**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**🙢🙢🙠🙠**



**BÁO CÁO ĐỒ ÁN**

**Môn Học: Công Nghệ .Net**

**Đề tài: Tìm hiểu về LINQ**

**❖ Giảng viên hướng dẫn ❖**

ThS. Huỳnh Hồ Thị Mộng Trinh

**Lớp :** SE310.K21.PMCL

**Sinh viên thực hiện :** Nguyễn Bá Tùng – 16521395

**Tp. Hồ Chí Minh, tháng 03 năm 2020**

# **NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

.....................................................................................................................................

*......................., ngày ............. tháng ………. năm 2020*

# **LỜI CẢM ƠN**

Lời đầu tiên, em muốn gửi lời cảm ơn đặc biệt đến ThS. Huỳnh Hồ Thị Mộng Trinh đã hỗ trợ em nhiệt tình trong thời gian học; đã truyền tải những kiến thức quý báu về môn học, và luôn sẵn sàng giúp em giải quyết những thắc mắc và các vấn đề gặp phải.

Mặt khác, em rất biết ơn sự giúp đỡ đến từ các giảng viên khoa Công nghệ phần mềm và bạn bè trong lớp, nhận xét và đánh giá của mọi người đã cho chúng em nhiều kiến thức và sự hiểu biết để chúng em có thể hoàn thành được đồ án môn học

*Thành phố Hồ Chí Minh, ngày 26 tháng 03 năm 2020*

**Danh mục hình ảnh trong tài liệu này**

|  |  |
| --- | --- |
| **Số** | **Tiêu đề** |
| 1 | Trích xuất dữ liệu cho ứng dụng trên nền .NET trước kia |
| 2 | Tương tác giữa LINQ đến các nguồn dữ liệu khác nhau |
| 3 | Kiến trúc của LINQ |
| 4 | LINQ hoạt động như thế nào |
| 5 | Cú pháp của LINQ Query |
| 6 | Cú pháp của LINQ Method |
| 7 | Mô tả cấu trúc của phương thức where() |
| 8 | Kiến trúc của LINQ to SQL |
| 9 | Tạo Database Connection |
| 10 | Tạo mới class LINQ to SQL |
| 11 | Thêm table từ database vào LINQ to SQL class |

**MỤC LỤC**

[**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN** 2](#_Toc35782111)

[**LỜI CẢM ƠN** 3](#_Toc35782112)

[**PHẦN 1. TỔNG QUAN VỀ ĐỀ TÀI** 7](#_Toc35782113)

[1. Đặt vấn đề 7](#_Toc35782114)

[2. Giải pháp 7](#_Toc35782115)

[3. Mục tiêu 8](#_Toc35782116)

[4. Công cụ sử dụng 8](#_Toc35782117)

[5. Bố cục của báo cáo này 8](#_Toc35782118)

[**PHẦN 2. TỔNG QUAN VỀ LINQ** 10](#_Toc35782119)

[1. Truy vấn dữ liệu và sự ra đời của LINQ 10](#_Toc35782120)

[2. LINQ là gì? 10](#_Toc35782121)

[3. Kiến trúc của LINQ 12](#_Toc35782122)

[4. LINQ hoạt động như thế nào? 13](#_Toc35782123)

[5. Ưu và nhược điểm của LINQ 15](#_Toc35782124)

[*a)* *Ưu điểm* 15](#_Toc35782125)

[*b)* *Nhược điểm* 16](#_Toc35782126)

[6. Các truy vấn cơ bản của LINQ 16](#_Toc35782127)

[*a)* *Lấy nguồn dữ liệu (datasource)* 16](#_Toc35782128)

[*b)* *Lọc dữ liệu (Filtering)* 16](#_Toc35782129)

[*c)* *Sắp xếp (Ordering)* 17](#_Toc35782130)

[*d)* *Gom nhóm (Grouping)* 18](#_Toc35782131)

[*e)* *Kết hợp (Joining)* 18](#_Toc35782132)

[7. LINQ Query và LINQ Method 19](#_Toc35782133)

