using System.Linq;

Аббревиатура LINQ означает Language-Integrated Query, т.е. *язык интегрированных запросов.* Это понятие охватывает ряд средств, позволяющих извлекать информацию из источника данных. Как вам должно быть известно, извлечение данных составляет важную часть многих программ. Например, программа может получать информацию из списка заказчиков, искать информацию в каталоге продукции или получать доступ к учетному документу, заведенному на работника. Как правило, такая информация хранится в базе

данных, существующей отдельно от приложения.

LINQ дополняет C# средствами, позволяющими формировать запросы для любого LINQ-совместимого источника данных. При этом синтаксис, используемый для формирования запросов, остается неизменным,

независимо от типа источника данных. Это, в частности, означает, что синтаксис,

требующийся для формирования запроса к реляционной базе данных, практически

ничем не отличается от синтаксиса запроса данных, хранящихся в массиве.

Для этой цели теперь не нужно прибегать к средствам SQL или другого внешнего по

отношению к C# механизма извлечения данных из источника. Возможности формировать

запросы отныне полностью интегрированы в язык С#.

Помимо SQL, LINQ можно использовать вместе с XML-файлами и наборами данных

ADO.NET Dataset. Не менее важным является применение LINQ вместе с массивами

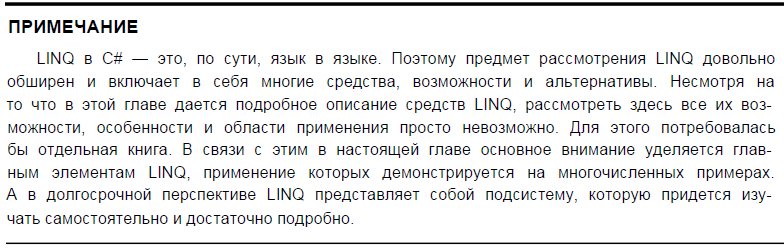
и коллекциями в C#.

LINQ поддерживается целым рядом взаимосвязанных средств, включая внедренный

в C# синтаксис запросов, лямбда-выражения, анонимные типы и методы расширения.

О лямбда-выражениях речь уже шла в главе 15, а остальные средства рассматриваются

в этой главе.



**Основы LINQ**

В основу LINQ положено понятие *запроса,* в котором определяется информация,

получаемая из источника данных. И хотя все эти запросы отличаются в деталях, их можно выразить, используя одни и те же синтаксические элементы LINQ.

Как только запрос будет сформирован, его можно выполнить. Это делается, в частности,

в цикле foreach. В результате выполнения запроса выводятся его результаты.

Поэтому использование запроса может быть разделено на две главные стадии. На первой

стадии запрос формируется, а на второй — выполняется. Таким образом, при формировании

запроса определяется, что *именно* следует извлечь из источника данных.

А при выполнении запроса выводятся конкретные *результаты.*

Для обращения к источнику данных по запросу, сформированному средствами

LINQ, в этом источнике должен быть реализован интерфейс IEnumerable. Он имеет

две формы: обобщенную и необобщенную. Как правило, работать с источником

данных легче, если в нем реализуется обобщенная форма IEnumerable<T>, где Т обозначает

обобщенный тип перечисляемых данных. Здесь и далее предполагается, что в

источнике данных реализуется форма интерфейса IEnumerable<T>. Этот интерфейс

объявляется в пространстве имен System.Collections.Generic. Класс, в котором

реализуется форма интерфейса IEnumerable<T>, поддерживает перечисление, а это

означает, что его содержимое может быть получено по очереди или в определенном

порядке. Форма интерфейса IEnumerable<T> поддерживается всеми массивами в С#.

Поэтому на примере массивов можно наглядно продемонстрировать основные принципы

работы LINQ. Следует, однако, иметь в виду, что применение LINQ не ограничивается

одними массивами.

**Простой запрос**

А теперь самое время обратиться к простому примеру использования LINQ. В приведенной

ниже программе используется запрос для получения положительных значений,

содержащихся в массиве целых значений.

