru/test/asp-test.html) Те, у кого id равен 100 и больше, основателями не являются. Задача состоит в том,

чтобы вывести все записи опционов, сгруппированные по статусу сотрудника. Все опционы основателей будут сгруппированы вместе, а отдельно от них — опционы сотрудников, не являющихся основателями.

Теперь необходим компаратор эквивалентности, который сможет выполнить это ключевое сравнение. Этот компаратор должен реализовывать **интерфейс IEqualityComparer**. Прежде чем рассматривать сам компаратор, давайте сначала взглянем на интерфейс:

C#

```
interface IEqualityComparer<T> {  
    bool Equals (T x, T y);  
    int GetHashCode(T x); }
```

Этот интерфейс требует реализации двух методов — Equals и GetHashCode. **Метод Equals** принимает два объекта одного типа T и возвращает true, если два объекта считаются эквивалентными, и false — в противном случае. **Метод GetHashCode** принимает единственный объект и возвращает хеш-код типа int для этого объекта.

Хеш-код — это числовое значение, обычно вычисляемое математически на основе некоторой части данных объекта, известной как ключ, в целях уникальной идентификации объекта. Функция вычисляемого хеш-кода состоит в том, чтобы служить индексом в некоторой структуре данных для хранения объекта и последующего его нахождения. Поскольку допускается, что множество ключей производят один и тот же хеш-код, что делает его не уникальным, также есть необходимость в определении эквивалентности двух ключей. В этом предназначение метода Equals.

Ниже приведен код класса, реализующего интерфейс IEqualityComparer:

C#

Пройди тесты

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

```
public class MyFounderNumberComparer : IEqualityComparer<int>
{
    public bool Equals(int x, int y)
    {
        return (isFounder(x) == isFounder(y));
    }

    public int GetHashCode(int i)
    {
        int f = 1;
        int nf = 100;
        return (isFounder(i) ? f.GetHashCode() : nf.GetHashCode());
    }

    public bool isFounder(int id)
    {
        return (id < 100);
    }
}
```

В дополнение к методам, которые требует интерфейс, был добавлен метод `isFounder` для определения того, является ли сотрудник основателем компании на базе приведенного выше определения. Это немного прояснит код. Этот метод сделан общедоступным, чтобы его можно было вызывать извне интерфейса, что и будет осуществляться в примере.

Компаратор эквивалентности рассматривает любой целочисленный идентификатор сотрудника, который меньше 100, как признак основателя компании, и если два идентификатора указывают на то, что оба являются основателями или оба основателями не являются, они считаются эквивалентными. Для основателей возвращается хеш-код 1, а для не основателей — 100, так что все основатели попадают в одну группу, а прочие — в другую.

Пример использования `GroupBy` приведен ниже:

C#

Пройди тесты

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

```

MyFounderNumberComparer comp = new MyFounderNumberComparer();

EmployeeOptionEntry[] empOptions = EmployeeOptionEntry.GetEmployeeOptionEntries();
IEnumerable<IGrouping<int, EmployeeOptionEntry>> opts = empOptions
    .GroupBy(o => o.id, comp);

// Сначала перечисление по последовательности IGrouping
foreach (IGrouping<int, EmployeeOptionEntry> keyGroup in opts)
{
    Console.WriteLine("Записи опционов для: " +
        (comp.isFounder(keyGroup.Key) ? "основатель" : "работяга"));

    // Теперь перечисление по сгруппированной последовательности
    // элементов EmployeeOptionEntry
    foreach (EmployeeOptionEntry element in keyGroup)
        Console.WriteLine("id={0} : optionsCount={1} : dateAwarded={2:d}",
            element.id, element.optionsCount, element.dateAwarded);
}

```

В этом примере экземпляр компаратора эквивалентности создается заблаговременно — в противоположность тому, чтобы делать это в вызове метода GroupBy — так что его можно использовать для вызова метода isFounder в цикле foreach. Ниже показан результат работы этого кода:

```

Записи опционов для: основатель
id=1 : optionsCount=2 : dateAwarded=31.12.1999
id=2 : optionsCount=10000 : dateAwarded=30.06.1992
id=2 : optionsCount=10000 : dateAwarded=01.01.1994
id=3 : optionsCount=5000 : dateAwarded=30.09.1997
id=2 : optionsCount=10000 : dateAwarded=01.04.2003
id=3 : optionsCount=7500 : dateAwarded=30.09.1998
id=3 : optionsCount=7500 : dateAwarded=30.09.1998
id=4 : optionsCount=1500 : dateAwarded=31.12.1997

Записи опционов для: работяга
id=101 : optionsCount=2 : dateAwarded=31.12.1998

