

Операции Select и SelectMany

[LINQ \(../base/level1/info_linq.php\)](#) --- [LINQ to Objects \(../level1/linq_index.php\)](#) --- Операции Select и SelectMany

Select

Операция **Select** используется для создания выходной последовательности одного типа элементов из входной последовательности элементов другого типа. Эти типы не обязательно должны совпадать.

Существуют два прототипа этой операции, которые описаны ниже:

Первый прототип Select

C#

```
public static IEnumerable<S> Select<T, S>(
    this IEnumerable<T> source,
    Func<T, S> selector);
```

Этот прототип **Select** принимает входную последовательность и делегат метода-селектора в качестве входных параметров, а возвращает объект, который при перечислении проходит по входной последовательности и выдает последовательность элементов типа *S*. Как упоминалось ранее, *T* и *S* могут быть как одного, так и разных типов.

При вызове **Select** делегат метода-селектора передается в аргументе *selector*. Метод-селектор должен принимать тип *T* в качестве входного, где *T* — тип элементов, содержащихся во входной последовательности, и возвращать элемент типа *S*. Операция **Select** вызовет метод-селектор для каждого элемента входной последовательности, передав ему этот элемент. Метод-селектор выберет интересующую часть входного элемента, создаст новый элемент — возможно, другого типа (даже анонимного) — и вернет его.

Второй прототип Select

Пройди тесты

C#

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

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

```
public static IEnumerable<S> Select<T, S>(
    this IEnumerable<T> source,
    Func<T, int, S> selector);
```

В этом прототипе операции Select методу-селектору передается дополнительный целочисленный параметр. Это индекс, начинающийся с нуля, входного элемента во входной последовательности.

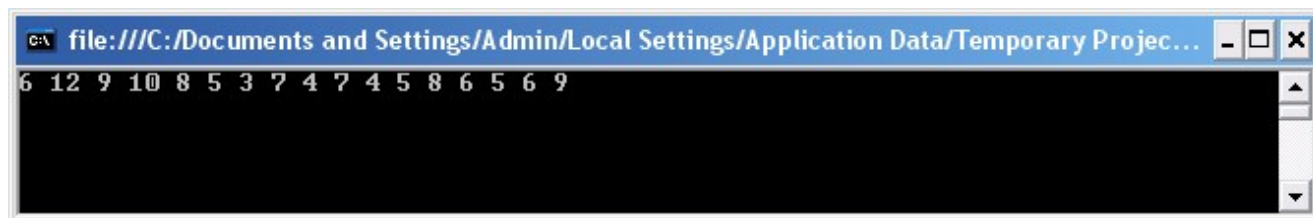
Пример вызова первого прототипа показан ниже:

C#

```
string[] cars = { "Nissan", "Aston Martin", "Chevrolet", "Alfa Romeo", "Chrysler", "Dodge",
    "BMW",
    "Ferrari", "Audi", "Bentley", "Ford", "Lexus", "Mercedes", "Toyota", "Volvo", "Subaru", "Жигули :)"};

IEnumerable<int> sequence = cars.Select(p => p.Length);

foreach (int i in sequence)
    Console.Write(i + " ");
```



Обратите внимание, что метод-селектор передается через лямбда-выражение. В данном случае лямбда-выражение вернет длину каждого элемента из входной последовательности. Также отметьте, что хотя тип входных элементов — строка, тип выходных элементов — int.

Это простой пример, потому что никакого класса не генерируется. Ниже приведена более интересная демонстрация использования первого прототипа:

