# 1.LinQ用途:

## 使用相同的基本查询表达式模式来查询和转换 SQL 数据库、ADO.NET 数据集、XML 文档和流以及 .NET 集合中的数据。

# 2.LinQ的特点

## LINQ，查询现在已是高级语言构造，就如同类、方法、事件等等。

* 查询表达式可用于查询和转换来自任意支持 LINQ 的数据源中的数据。例如，单个查询可以从 SQL 数据库检索数据，并生成 XML 流作为输出。
* 查询表达式容易掌握，因为它们使用许多常见的 C# 语言构造。有关更多信息，请参见 [Getting Started with LINQ in C#](https://msdn.microsoft.com/zh-cn/library/bb397933.aspx)。
* 查询表达式中的变量都是强类型的，但许多情况下您不需要显式提供类型，因为编译器可以推断类型。有关更多信息，请参见[Type Relationships in LINQ Query Operations (C#)](https://msdn.microsoft.com/zh-cn/library/bb397924.aspx)。
* 在您循环访问 **foreach** 语句中的查询变量之前，不会执行查询。有关更多信息，请参见 [Introduction to LINQ Queries (C#)](https://msdn.microsoft.com/zh-cn/library/bb397906.aspx)。
* 在编译时，根据 C# 规范中设置的规则将查询表达式转换为“标准查询运算符”方法调用。任何可以使用查询语法表示的查询也可以使用方法语法表示。但是，在大多数情况下，查询语法更易读和简洁。有关更多信息，请参见 [C# 语言规范](https://msdn.microsoft.com/zh-cn/library/ms228593.aspx)和[Standard Query Operators Overview](https://msdn.microsoft.com/zh-cn/library/bb397896.aspx)。
* 作为编写 LINQ 查询的一项规则，建议尽量使用查询语法，只在必需的情况下才使用方法语法。这两种不同形式在语义或性能上没有区别。查询表达式通常比用方法语法编写的等效表达式更易读。
* 一些查询操作，如 [Count<TSource>](https://msdn.microsoft.com/zh-cn/library/bb338038.aspx) 或 [Max](https://msdn.microsoft.com/zh-cn/library/bb335614.aspx)，没有等效的查询表达式子句，因此必须表示为方法调用。方法语法可以通过多种方式与查询语法组合。有关更多信息，请参见[Query Syntax and Method Syntax in LINQ (C#)](https://msdn.microsoft.com/zh-cn/library/bb397947.aspx)。
* 查询表达式可以编译为表达式树或委托，具体取决于查询所应用到的类型。 [IEnumerable<T>](https://msdn.microsoft.com/zh-cn/library/9eekhta0.aspx) 查询编译为委托。 [IQueryable](https://msdn.microsoft.com/zh-cn/library/system.linq.iqueryable.aspx) 和 [IQueryable<T>](https://msdn.microsoft.com/zh-cn/library/bb351562.aspx) 查询编译为表达式树。有关更多信息，请参见[表达式树（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-cn/library/bb397951.aspx)。

# 3.LinQ实例

class LINQQueryExpressions

{

static void Main()

{

// Specify the data source.

int[] scores = new int[] { 97, 92, 81, 60 };

// Define the query expression.

IEnumerable<int> scoreQuery =

from score in scores

where score > 80

select score;

// Execute the query.

foreach (int i in scoreQuery)

{

Console.Write(i + " ");

}

}

}

// Output: 97 92 81

# 4.实例2.LinQ操作xml文件

## 准备一个PurchaseOrder.xml文件,内容如下

|  |
| --- |
| <?xml version="1.0"?>  <PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">  <Address Type="Shipping">  <Name>Ellen Adams</Name>  <Street>123 Maple Street</Street>  <City>Mill Valley</City>  <State>CA</State>  <Zip>10999</Zip>  <Country>USA</Country>  </Address>  <Address Type="Billing">  <Name>Tai Yee</Name>  <Street>8 Oak Avenue</Street>  <City>Old Town</City>  <State>PA</State>  <Zip>95819</Zip>  <Country>USA</Country>  </Address>  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>  <Items>  <Item PartNumber="872-AA">  <ProductName>Lawnmower</ProductName>  <Quantity>1</Quantity>  <USPrice>148.95</USPrice>  <Comment>Confirm this is electric</Comment>  </Item>  <Item PartNumber="873-AA">  <ProductName>Toy Gun</ProductName>  <Quantity>1</Quantity>  <USPrice>105.95</USPrice>  <Comment>Confirm this is electric</Comment>  </Item>  <Item PartNumber="875-AA">  <ProductName>Sound Box Speaker</ProductName>  <Quantity>1</Quantity>  <USPrice>118.5</USPrice>  <Comment>Confirm this is electric</Comment>  </Item>  <Item PartNumber="926-AA">  <ProductName>Baby Monitor</ProductName>  <Quantity>2</Quantity>  <USPrice>39.98</USPrice>  <ShipDate>1999-05-21</ShipDate>  </Item>  </Items>  </PurchaseOrder> |

## 然后写一个静态函数, TestLinqToXML()

|  |
| --- |
| static void TestLinqToXML()  {  var fileName = "PurchaseOrder.xml";  var currDir = Environment.CurrentDirectory;  var filePath = Path.Combine(currDir, fileName);  XElement purchaseOrder = XElement.Load(filePath);  IEnumerable<XElement> pricesByPartNos = from item in purchaseOrder.Descendants("Item")  where (int)item.Element("Quantity") \* (decimal)item.Element("USPrice")>100  orderby (string)item.Element("PartNumber")  select item;  foreach (var item in pricesByPartNos)  {  Console.WriteLine(item);  }    } |

