Запросы LINQ

Одним из привлекательных для разработчиков средств LINQ является SQL-подобный синтаксис, доступный в LINQ-запросах. Этот синтаксис использовался в нескольких примерах LINQ, приведенных ранее. Синтаксис предоставлен через расширение языка C#, которое называется **выражения запросов**. Выражения запросов позволяют запросам LINQ принимать форму, подобную SQL, всего лишь с рядом небольших отличий.

Для выполнения запроса LINQ выражения запросов не обязательны. Альтернативой является использование стандартной точечной нотации C# с вызовом методов на объектах и классах. Во многих случаях применение стандартной точечной нотации оказывается более предпочтительным, поскольку она более наглядно демонстрирует, что в действительности происходит и когда.

При записи запроса в стандартной точечной нотации не происходит никакой трансформации при компиляции. Именно поэтому во многих примерах не используется синтаксис выражений запросов, а предпочтение отдается стандартному синтаксису точечной нотации. Однако привлекательность синтаксиса выражений запросов бесспорна. Впечатление чего-то знакомого, которое он производит, когда вы формулируете свой первый запрос, может оказаться весьма привлекательным.

Получить представление о различиях между этими двумя синтаксисами лучше всего на примере:

C#

Пройди тесты

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

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

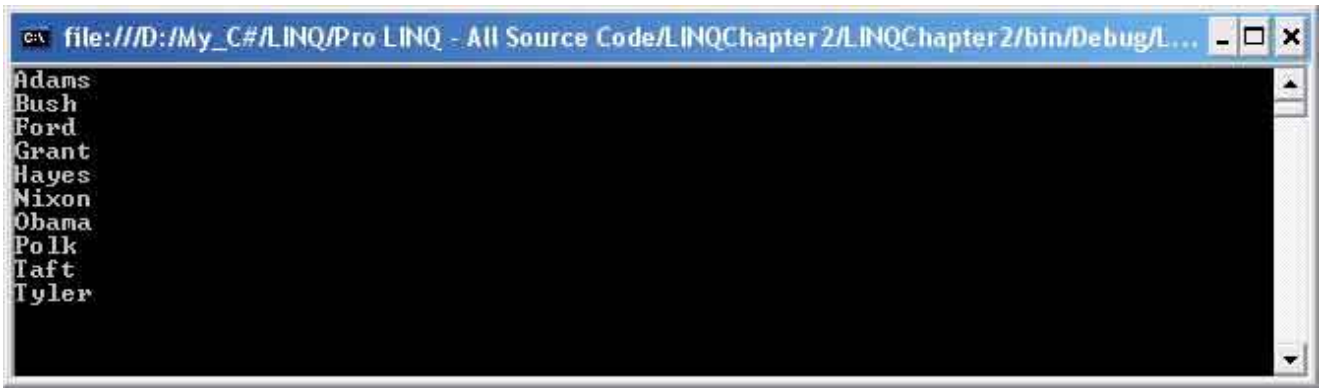
```
string[] names = {
        "Adams", "Arthur", "Buchanan", "Bush", "Carter", "Cleveland",
        "Clinton", "Coolidge", "Eisenhower", "Fillmore", "Ford",  "Garfield",
        "Grant", "Harding", "Harrison", "Hayes", "Hoover", "Jackson",
        "Jefferson", "Johnson", "Kennedy", "Lincoln", "Madison", "McKinley",
        "Monroe", "Nixon", "Obama", "Pierce", "Polk", "Reagan", "Roosevelt",
        "Taft", "Taylor", "Truman", "Tyler", "Van Buren", "Washington", "Wilson"};

      // Использование точечной нотации
      IEnumerable<string> sequence = names
        .Where(n => n.Length < 6)
        .Select(n => n);

      // Использование синтаксиса выражения запроса
      IEnumerable<string> sequence = from n in names
                                     where n.Length < 6
                                     select n;

      foreach (string name in sequence)
      {
        Console.WriteLine("{0}", name);
      }
```

Первое, что можно заметить в примере с выражением запроса — это то, что в отличие от SQL, операция from предшествует операции select. Одной из причин этого была необходимость сужения контекста для средства IntelliSense. Без такой инверсии операций ввод в текстовом редакторе Visual Studio слова "select", за которым следует пробел, не позволит средству IntelliSense определить, какие переменные должны отображаться в его раскрывающемся списке. Контекст допустимых переменных в этой точке ничем не ограничен. Если же сначала указать, откуда поступают данные, то IntelliSense получает контекст и может предоставить список переменных для выбора. Приведенный выше код дает следующий результат:



Пройди тесты

Важно отметить, что синтаксис выражений запросов поддерживается только для наиболее распространенных операций запросов: Where, Select, SelectMany, Join, GroupJoin, GroupBy, OrderBy, ThenBy, OrderByDescending и ThenByDescending.