C#

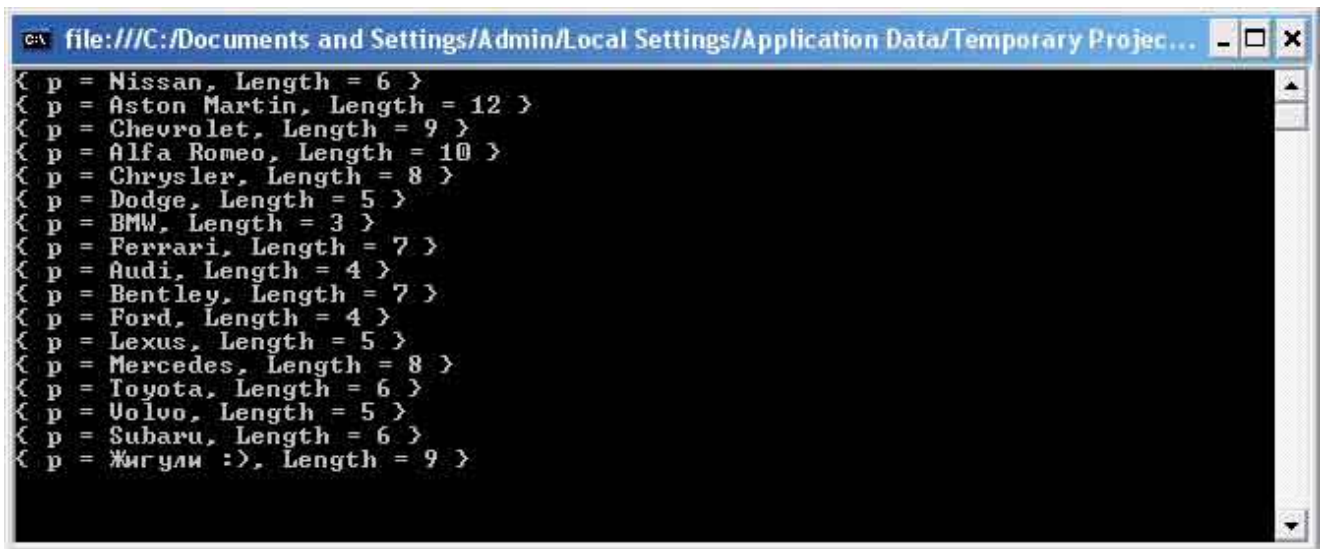
Пройди тесты

```
var sequence = cars.Select(p => new { p, p.Length });
```

```
foreach (var i in sequence)
    Console.WriteLine(i);
```

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

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



```
file:///C:/Documents and Settings/Admin/Local Settings/Application Data/Temporary Projec...
{ p = Nissan, Length = 6 }
{ p = Aston Martin, Length = 12 }
{ p = Chevrolet, Length = 9 }
{ p = Alfa Romeo, Length = 10 }
{ p = Chrysler, Length = 8 }
{ p = Dodge, Length = 5 }
{ p = BMW, Length = 3 }
{ p = Ferrari, Length = 7 }
{ p = Audi, Length = 4 }
{ p = Bentley, Length = 7 }
{ p = Ford, Length = 4 }
{ p = Lexus, Length = 5 }
{ p = Mercedes, Length = 8 }
{ p = Toyota, Length = 6 }
{ p = Volvo, Length = 5 }
{ p = Subaru, Length = 6 }
{ p = Жигули :), Length = 9 }
```

Обратите внимание, что лямбда-выражение создает экземпляр нового анонимного типа. Компилятор динамически сгенерирует анонимный тип, который будет содержать `string p` и `int p.Length`, и метод-селектор вернет этот вновь созданный объект. Поскольку тип возвращаемого элемента является анонимным, сослаться на него нет возможности. Поэтому присвоить выходную последовательность `Select` экземпляру `IEnumerable` какого-то известного типа нельзя, как это делалось в первом примере, где она присваивалась переменной типа `IEnumerable<int>`, представляющей выходную последовательность. Поэтому выходная последовательность присваивается переменной, указанной с помощью ключевого слова `var`.

С этим кодом связана одна проблема: управлять именами членов динамически сгенерированного анонимного класса нельзя. Однако, благодаря средству инициализации объектов C#, можно написать лямбда-выражение и задать имена членов анонимного класса, как показано ниже:

C#

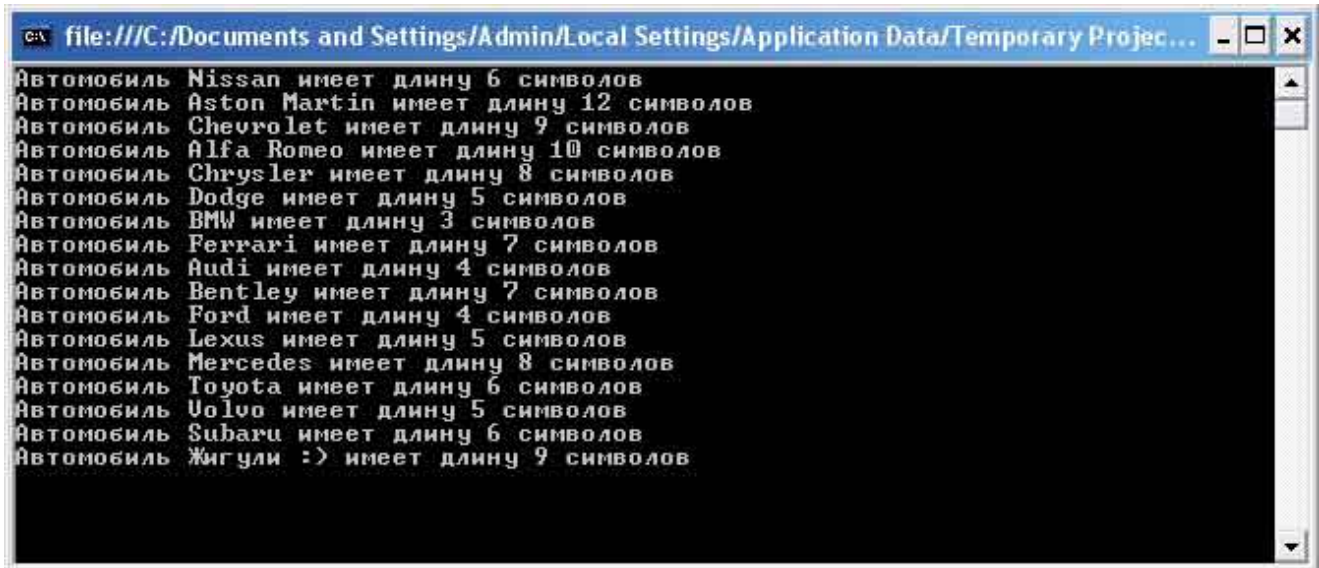
```
var carObj = cars.Select(p => new { LastName = p, Length = p.Length });

foreach (var i in carObj)
    Console.WriteLine("Автомобиль {0} имеет длину {1} символов", i.LastName, i.Length);
```