(***glava19\_1***)

class SimpleQuery

{

static void Main()

{

int[] nums = { 1, -2, 3, 0, -4, 5 };

//simple query to get only positive nums

var posNums = from n in nums

where n > 0

select n;

Console.Write("Positive values from nums: ");

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

Console.WriteLine();

}

}

Как видите, в конечном итоге отображаются только положительные значения, хранящиеся

в массиве nums. Несмотря на всю свою простоту, этот пример наглядно демонстрирует

основные возможности LINQ. Поэтому рассмотрим его более подробно.

Для применения средств LINQ в исходный текст программы следует включить пространство

имен System.Linq.

Затем в программе объявляется массив nums типа int. Все массивы в C# неявным

образом преобразуются в форму интерфейса IEnumerable<T>. Благодаря этому любой

массив в C# может служить в качестве источника данных, извлекаемых по запросу

LINQ.

Далее объявляется запрос, по которому из массива nums извлекаются элементы

только с положительными значениями.

//simple query to get only positive nums

var posNums = from n in nums

where n > 0

select n;

Переменная posNums называется *переменной запроса.* В ней хранится ссылка на ряд

правил, определяемых в запросе. Обратите внимание на применение ключевого слова

var для объявления переменной posNums неявным образом. Как вам должно быть

уже известно, благодаря этому переменная posNums становится неявно типизированной.

Такими переменными удобно пользоваться в запросах, хотя их тип можно объявить

и явным образом (это должна быть одна из форм интерфейса IEnumerable<T>).

Объявляемой переменной posNums в итоге присваивается выражение запроса.

Все запросы начинаются с оператора from, определяющего два элемента. Первым

из них является *переменная диапазона,* принимающая элементы из источника данных.

В рассматриваемом здесь примере эту роль выполняет переменная n. Вторым элементом

является источник данных (в данном случае — массив nums). Тип переменной диапазона

выводится из источника данных. Поэтому переменная n относится к типу int.

Ниже приведена общая форма оператора from.

from переменная\_диапазона in источник\_данных

Далее следует оператор where, обозначающий условие, которому должен удовлетворять

элемент в источнике данных, чтобы его можно было получить по запросу.

Ниже приведена общая форма синтаксиса оператора where.

Where булево\_выражение

В этой форме *булево\_выражение* должно давать результат типа bool. Такое выражение

иначе называется *предикатом.* В запросе можно указывать несколько операторов

where.

Иными словами, оператор where выполняет роль своеобразного фильтра, отбирая лишь определенные элементы.

Все запросы оканчиваются оператором select или group. В данном примере

используется оператор select, точно определяющий, что именно должно быть получено

по запросу. В таких простых примерах запросов, как рассматриваемый здесь,

выбирается конкретное значение диапазона. Поэтому по данному запросу возвращаются

только те целые значения, которые удовлетворяют условию, указанному в операторе

where. В более сложных запросах можно дополнительно уточнять, что именно

следует выбирать. Например, по запросу списка рассылки может быть получена лишь

фамилия адресата вместо его полного адреса. Обратите внимание на то, что оператор

select завершается точкой с запятой, поскольку это последний оператор в запросе.

А другие его операторы не оканчиваются точкой с запятой.

Для выполнения запроса в данном примере программы организуется следующий

цикл.

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

В этом цикле переменная posNums указывается в качестве коллекции, к которой

происходит обращение на каждом шаге цикла. В цикле foreach соблюдаются правила,

определенные в запросе и доступные по ссылке из переменной posNums. На каждом

шаге цикла возвращается очередной элемент, полученный из массива.

В данном примере тип int переменной шага цикла i указывается явно, поскольку по

запросу извлекаются элементы именно этого типа. Явное указание типа переменной

шага цикла вполне допустимо в тех случаях, когда заранее известен тип значения, выбираемого

по запросу. Но в более сложных случаях оказывается проще, а иногда даже

нужно, указывать тип переменной шага цикла неявным образом с помощью ключевого

слова var.

