*查询*是一种从数据源检索数据的表达式。 不同的数据源具有不同的原生查询语言，例如，用于关系数据库的 SQL 和用于 XML 的 XQuery。 开发人员对于他们必须支持的每种数据源或数据格式，都必须学习一种新的查询语言。 LINQ 通过为各种数据源和数据格式提供一致的 C# 语言模型，简化了这一情况。 在 LINQ 查询中，你始终使用 C# 对象。 当有 LINQ 提供程序可用时，你可以使用相同的基本编码模式来查询和转换 XML 文档、SQL 数据库、.NET 集合中的数据以及任何其他格式的数据。

**查询操作的三个部分**

所有 LINQ 查询操作都由以下三个不同的操作组成：

1. 获取数据源。
2. 创建查询。
3. 执行查询。

下面的示例演示如何用源代码表示查询操作的三个部分。 为方便起见，此示例将一个整数数组用作数据源；但其中涉及的概念同样适用于其他数据源。 本文的其余部分也引用了此示例。

C#复制

// The Three Parts of a LINQ Query:

// 1. Data source.

int[] numbers = [ 0, 1, 2, 3, 4, 5, 6 ];

// 2. Query creation.

// numQuery is an IEnumerable<int>

var numQuery =

from num in numbers

where (num % 2) == 0

select num;

// 3. Query execution.

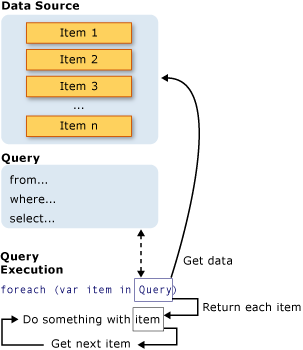
foreach (int num in numQuery)

{

Console.Write("{0,1} ", num);

}

下图演示完整的查询操作。 在 LINQ 中，查询的执行不同于查询本身。 换句话说，你不会通过创建查询变量来检索任何数据。



**数据源**

上例中的数据源是一个数组，它支持泛型 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) 接口。 这一事实意味着该数据源可以用 LINQ 进行查询。 查询在 foreach 语句中执行，且 foreach 需要 [IEnumerable](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.ienumerable) 或 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1)。 支持 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) 或派生接口（如泛型 [IQueryable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iqueryable-1)）的类型称为可查询类型 。

可查询类型不需要进行修改或特殊处理就可以用作 LINQ 数据源。 如果源数据还没有作为可查询类型出现在内存中，则 LINQ 提供程序必须以此方式表示源数据。 例如，LINQ to XML 将 XML 文档加载到可查询的 [XElement](https://learn.microsoft.com/zh-cn/dotnet/api/system.xml.linq.xelement) 类型中：

C#复制

// Create a data source from an XML document.

// using System.Xml.Linq;

XElement contacts = XElement.Load(@"c:\myContactList.xml");

使用 [EntityFramework](https://learn.microsoft.com/zh-cn/ef/core/)，你在 C# 类与数据库架构之间创建对象关系映射。 你针对这些对象编写查询，然后 EntityFramework 在运行时处理与数据库的通信。 下例中，Customers 表示数据库中的特定表，而查询结果的类型 [IQueryable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iqueryable-1) 派生自 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1)。

C#复制

Northwnd db = new Northwnd(@"c:\northwnd.mdf");

// Query for customers in London.

IQueryable<Customer> custQuery =

from cust in db.Customers

where cust.City == "London"

select cust;

有关如何创建特定类型的数据源的详细信息，请参阅各种 LINQ 提供程序的文档。 但基本规则很简单：LINQ 数据源是支持泛型 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) 接口（或者是继承该接口的接口，通常表示为 [IQueryable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iqueryable-1)）的任何对象。

**备注**

支持非泛型 [**IEnumerable**](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.ienumerable) 接口的类型（如 [**ArrayList**](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.arraylist)）还可用作 LINQ 数据源。 有关详细信息，请参阅[**如何使用 LINQ 查询 ArrayList (C#)**](https://learn.microsoft.com/zh-cn/dotnet/csharp/linq/how-to-query-collections)。

**查询**

查询指定要从数据源中检索的信息。 查询还可以指定在返回这些信息之前如何对其进行排序、分组和结构化。 查询存储在查询变量中，并用查询表达式进行初始化。 你使用 [C# 查询语法](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/keywords/query-keywords)来编写查询。

上一个示例中的查询从整数数组中返回所有偶数。 该查询表达式包含三个子句：from、where 和 select。 （如果你熟悉 SQL，你会注意到这些子句的顺序与 SQL 中的顺序相反。）from 子句指定数据源，where 子句应用筛选器，select 子句指定返回的元素的类型。 本部分详细介绍了所有查询子句。 目前需要注意的是，在 LINQ 中，查询变量本身不执行任何操作并且不返回任何数据。 它只是存储在以后某个时刻执行查询时为生成结果而必需的信息。 有关如何构造查询的详细信息，请参阅[标准查询运算符概述 (C#)](https://learn.microsoft.com/zh-cn/dotnet/csharp/linq/standard-query-operators/)。

