1. 使用查询语法。
2. 使用方法语法。
3. 组合使用查询语法和方法语法。

下面的示例使用前面列出的每种方式演示一些简单的 LINQ 查询。一般的规则是尽可能使用 (1)，而在必要时使用 (2) 和 (3)。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 这些查询作用于简单的内存中集合；但是，基本语法与 LINQ to SQL 和 LINQ to XML 中使用的语法相同。 |

**查询语法**

编写大多数查询的推荐方式是使用查询语法来创建查询表达式。下面的示例演示了三个查询表达式。第一个查询表达式演示如何通过用 **where** 子句应用条件来筛选或限制结果，它返回源序列中值大于 7 或小于 3 的所有元素。第二个表达式演示如何对返回的结果进行排序。第三个表达式演示如何按照键对结果进行分组，此查询可根据单词的第一个字母返回两个组。

// Query #1.

List<int> numbers = new List<int>() { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

// The query variable can also be implicitly typed by using var

IEnumerable<int> filteringQuery =

from num in numbers

where num < 3 || num > 7

select num;

// Query #2.

IEnumerable<int> orderingQuery =

from num in numbers

where num < 3 || num > 7

orderby num ascending

select num;

// Query #3.

string[] groupingQuery = { "carrots", "cabbage", "broccoli", "beans", "barley" };

IEnumerable<IGrouping<char, string>> queryFoodGroups =

from item in groupingQuery

group item by item[0];

请注意，这些查询的类型是 [IEnumerable<T>](https://msdn.microsoft.com/zh-cn/library/9eekhta0.aspx)。所有这些查询都可以使用 **var** 编写，如下面的示例所示：

var query = from num in numbers...

在上述每个示例中，直到您在 **foreach** 语句中循环访问查询变量时，查询才会实际执行。有关更多信息，请参见 [Introduction to LINQ Queries (C#)](https://msdn.microsoft.com/zh-cn/library/bb397906.aspx)。

**方法语法**

某些查询操作必须表示为方法调用。最常见的此类方法是那些返回单一数值的方法，如 [Sum](https://msdn.microsoft.com/zh-cn/library/bb298138.aspx)、[Max](https://msdn.microsoft.com/zh-cn/library/bb335614.aspx)、[Min](https://msdn.microsoft.com/zh-cn/library/bb298087.aspx)、[Average](https://msdn.microsoft.com/zh-cn/library/bb354760.aspx) 等。这些方法在任何查询中都必须总是最后调用，因为它们仅表示单个值，不能充当其他查询操作的数据源。下面的示例演示查询表达式中的方法调用：

List<int> numbers1 = new List<int>() { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

List<int> numbers2 = new List<int>() { 15, 14, 11, 13, 19, 18, 16, 17, 12, 10 };

// Query #4.

double average = numbers1.Average();

// Query #5.

IEnumerable<int> concatenationQuery = numbers1.Concat(numbers2);

如果该方法具有参数，则这些参数以 [lambda](https://msdn.microsoft.com/zh-cn/library/bb397687.aspx) 表达式的形式提供，如下面的示例所示：

// Query #6.

IEnumerable<int> largeNumbersQuery = numbers2.Where(c => c > 15);

在上述查询中，只有查询 4 立即执行。这是因为它返回单个值，而不是一个泛型 [IEnumerable<T>](https://msdn.microsoft.com/zh-cn/library/9eekhta0.aspx) 集合。方法本身必须使用 **foreach** 才能计算它的值。

上述每个查询都可以通过结合使用隐式类型化与 [var](https://msdn.microsoft.com/zh-cn/library/bb383973.aspx) 进行编写，如下面的示例所示：

// var is used for convenience in these queries

var average = numbers1.Average();

var concatenationQuery = numbers1.Concat(numbers2);

var largeNumbersQuery = numbers2.Where(c => c > 15);

**混合的查询和方法语法**

此示例演示如何对查询子句的结果使用方法语法。只需将查询表达式括在括号内，然后应用点运算符并调用此方法。在下面的示例中，查询 7 返回其值在 3 和 7 之间的数字个数。但是，通常更好的做法是使用另一个变量来存储方法调用的结果。这样就不太容易将查询本身与查询结果相混淆。

// Query #7.

// Using a query expression with method syntax

int numCount1 =

(from num in numbers1

where num < 3 || num > 7

select num).Count();

// Better: Create a new variable to store

// the method call result

IEnumerable<int> numbersQuery =

from num in numbers1

where num < 3 || num > 7

select num;

int numCount2 = numbersQuery.Count();

由于查询 7 返回单个值而不是一个集合，因此该查询立即执行。

上述查询可以通过结合使用隐式类型化与 **var** 进行编写，如下所示：

var numCount = (from num in numbers...

它可以按如下方式使用方法语法进行编写：

var numCount = numbers.Where(n => n < 3 || n > 7).Count();

它可以按如下方式使用显式类型化进行编写：

int numCount = numbers.Where(n => n < 3 || n > 7).Count();

# 在内存中存储查询结果（C# 编程指南）

查询基本上是一组有关如何检索和组织数据的指令。若要执行查询，需要调用它的 [GetEnumerator](https://msdn.microsoft.com/zh-cn/library/s793z9y2.aspx) 方法。当您使用 **foreach** 循环来循环访问元素时，将执行此调用。若要计算查询和存储其结果，而不执行 **foreach** 循环，请对查询变量调用下列方法之一：

* [ToList<TSource>](https://msdn.microsoft.com/zh-cn/library/bb342261.aspx)
* [ToArray<TSource>](https://msdn.microsoft.com/zh-cn/library/bb298736.aspx)
* [ToDictionary<TSource, TKey, TElement>](https://msdn.microsoft.com/zh-cn/library/bb548657.aspx)
* [ToLookup<TSource, TKey, TElement>](https://msdn.microsoft.com/zh-cn/library/bb549211.aspx)

建议在存储查询结果时，将返回的集合对象分配给一个新变量，如下面的示例所示：

class StoreQueryResults

{

static List<int> numbers = new List<int>() { 1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };

static void Main()

{

IEnumerable<int> queryFactorsOfFour =

from num in numbers

where num % 4 == 0

select num;

// Store the results in a new variable

// without executing a foreach loop.

List<int> factorsofFourList = queryFactorsOfFour.ToList();

// Iterate the list just to prove it holds data.

foreach (int n in factorsofFourList)

{

Console.WriteLine(n);

}

// Keep the console window open in debug mode.

Console.WriteLine("Press any key");

Console.ReadKey();

}

}

**请参阅**

[LINQ (Language-Integrated Query)](https://msdn.microsoft.com/zh-cn/library/bb397926.aspx)

[Walkthrough: Writing Queries in C# (LINQ)](https://msdn.microsoft.com/zh-cn/library/bb397900.aspx)

[LINQ 查询表达式（C# 编程指南）](https://msdn.microsoft.com/zh-cn/library/bb397676.aspx)

[where 子句（C# 参考）](https://msdn.microsoft.com/zh-cn/library/bb311043.aspx)