**Неоднократное выполнение запросов**

Итак, в запросе определяются правила, по которым извлекаются данные, но этого

явно недостаточно для получения результатов, поскольку запрос должен быть выполнен,

причем это может быть сделано несколько раз. Если же в промежутке между последовательно

производимыми попытками выполнить один и тот же запрос источник

данных изменяется, то получаемые результаты могут отличаться. Ниже приведен другой вариант рассматриваемой здесь программы, где содержимое массива nums изменяется в промежутке

между двумя последовательно производимыми попытками выполнить один и тот же

запрос, хранящийся в переменной posNums.

(***glava19\_2***)

class SimpQuery

{

static void Main()

{

int[] nums = { 1, -2, 3, 0, -4, 5 };

//simple quaery only pos nums

var posNums = from n in nums

where n > 0

select n;

Console.Write("Positive numbers from nums: ");

//make a task and show results

foreach (var i in posNums) Console.Write(i + " ");

Console.WriteLine();

//make some change in nums

Console.WriteLine("\nSet value 99 for nums[1]\n");

nums[1] = 99;

Console.Write("Positive values from nums after changes: ");

foreach (var i in posNums) Console.Write(i + " ");

Console.WriteLine();

}

}

Как следует из результата выполнения приведенной выше программы, значение

элемента массива nums[1] изменилось с -2 на 99, что и отражают результаты повторного

выполнения запроса. Этот важный момент следует подчеркнуть особо. Каждая

попытка выполнить запрос приносит свои результаты, получаемые при перечислении

текущего содержимого источника данных. Поэтому если источник данных претерпевает

изменения, то могут измениться и результаты выполнения запроса. Преимущества

такого подхода к обработке запросов весьма значительны. Так, если по запросу получается

список необработанных заказов в Интернет-магазине, то при каждой попытке

выполнить запрос желательно получить сведения обо всех заказах, включая и только

что введенные.

**Связь между типами данных в запросе**

Тип переменной диапазона должен соответствовать типу элементов, хранящихся

в источнике данных. Следовательно, тип переменной диапазона зависит от типа

источника данных. Как правило, тип переменной диапазона может быть выведен

средствами С#. Но выводимость типов может быть осуществлена при условии, что

в источнике данных реализована форма интерфейса IEnumerable<T>, где Т обозначает

тип элементов в источнике данных. (Как упоминалось выше, форма интерфейса

IEnumerable<T> реализуется во всех массивах, как, впрочем, и во многих других источниках

данных.) Но если в источнике данных реализован необобщенный вариант

интерфейса IEnumerable, то тип переменной диапазона придется указывать явно.

И это делается в операторе from. Ниже приведен пример явного объявления типа

int переменной диапазона n.

var posNums = from int n in nums

Тип объекта, возвращаемого по запросу, представляет собой экземпляр интерфейса

IEnumerable<T>, где Т — тип получаемых элементов. Следовательно, тип переменной

запроса должен быть экземпляром интерфейса IEnumerable<T>, а значение

Т должно определяться типом значения, указываемым в операторе select.

С учетом явного указания типа IEnumerable<int> упомянутый выше запрос можно было бы составить следующим образом.

IEnumerable<int> posNums = from n in nums

where n > 0

select n;

Следует иметь в виду, что тип элемента, выбираемого оператором select, должен

соответствовать типу аргумента, передаваемого форме интерфейса IEnumerable<T>,

используемой для объявления переменной запроса.

Зачастую при объявлении переменных

запроса используется ключевое слово var вместо явного указания ее типа,

поскольку это дает компилятору возможность самому вывести соответствующий тип

данной переменной из оператора select.