**备注**

还可以使用方法语法来表示查询。 有关详细信息，请参阅 [**LINQ 中的查询语法和方法语法**](https://learn.microsoft.com/zh-cn/dotnet/csharp/linq/get-started/write-linq-queries)。

**标准查询运算符按执行方式的分类**

标准查询运算符方法的 LINQ to Objects 实现主要通过两种方法之一执行：立即执行和延迟执行。 使用延迟执行的查询运算符可以进一步分为两种类别：流式处理和非流式处理。

**即时**

立即执行指的是读取数据源并执行一次运算。 返回标量结果的所有标准查询运算符都立即执行。 Count、Max、Average 和 First 就属于此类查询。 由于查询本身必须使用 foreach 来返回结果，因此这些方法在执行时不使用显式 foreach 语句。 这些查询返回单个值，而不是 IEnumerable 集合。 可以使用 [Enumerable.ToList](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.tolist) 或 [Enumerable.ToArray](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.toarray) 方法强制任何查询立即执行。 立即执行可重用查询结果，而不是查询声明。 结果被检索一次，然后存储以供将来使用。 下面的查询返回源数组中偶数的计数：

C#复制

var evenNumQuery =

from num in numbers

where (num % 2) == 0

select num;

int evenNumCount = evenNumQuery.Count();

要强制立即执行任何查询并缓存其结果，可调用 [ToList](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.tolist) 或 [ToArray](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.toarray) 方法。

C#复制

List<int> numQuery2 =

(from num in numbers

where (num % 2) == 0

select num).ToList();

// or like this:

// numQuery3 is still an int[]

var numQuery3 =

(from num in numbers

where (num % 2) == 0

select num).ToArray();

此外，还可以通过在紧跟查询表达式之后的位置放置一个 foreach 循环来强制执行查询。 但是，通过调用 ToList 或 ToArray，也可以将所有数据缓存在单个集合对象中。

**已推迟**

延迟执行指的是不在代码中声明查询的位置执行运算。 仅当对查询变量进行枚举时才执行运算，例如通过使用 foreach 语句执行。 查询的执行结果取决于执行查询而非定义查询时的数据源内容。 如果多次枚举查询变量，则每次结果可能都不同。 几乎所有返回类型为 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) 或 [IOrderedEnumerable<TElement>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iorderedenumerable-1) 的标准查询运算符皆以延迟方式执行。 延迟执行提供了查询重用功能，因为在每次循环访问查询结果时，查询都会从数据源中提取更新的数据。 以下代码演示了延迟执行的示例：

C#复制

foreach (int num in numQuery)

{

Console.Write("{0,1} ", num);

}

foreach 语句也是检索查询结果的地方。 例如，在上一个查询中，迭代变量 num 保存了返回的序列中的每个值（一次保存一个值）。

由于查询变量本身从不保存查询结果，因此你可以重复执行它来检索更新的数据。 例如，单独的应用程序可能会不断更新数据库。 在应用程序中，你可以创建一个检索最新数据的查询，并可以不时地执行该查询以便检索更新的结果。

使用延迟执行的查询运算符可以进一步分类为流式处理和非流式处理。

**流式处理**

流式处理运算符不需要在生成元素前读取所有源数据。 在执行时，流式处理运算符一边读取每个源元素，一边对该源元素执行运算，并在可行时生成元素。 流式处理运算符将持续读取源元素直到可以生成结果元素。 这意味着可能要读取多个源元素才能生成一个结果元素。

**非流式处理**

非流式处理运算符必须先读取所有源数据，然后才能生成结果元素。 排序或分组等运算均属于此类别。 在执行时，非流式处理查询运算符读取所有源数据，将其放入数据结构，执行运算，然后生成结果元素。