[*a)* *LINQ Query Syntax* 19](#_Toc35782134)

[*b)* *LINQ Method Syntax* 20](#_Toc35782135)

[8. Chuyển đổi dữ liệu với LINQ 23](#_Toc35782136)

[*a)* *Kết hợp nhiều tập dữ liệu đầu vào thành 1 tập dữ liệu đầu ra* 24](#_Toc35782137)

[*b)* *Chuyển đổi các đối tượng thành XML* 25](#_Toc35782138)

[*c)* *Thực thi các tương tác trên dữ liệu nguồn* 26](#_Toc35782139)

[**PHẦN 3. CÁC THÀNH PHẦN CỦA LINQ** 28](#_Toc35782140)

[**PHẦN 4. SẢN PHẨM DEMO** 30](#_Toc35782141)

[**PHẦN 5. KẾT LUẬN** 31](#_Toc35782142)

# **PHẦN 1. TỔNG QUAN VỀ ĐỀ TÀI**

1. **Đặt vấn đề**

Dot Net (hay .NET) là một nền tảng (Framework) cho phép lập trình viên sử dụng để phát triển các ứng dụng, website. Nền tảng này được phát triển bởi Microsoft và chạy trên hệ điều hành Microsoft Window. Các ngôn ngữ được sử dụng trên nền tảng .NET gồm có C#, VB.Net. Hiện tại, số lượng sản phẩm phần mềm phát triển trên nền .NET cũng như nhu cầu nhân lực là vô cùng lớn.

Quy luật cơ bản trong lập trình là dữ liệu phải nằm trong bộ nhớ chính. Do đó khi cần dữ liệu chúng ta phải dùng cách nào đó để đưa dữ liệu vào bộ nhớ (đọc text file, truy vấn từ database ...). Trong môi trường .NET, dữ liệu trong bộ nhớ thường được thể hiện ở dạng các đối tượng và trước LINQ, chúng ta không có cách nào để móc nối các đối tượng hay thực hiện bất kỳ thao tác truy vấn nào.

Trước đây, cách phổ biến nhất để ứng dụng lấy dữ liệu từ các hệ cơ sở dữ liệu (CSDL) là sử dụng SQL (Structure Query Language - ngôn ngữ truy vấn cấu trúc). SQL có cú pháp rất khác nhau đối với từng ngôn ngữ lập trình phổ dụng như C# và VB.NET, do vậy lập trình viên phải nhọc công để kết nối và trích xuất dữ liệu khi tham gia phát triển phần mềm.

Một vấn đề khác với SQL là nó chỉ dùng để truy vấn dữ liệu trong các CSDL dạng quan hệ. Nếu muốn truy cập dữ liệu XML hay dạng khác (như trang HTML, email...), lập trình viên cần tìm hiểu và đối ứng nhiều cú pháp truy vấn khác nhau (XPath/XQuery).

Không dừng lại ở đó, trong quá trình phát triển phần mềm, lập trình viên còn phải đối mặt với nhiều vấn đề khác. Về vấn đề "Error", việc lập trình tương tác với CSDL tại cấp độ native language là bất khả thi. Vì thế lỗi thường khó phát hiện rõ và quản lý lỗi cũng gặp nhiều trở ngại.

1. **Giải pháp**

Để giảm gánh nặng thao tác trên nhiều ngôn ngữ khác nhau và cải thiện năng suất lập trình, để giải quyết những khó khăn lập trình viên gặp phải khi phát triển dựa trên nền tảng “con” của mình, Microsoft đã phát triển giải pháp tích hợp dữ liệu cho .NET Framework có tên gọi là LINQ (Language Integrated Query), đây là thư viện mở rộng cho các ngôn ngữ lập trình C# và Visual Basic.NET (có thể mở rộng cho các ngôn ngữ khác) cung cấp khả năng truy vấn trực tiếp dữ liệu Object, cơ sở dữ liệu và XML.