Пройди тесты

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

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



Для примера второго прототипа будет добавлен индекс, который передается методу-селектору, в тип элемента выходной последовательности:

C#

```
var carObj = cars.Select((p, i) => new { Index = i + 1, LastName = p });

foreach (var i in carObj)
    Console.WriteLine( i.Index + ". " + i.LastName);
```

Этот пример выводит номер индекса плюс единица, за которым следует имя. Код производит следующий результат:



Пройди тесты

SelectMany

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

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

Операция SelectMany используется для создания выходной последовательности с проекцией "один ко многим" из входной последовательности. В то время как операция Select возвращает один выходной элемент для каждого входного элемента, SelectMany вернет ноль или более выходных элементов для каждого входного.

Существуют два прототипа этой операции, которые рассматриваются ниже:

Первый прототип SelectMany

C#

```
public static IEnumerable<S> SelectMany<T, S>(
    this IEnumerable<T> source,
    Func<T, IEnumerable<S>> selector);
```

Этот прототип операции получает входную последовательность элементов типа T и делегат метода-селектора, а возвращает объект, который при перечислении проходит по входной последовательности, получая каждый элемент индивидуально из входной последовательности и передавая его в метод-селектор. Последний затем возвращает объект, который во время перечисления выдает ноль или более элементов типа S в промежуточную выходную последовательность. Операция SelectMany вернет конкатенированную выходную последовательность при каждом вызове метода-селектора.

Второй прототип SelectMany

C#

```
public static IEnumerable<S> SelectMany<T, S>(
    this IEnumerable<T> source,
    Func<T, int, IEnumerable<S>> selector);
```

Этот прототип ведет себя так же, как и первый, за исключением того, что методу-селектору дополнительно передается индекс, начинающийся с нуля, каждого элемента входной последовательности.

Ниже показан пример вызова первого прототипа:

C#

Пройди тесты

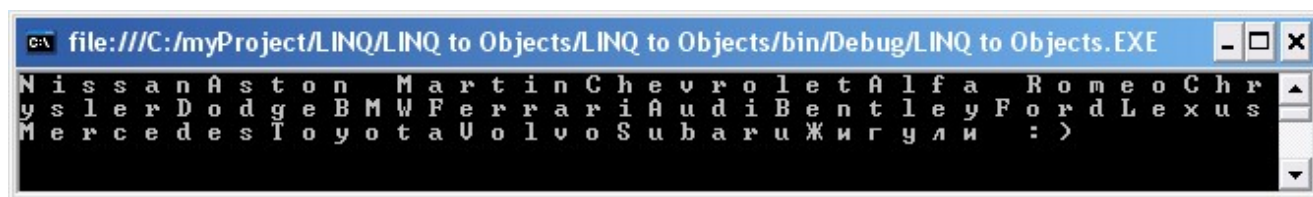
C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

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

```
string[] cars = { "Nissan", "Aston Martin", "Chevrolet", "Alfa Romeo", "Chrysler", "Dodge",  
"BMW",  
                "Ferrari", "Audi", "Bentley", "Ford", "Lexus", "Mercedes", "Toyota", "Volvo", "Subaru", "Жигули :)"};  
  
IEnumerable<char> chars = cars.SelectMany(p => p.ToArray());  
  
foreach (char c in chars)  
    Console.Write(c + " ");
```

В предыдущем примере метод-селектор принимает на входе строку и, вызывая метод `ToArray` на этой строке, возвращает массив символов, который становится выходной последовательностью типа `char`.