Грамматика выражений запросов

Выражения запросов должны подчиняться перечисленным ниже правилам:

Выражение должно начинаться с конструкции from.

Остальная часть выражения может содержать ноль или более конструкций from, let или where. Конструкция from — это генератор, который объявляет одну или более переменных диапазона, перечисляющих последовательность или соединение нескольких последовательностей. Конструкция let представляет переменную диапазона и присваивает ей значение. Конструкция where фильтрует элементы из входной последовательности или соединения несколько входных последовательностей в выходную последовательность.

Остальная часть выражения запроса может затем включать конструкцию orderby, содержащую одно или более полей сортировки с необязательным направлением упорядочивания. Направлением может быть *ascending (по возрастанию)* или *descending (по убыванию)*.

Затем в оставшейся части выражения может идти конструкция select или group.

Наконец в оставшейся части выражения может следовать необязательная конструкция продолжения. Такой конструкцией может быть либо into, ноль или более конструкций join, или же другая повторяющаяся последовательность перечисленных элементов, начиная с конструкций из правила 2. Конструкция into направляет результаты запроса в воображаемую выходную последовательность, которая служит конструкцией from для последующих выражения запросов, начиная с конструкций из правила 2.

Трансляция выражений запросов

Предположим, что создано синтаксически корректное выражение запроса. Следующая сложность связана с тем, как компилятор транслирует это выражение запроса в код C#. Он должен транслировать выражение в стандартную точечную нотацию C#, о которой говорилось ранее. Но как это делается?

Чтобы транслировать выражение запроса, компилятор ищет в нем шаблоны кода (code patterns). Компилятор выполняет несколько шагов трансляции в определенном порядке, чтобы превратить выражение запроса в стандартную нотацию C#. На каждом из этих шагов ищется один или более взаимосвязанных шаблонов кода. Прежде чем перейти к следующему шагу, компилятор должен многократно

транслировать все вхождения шаблонов кода для данного шага трансляции. На каждом шаге он исходит из предположения, что все шаблоны кода для всех предыдущих шагов уже транслированы.

Прозрачные идентификаторы

Некоторые трансляции вставляют переменные перечислений с прозрачными идентификаторами. На шаге трансляции, описанном в следующем разделе, прозрачный идентификатор обозначен звездочкой (*). Его не нужно путать с символом * — шаблоном полей для выбора в SQL. При трансляции выражения запроса иногда компилятором генерируются дополнительные перечисления, и прозрачные идентификаторы используются для прохода по ним. Прозрачные идентификаторы существуют только во время процесса трансляции, и как только выражение запроса полностью транслировано, никаких прозрачных идентификаторов в запросе не остается.

Шаги трансляции

Теперь рассмотрим шаги трансляции. При этом для представления определенных частей запроса используются буквы переменных, описанные ниже:

Переменные трансляции LINQ

Переменная	Описание	Пример
c	Сгенерированная компилятором временная переменная	-
e	Переменная диапазона	from e in s
s	Входная последовательность	from e in s
f	Выбранный элемент-поле или новый анонимный тип	from e in s select f
g	Групповой элемен	from e in s group g by k
i	Воображаемая последовательность into	from e in s select f into i

k	Групповой или объединенный ключевой элемент	from e in s group g by k
l	Переменная, представленная let	from e in s let l = v
o	Упорядочивающий элемент	from e in s orderby o
v	Значение, присвоенное переменной let	from e in s let l = v
w	Конструкция where	from e in s where w

Здесь необходимо высказать одно предостережение. Описание шагов трансляции может показаться довольно сложным. Пусть это вас не расстраивает. Для написания запросов LINQ полностью понимать все шаги трансляции не обязательно. Они приводятся здесь лишь для того, чтобы предоставить дополнительную информацию о трансляции на тот случай, если вдруг она понадобится, что случается редко, а может быть и никогда.

Назад (1_4.php)	4	5	6	Вперед (1_6.php)

Пройди тесты

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

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