1. **Mục tiêu**

* Tìm hiểu tổng quan về ngôn ngữ tích hợp truy vấn LINQ.
* Tìm hiểu về cấu trúc, nguyên lý, cách hoạt động của LINQ.
* Tìm hiểu về những điểm mạnh và điểm yếu của LINQ để áp dụng trong trường hợp cần thiết.
* Tìm hiểu về các thành phần của LINQ, các toán tử và cú pháp của LINQ.
* Xây dựng ứng dụng demo sử dụng LINQ.

1. **Công cụ sử dụng**

* Microsoft Visual Studio 2017 (Công cụ lập trình).
* Microsoft Word (Công cụ soạn thảo).
* Microsoft SQL Server (Hệ quản trị cơ sở dữ liệu).

1. **Bố cục của báo cáo này**

Phần 1: Tổng quan về đề tài: Giới thiệu đề tài, vấn đề và cách giải quyết.

Phần 2: Tổng quan về LINQ: Tìm hiểu về LINQ, kiến trúc, cách hoạt động của LINQ.

Phần 3: Các thành phần của LINQ: Tìm hiểu về các thành phần mà LINQ có hỗ trợ.

Phần 4: Sản phẩm Demo: Ứng dụng LINQ vào làm phát triển phần mềm.

Phần 5: Tổng kết: Kết luận các kết quả đạt được, tài liệu tham khảo cũng như những hạn chế còn tồn đọng.

# **PHẦN 2. TỔNG QUAN VỀ LINQ**

1. **Truy vấn dữ liệu và sự ra đời của LINQ**

Một truy vấn là một diễn tả để lấy dữ liệu từ 1 nguồn dữ liệu nào đó. Các truy vấn thường được dùng trong các ngôn ngữ truy vấn đặc trưng. Các ngôn ngữ truy vấn khác nhau được phát triển khác nhau để áp dụng cho các nguồn dữ liệu khác nhau, ví dụ SQL dùng cho cơ sở dữ liệu quan hệ và XQuery dùng cho XML. Vì vậy, các nhà phát triển phải học các ngôn ngữ truy vấn cho mỗi dạng nguồn dữ liệu hoặc định dạng dữ liệu. Do đó, Microsoft mới nghĩ ra việc xây dựng 1 ngôn ngữ truy vấn nào đó mà có thể dùng cho tất cả nguồn dữ liệu, tất cả các định dạng dữ liệu khác nhau. Ngôn ngữ truy vấn đó là[**LINQ**](https://www.dammio.com/glossary/linq).

|  |
| --- |
| *Hình 1. Trích xuất dữ liệu cho ứng dụng trên nền .NET trước kia* |

1. **LINQ là gì?**

Để giảm gánh nặng thao tác trên nhiều ngôn ngữ khác nhau và cải thiện năng suất lập trình, Microsoft đã phát triển giải pháp tích hợp dữ liệu cho .NET Framework có tên gọi là **LINQ** (**Language Integrated Query**, tạm dịch là ***ngôn ngữ truy vấn tích hợp***), đây là thư viện mở rộng cho các ngôn ngữ lập trình C# và Visual Basic.NET (có thể mở rộng cho các ngôn ngữ khác) cung cấp khả năng truy vấn trực tiếp dữ liệu Object, CSDL và XML. LINQ là một tập hợp các thành phần mở rộng cho phép viết các câu truy vấn dữ liệu ngay trong một ngôn ngữ lập trình, như C# hoặc VB.NET.

|  |
| --- |
| *Hình 2. Tương tác giữa LINQ đến các nguồn dữ liệu khác nhau* |

Tóm lại, LINQ ra đời nhằm làm giảm công sức của lập trình viên cho những công việc đơn giản, lặp lại và chung chung trước đây, như trong ví dụ sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15. | class Program  {  static void Main(string[] args)  {  Student[] studentArray = {  new Student() { StudentID = 1, StudentName = "John", Age = 18},  new Student() { StudentID = 2, StudentName = "Steve", Age = 21},  new Student() { StudentID = 3, StudentName = "Bill", Age = 25},  new Student() { StudentID = 4, StudentName = "Ram" , Age = 20},  new Student() { StudentID = 5, StudentName = "Ron" , Age = 31},  new Student() { StudentID = 6, StudentName = "Chris", Age = 17},  new Student() { StudentID = 7, StudentName = "Rob",Age = 19 }  };  }  } | |