### 注意,也可以使用Lambda函数的写法

|  |
| --- |
| static void TestLinqToXMLLambda()  {  var filepath = Environment.CurrentDirectory+ "\\PurchaseOrder.xml";  XElement parchaseOrder = XElement.Load(filepath);  IEnumerable<XElement> pricesByPartNos = parchaseOrder.Descendants("Item")  .Where(item => (int)item.Element("Quantity") \* (decimal)item.Element("USPrice") > 100)  .OrderBy(order=>order.Element("PartNumber"));  foreach (var item in pricesByPartNos)  {  Console.WriteLine(item);  }  } |

### 效果如下

|  |
| --- |
|  |

# 5.LinQ实例3.用LinQ生成一副扑克牌,需要2给辅助函数,他们是生成器函数,返回的是可迭代对象

|  |
| --- |
| using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Xml;  using System.Xml.Linq;  namespace csharpbasic\_linqdemo  {  static class Program //必须是静态类  {  static IEnumerable<string> Suits()//生成扑克牌辅助函数一  {  yield return "clubs";  yield return "diamonds";  yield return "spades";  yield return "hearts";  }  static IEnumerable<string> Ranks()//生成扑克牌辅助函数二  {  yield return "two";  yield return "three";  yield return "four";  yield return "five";  yield return "six";  yield return "seven";  yield return "eight";  yield return "nine";  yield return "ten";  yield return "jack";  yield return "queen";  yield return "king";  yield return "ace";  }  static void LinQCardDemo() //用linq生成一幅扑克牌  {  //一对多查询  var deckCards = Suits().SelectMany(suit=>Ranks().Select(rank=>new {Suit = suit,Rank = rank}));  foreach (var item in deckCards)  {  Console.WriteLine(item);  }  }  //洗牌函数,这个函数是扩展方法,必须把类的声明改为static,必须是非泛型静态类  public static IEnumerable<T> ShaffulCard<T>(this IEnumerable<T> first, IEnumerable<T> second)  {  var firstItor = first.GetEnumerator();//获取第一个迭代器  var secondItor = second.GetEnumerator();//获取第二个迭代器  while (firstItor.MoveNext() && secondItor.MoveNext())  {  yield return firstItor.Current;  yield return secondItor.Current;  }  }  static void Main(string[] args)  {  LinQCardDemo();  }  }  } |

### 效果

|  |
| --- |
|  |

## 然后我们可以添加一个洗牌的方法,这是一个扩展方法,需要在非泛型静态类中定义

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Xml;  using System.Xml.Linq;  namespace csharpbasic\_linqdemo  {  static class Program //必须是静态类  {  static IEnumerable<string> Suits()//生成扑克牌辅助函数一  {  yield return "clubs";  yield return "diamonds";  yield return "spades";  yield return "hearts";  }  static IEnumerable<string> Ranks()//生成扑克牌辅助函数二  {  yield return "two";  yield return "three";  yield return "four";  yield return "five";  yield return "six";  yield return "seven";  yield return "eight";  yield return "nine";  yield return "ten";  yield return "jack";  yield return "queen";  yield return "king";  yield return "ace";  }  static void LinQCardDemo() //用linq生成一幅扑克牌  {  //一对多查询  var deckCards = Suits().SelectMany(suit=>Ranks().Select(rank=>new {Suit = suit,Rank = rank}));  foreach (var item in deckCards)  {  Console.WriteLine(item);  }  }  //洗牌函数,这个函数是扩展方法,必须把类的声明改为static,必须是非泛型静态类  public static IEnumerable<T> ShaffulCard<T>(this IEnumerable<T> first, IEnumerable<T> second)  {  var firstItor = first.GetEnumerator();//获取第一个迭代器  var secondItor = second.GetEnumerator();//获取第二个迭代器  while (firstItor.MoveNext() && secondItor.MoveNext())  {  yield return firstItor.Current;  yield return secondItor.Current;  }  }  static void ShuffleDemo()  {  var deckCards = Suits().SelectMany(suit => Ranks().Select(rank => new { Suit = suit, Rank = rank }));  Console.WriteLine("Before Shuffle\n");  foreach (var item in deckCards)  {  Console.WriteLine(item);  }  var first = deckCards.Take(26);  var second = deckCards.Skip(26);  var result = first.ShaffulCard(second);  Console.WriteLine("After Shuffle\n");  foreach (var item in result)  {  Console.WriteLine(item);  }  }    static void Main(string[] args)  {  //LinQCardDemo();  ShuffleDemo();    }  }  } |

### 效果

|  |
| --- |
|  |
|  |

### 更多.参考这里: <https://learn.microsoft.com/zh-cn/dotnet/csharp/tutorials/working-with-linq>

# 扩展参考XML 文档：[LINQ to XML](https://learn.microsoft.com/zh-cn/dotnet/standard/linq/linq-xml-overview)

* ADO.NET 实体框架：[LINQ to Entities](https://learn.microsoft.com/zh-cn/dotnet/framework/data/adonet/ef/language-reference/linq-to-entities)
* .NET 集合、文件、字符串等：[LINQ to objects](https://learn.microsoft.com/zh-cn/dotnet/csharp/programming-guide/concepts/linq/linq-to-objects)

若要更深入地全面了解 LINQ，请参阅 [C# 中的 LINQ](https://learn.microsoft.com/zh-cn/dotnet/csharp/linq/linq-in-csharp)。

若要开始在 C# 中使用 LINQ，请参阅教程[使用 LINQ](https://learn.microsoft.com/zh-cn/dotnet/csharp/tutorials/working-with-linq)。

这里有一个Microsoft的linq参考: [console-linq](file:///G:\C以及其他教程\后端\.Net\.net电子书+资料\cSharp-pdf电子书\dotnet-samples-main\csharp\getting-started\console-linq)