```

Как видите, все записи опционов сотрудников, id которых меньше 100, попадают в группу основателей. В противном случае они попадают в группу работяг :)

Для примера применения третьего прототипа GroupBy предположим, что нас интересуют даты назначения опционов для каждого сотрудника. Этот код будет очень похож на код примера первого прототипа. Итак, ниже вместо возврата последовательности сгруппированных объектов EmployeeOptionEntry будут возвращаться даты:

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

C# .NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

```

EmployeeOptionEntry[] empOptions = EmployeeOptionEntry.GetEmployeeOptionEntries();
IEnumerable<IGrouping<int, DateTime>> opts = empOptions
    .GroupBy(o => o.id, e => e.dateAwarded);

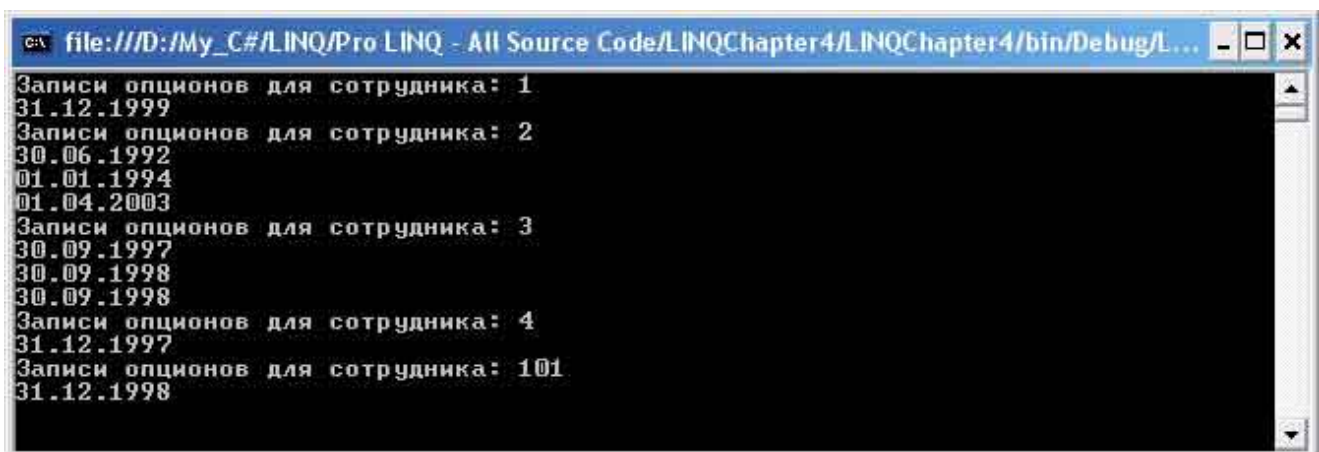
// Сначала перечисление по последовательности IGrouping
foreach (IGrouping<int, DateTime> keyGroup in opts)
{
    Console.WriteLine("Записи опционов для сотрудника: " + keyGroup.Key);

    // Теперь перечисление по сгруппированной последовательности элементов DateTime
    foreach (DateTime date in keyGroup)
        Console.WriteLine(date.ToShortDateString());
}

```

Обратите внимание, что в вызове операции GroupBy второй аргумент elementSelect просто возвращает член dateAwarded. Поскольку возвращается DateTime, интерфейс IGrouping теперь служит для типа DateTime вместо EmployeeOptionEntry.

Как и можно было ожидать, выводятся даты назначений опционов, сгруппированные по сотрудникам:



```

file:///D:/My_C#/LINQ/Pro LINQ - All Source Code/LINQChapter4/LINQChapter4/bin/Debug/L...
Записи опционов для сотрудника: 1
31.12.1999
Записи опционов для сотрудника: 2
30.06.1992
01.01.1994
01.04.2003
Записи опционов для сотрудника: 3
30.09.1997
30.09.1998
30.09.1998
Записи опционов для сотрудника: 4
31.12.1997
Записи опционов для сотрудника: 101
31.12.1998

```

Назад (2_7.php)	7	8	9	Вперед (2_9.php)
-----------------	---	---	---	------------------

Пройди тесты



Лучший чат для C# программистов (<https://t.me/professorweb>)

Professor Web (/)

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)