**分类表**

下表按照执行方法对每个标准查询运算符方法进行了分类。

**备注**

如果某个运算符被标入两个列中，则表示在运算中涉及两个输入序列，每个序列的计算方式不同。 在此类情况下，参数列表中的第一个序列始终以延迟流式处理方式来执行计算。

展开表

| **标准查询运算符** | **返回类型** | **立即执行** | **延迟的流式处理执行** | **延迟非流式处理执行** |
| --- | --- | --- | --- | --- |
| [Aggregate](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.aggregate) | TSource | X |  |  |
| [All](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.all) | [Boolean](https://learn.microsoft.com/zh-cn/dotnet/api/system.boolean) | X |  |  |
| [Any](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.any) | [Boolean](https://learn.microsoft.com/zh-cn/dotnet/api/system.boolean) | X |  |  |
| [AsEnumerable](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.asenumerable) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Average](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.average) | 单个数值 | x |  |  |
| [Cast](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.cast) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Concat](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.concat) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Contains](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.contains) | [Boolean](https://learn.microsoft.com/zh-cn/dotnet/api/system.boolean) | X |  |  |
| [Count](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.count) | [Int32](https://learn.microsoft.com/zh-cn/dotnet/api/system.int32) | X |  |  |
| [DefaultIfEmpty](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.defaultifempty) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Distinct](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.distinct) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [ElementAt](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.elementat) | TSource | X |  |  |
| [ElementAtOrDefault](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.elementatordefault) | TSource? | X |  |  |
| [Empty](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.empty) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) | X |  |  |
| [Except](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.except) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X | X |
| [First](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.first) | TSource | X |  |  |
| [FirstOrDefault](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.firstordefault) | TSource? | X |  |  |
| [GroupBy](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.groupby) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  |  | X |
| [GroupJoin](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.groupjoin) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X | X |
| [Intersect](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.intersect) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X | X |
| [Join](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.join) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X | X |
| [Last](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.last) | TSource | X |  |  |
| [LastOrDefault](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.lastordefault) | TSource? | X |  |  |
| [LongCount](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.longcount) | [Int64](https://learn.microsoft.com/zh-cn/dotnet/api/system.int64) | X |  |  |
| [Max](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.max) | 单个数值 TSource 或 TResult? | X |  |  |
| [Min](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.min) | 单个数值 TSource 或 TResult? | X |  |  |
| [OfType](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.oftype) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [OrderBy](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.orderby) | [IOrderedEnumerable<TElement>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iorderedenumerable-1) |  |  | X |
| [OrderByDescending](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.orderbydescending) | [IOrderedEnumerable<TElement>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iorderedenumerable-1) |  |  | X |
| [Range](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.range) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Repeat](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.repeat) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Reverse](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.reverse) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  |  | X |
| [Select](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.select) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [SelectMany](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.selectmany) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [SequenceEqual](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.sequenceequal) | [Boolean](https://learn.microsoft.com/zh-cn/dotnet/api/system.boolean) | X |  |  |
| [Single](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.single) | TSource | X |  |  |
| [SingleOrDefault](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.singleordefault) | TSource? | X |  |  |
| [Skip](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.skip) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [SkipWhile](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.skipwhile) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Sum](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.sum) | 单个数值 | x |  |  |
| [Take](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.take) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [TakeWhile](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.takewhile) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [ThenBy](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.thenby) | [IOrderedEnumerable<TElement>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iorderedenumerable-1) |  |  | X |
| [ThenByDescending](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.thenbydescending) | [IOrderedEnumerable<TElement>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.iorderedenumerable-1) |  |  | X |
| [ToArray](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.toarray) | TSource[] 数组 | X |  |  |
| [ToDictionary](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.todictionary) | [Dictionary<TKey,TValue>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.dictionary-2) | X |  |  |
| [ToList](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.tolist) | [IList<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ilist-1) | X |  |  |
| [ToLookup](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.tolookup) | [ILookup<TKey,TElement>](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.ilookup-2) | X |  |  |
| [Union](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.union) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |
| [Where](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.where) | [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) |  | X |  |

**LINQ to objects**

“LINQ to Objects”是指将 LINQ 查询直接用于任何 [IEnumerable](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.ienumerable) 或 [IEnumerable<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.ienumerable-1) 集合。 可以使用 LINQ 来查询任何可枚举的集合，例如 [List<T>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.list-1)、[Array](https://learn.microsoft.com/zh-cn/dotnet/api/system.array) 或 [Dictionary<TKey,TValue>](https://learn.microsoft.com/zh-cn/dotnet/api/system.collections.generic.dictionary-2)。 该集合可以是用户定义的集合，也可以是由 .NET API 返回的类型。 而采用 LINQ 方法，只需编写描述要检索的内容的声明性代码。 LINQ to Objects 很好地介绍了如何使用 LINQ 进行编程。

LINQ 查询与传统 foreach 循环相比具有三大优势：

* 它们更简明、更易读，尤其在筛选多个条件时。
* 它们使用最少的应用程序代码提供强大的筛选、排序和分组功能。
* 无需修改或只需做很小的修改即可将它们移植到其他数据源。

要对数据执行的操作越复杂，就越能体会到 LINQ 相较于传统迭代技术的优势。

**在内存中存储查询结果**

查询基本上是针对如何检索和组织数据的一套说明。 当请求结果中的每个后续项目时，查询将延迟执行。 使用 foreach 循环访问结果时，项将在受到访问时返回。 若要在不执行 foreach 循环的情况下评估查询并存储其结果，只需调用查询变量上的以下方法之一：

* [ToList](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.tolist)
* [ToArray](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.toarray)
* [ToDictionary](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.todictionary)
* [ToLookup](https://learn.microsoft.com/zh-cn/dotnet/api/system.linq.enumerable.tolookup)

在存储查询结果时，应将返回的集合对象分配给一个新变量，如下面的示例所示：

C#复制

List<int> numbers = [1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20];

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.

var factorsofFourList = queryFactorsOfFour.ToList();

// Read and write from the newly created list to demonstrate that it holds data.

Console.WriteLine(factorsofFourList[2]);

factorsofFourList[2] = 0;

Console.WriteLine(factorsofFourList[2]);

**另请参阅**

* [演练：用 C# 编写查询](https://learn.microsoft.com/zh-cn/dotnet/csharp/linq/get-started/walkthrough-writing-queries-linq)
* [foreach, in](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/statements/iteration-statements#the-foreach-statement)
* [查询关键字 (LINQ)](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/keywords/query-keywords)