Таким образом, для одного элемента входной последовательности, которым в данном случае является экземпляр `string`, метод-селектор возвращает последовательность символов. Для каждой входной строки выводится последовательность `char`. Операция `SelectMany` соединяет все эти последовательности символов в единую последовательность, которую и возвращает. Вывод предыдущего кода выглядит следующим образом:



Это достаточно простой запрос, но не слишком демонстративный в смысле типового применения. В следующем примере будут использоваться общие классы `Employee` и `EmployeeOptionEntry` (2_0.php).

На массиве элементов `Employee` будет вызвана операция `SelectMany`, и для каждого элемента `Employee` в массиве делегат метода-селектора вернет ноль или более элементов созданного анонимного класса, содержащего `id` и `optionsCount` из массива элементов `EmployeeOptionEntry` для объекта `Employee`:

C#

Пройди тесты

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

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

```

Employee[] employees = Employee.GetEmployeesArray();
EmployeeOptionEntry[] empOptions = EmployeeOptionEntry.GetEmployeeOptionEntries();

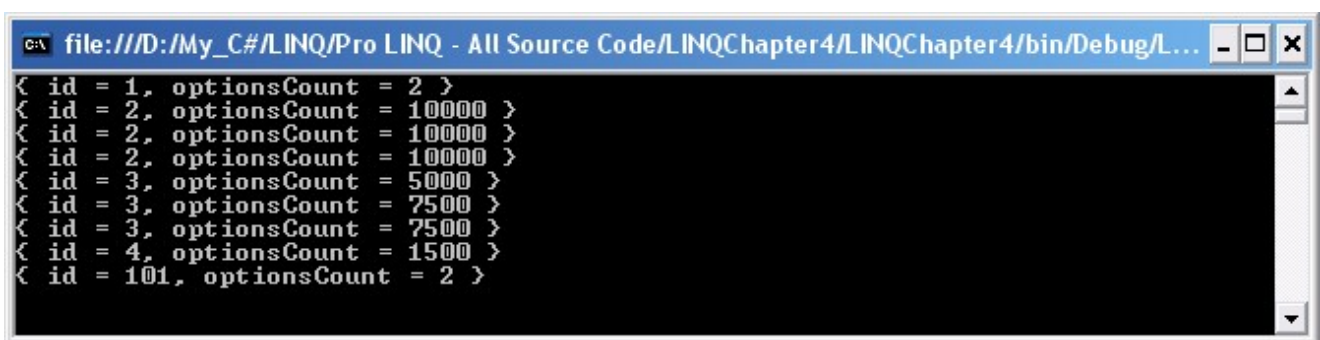
var employeeOptions = employees
    .SelectMany(e => empOptions
        .Where(eo => eo.id == e.id)
        .Select(eo => new
        {
            id = eo.id,
            optionsCount = eo.optionsCount
        }));

foreach (var item in employeeOptions)
    Console.WriteLine(item);

```

В этом примере каждый сотрудник в массиве `Employee` передается в лямбда-выражение, в свою очередь, переданное операции `SelectMany`. Это лямбда-выражение затем извлечет каждый элемент `EmployeeOptionEntry`, чей `id` соответствует `id` текущего сотрудника, переданного ему посредством операции `Where`. Это эффективно связывает массив `Employee` с массивом `EmployeeOptionEntry` по их членам `id`. Операция `Select` лямбда-выражения затем создает анонимный объект, содержащий члены `id` и `optionsCount` для каждой соответствующей записи в массиве `EmployeeOptionEntry`. Это значит, что лямбда-выражение возвращает последовательность из нуля или более анонимных объектов для каждого переданного сотрудника. Это дает в результате последовательность или последовательности, которые операция `SelectMany` может соединить вместе.

Предыдущий код даст такой вывод:



```

C:\ file:///D:/My_C#/LINQ/Pro LINQ - All Source Code/LINQChapter4/LINQChapter4/bin/Debug/L...
< id = 1, optionsCount = 2 >
< id = 2, optionsCount = 10000 >
< id = 2, optionsCount = 10000 >
< id = 2, optionsCount = 10000 >
< id = 3, optionsCount = 5000 >
< id = 3, optionsCount = 7500 >
< id = 3, optionsCount = 7500 >
< id = 4, optionsCount = 1500 >
< id = 101, optionsCount = 2 >

```

Имейте в виду, что это лямбда-выражение не особо эффективно, если входных элементов много. Дело в том, что лямбда-выражение вызывается для каждого входного элемента. После обработки первых пяти входных элементов просто возвращается пустой массив. Для повышения производительности необходимо отдать предпочтение операции `Take`.

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

Операция `SelectMany` также полезна для соединения множества последовательностей в одну.

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

Назад (2_1.php)	1	2	3	Вперед (2_3.php)
-----------------	---	---	---	------------------



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

Professor Web (/)

Наш любимый хостинг (/)

Пройди тесты

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

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