Когда запрос выполняется в цикле foreach, тип переменной шага цикла должен

быть таким же, как и тип переменной диапазона. В предыдущих примерах тип этой

переменной указывался явно как int. Но имеется и другая возможность: предоставить

компилятору самому вывести тип данной переменной, и для этого достаточно указать

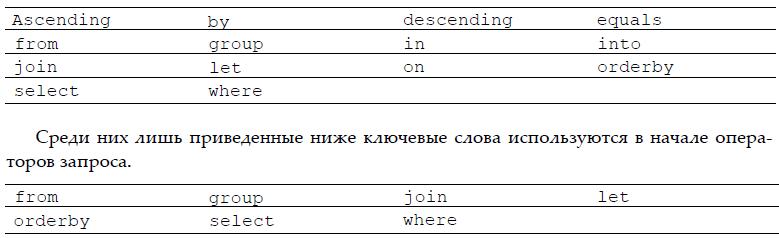
ее тип как var. Как будет показано далее в этой главе, ключевое слово var приходится

использовать и в тех случаях, когда тип данных просто неизвестен.

**Общая форма запроса**

У всех запросов имеется общая форма, основывающаяся на ряде приведенных ниже

контекстно-зависимых ключевых слов.



Запрос должен начинаться с ключевого слова from и оканчиваться ключевым словом

select или group. Оператор select определяет тип значения, перечисляемого

по запросу, а оператор group возвращает данные группами, причем каждая группа

может перечисляться по отдельности. Как следует из приведенных выше примеров,

в операторе where указываются критерии, которым должен удовлетворять искомый

элемент, чтобы быть полученным по запросу. А остальные операторы позволяют уточнить

запрос. Все они рассматриваются далее по порядку.

**Отбор запрашиваемых значений с помощью оператора where**

Как пояснялось выше, оператор where служит для отбора данных, возвращаемых

по запросу. В предыдущих примерах этот оператор был продемонстрирован в своей

простейшей форме, в которой для отбора данных используется единственное условие.

Однако для более тщательного отбора данных можно задать несколько условий

и, в частности, в нескольких операторах where. В качестве примера рассмотрим следующую

программу, в которой из массива выводятся только те значения, которые положительны

и меньше 10.

(***glava19\_3***)

class TwoWheres

{

static void Main()

{

int[] nums = { 1, -2, 3, -3, 0, -8, 12, 19, 6, 9, 10 };

//make query to get positive less then 10

var posNums = from n in nums

where n > 0

where n < 10

select n;

Console.Write("Positive values less then 10: ");

foreach (var i in posNums) Console.Write(i + " ");

Console.WriteLine();

}

}

Условие в первом операторе where требует, чтобы элемент массива был больше

нуля. А условие во втором операторе where требует, чтобы элемент массива был меньше

10. Следовательно, запрашиваемый элемент массива должен находиться в пределах

от 1 до 9 (включительно), чтобы удовлетворять обоим условиям.

В таком применении двух операторов where для отбора данных нет ничего дурного,

но аналогичного эффекта можно добиться с помощью более компактно составленного

условия в единственном операторе where. Ниже приведен тот же самый запрос,

переформированный по этому принципу.

var posNums = from n in nums

where n > 0 where n < 10

select n;

Как правило, в условии оператора where разрешается использовать любое допустимое

в C# выражение, дающее булев результат. Например, в приведенной ниже программе

определяется массив символьных строк. В ряде этих строк содержатся адреса

Интернета. По запросу в переменой netAddrs извлекаются только те строки, которые

содержат более четырех символов и оканчиваются на *"*.net*"*. Следовательно, по данному

запросу обнаруживаются строки, содержащие адреса Интернета с именем .net

домена самого верхнего уровня.

(***glava19\_4***)

class WhereDemo2

{

static void Main()

{

string[] str = {".com", ".net", "hsNameA.com",

"hsNameB.net", "test", ".network", "hsNameC.net", "hsNameD.com"};

//make query to get addresses of internet

//that ends with .net

var newAddrs = from adr in str

where adr.Length > 4 && adr.EndsWith(".net", StringComparison.Ordinal)

select adr;

//show results

foreach (var strr in newAddrs) Console.WriteLine(strr);

}

}

Обратите внимание на то, что в операторе where данной программы используется

один из методов обработки символьных строк под названием EndsWith(). Он возвращает

логическое значение true, если вызывающая его строка оканчивается последовательностью

символов, указываемой в качестве аргумента этого метода.