Giả sử có một mảng các đối tượng *Student* như trên, nếu sử dụng C# 2.0 để lọc ra các student có độ tuổi từ 12 đến 20, lập trình viên cần sử dụng *deligate* như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16. | class StudentExtension  {  public static Student[] where(Student[] stdArray, FindStudent del)  {  int i=0;  Student[] result = new Student[10];  foreach (Student std in stdArray)  if (del(std))  {  result[i] = std;  i++;  }  return result;  }  } | |

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04. | Student[] students = StudentExtension.where(studentArray,  delegate(Student std){  return std.Age > 12 && std.Age < 20;  });  } | |

Mọi việc sẽ đơn giản hơn nhờ vào LINQ, lập trình viên chỉ đơn giản sử dụng câu lệnh như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01. | Student[] teenAgerStudents = studentArray.Where(s => s.age > 12 &&  s.age < 20).ToArray(); | |

1. **Kiến trúc của LINQ**

|  |
| --- |
| *Hình 3. Kiến trúc của LINQ* |

Với kiến trúc 3 tầng như trên, LINQ cho phép truy vấn và chuyển đổi dữ liệu từ các nguồn:

* XML Documents
* SQL Databases.
* ADO.NET Datasets
* .NET Collections

1. **LINQ hoạt động như thế nào?**

|  |
| --- |
| *Hình 4. LINQ hoạt động như thế nào* |

Như mô tả trong sơ đồ trên, lập trình viên có thể viết các truy vấn LINQ bằng bất kỳ ngôn ngữ lập trình được hỗ trợ .NET nào như C #, VB.NET, J #,…

LINQ Provider là một thành phần nằm giữa các truy vấn LINQ và nguồn dữ liệu Datasource. LINQ Provider sẽ chuyển đổi các truy vấn LINQ thành định dạng có thể hiểu được bởi các datasource tương ứng. Ví dụ: LINQ Provider của LINQ to SQL sẽ chuyển đổi các truy vấn LINQ thành các câu lệnh SQL mà cơ sở dữ liệu SQL Server có thể hiểu được. Tương tự, LINQ Provider của LINQ to XML sẽ chuyển đổi các truy vấn thành định dạng mà tài liệu XML có thể hiểu được.

LINQ Provider là thành phần implement IQueryProvider và IQueryable Interface cho datasource cụ thể. Nói cách khác, nó cho phép chúng ta viết các truy vấn LINQ đối với một datasource nào đó. Không có LINQ Provider, các truy vấn LINQ không thể thực hiện được. Lập trình viên hoàn toàn có thể chủ động tạo ra các LINQ Provider tùy biến của riêng họ bằng cách implement IQueryProvider và IQueryable Interface cho datasource họ mong muốn.

Hoạt động truy vấn của LINQ bao gồm 3 tác vụ chính:

* Tạo kết nối với nguồn dữ liệu Datasource.
* Tạo truy vấn.
* Thực thi truy vấn.

Quá trình trên có thể được minh họa qua ví dụ sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22. | class IntroToLINQ  {  static void Main()  {  // The Three Parts of a LINQ Query:  // 1. Data source.  int[] numbers = new int[7] { 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);  }  }  } | |

1. **Ưu và nhược điểm của LINQ**
2. *Ưu điểm*

Lập trình viên không cần phải học các cú pháp ngôn ngữ truy vấn mới cho các nguồn dữ liệu khác nhau vì LINQ đã cung cấp cú pháp truy vấn chung để truy vấn các nguồn dữ liệu khác nhau.

Readable Code: LINQ làm cho mã dễ đọc hơn để các nhà phát triển có thể dễ dàng hiểu và duy trì.

Cách truy vấn được chuẩn hóa: Có thể sử dụng cùng một cú pháp LINQ để truy vấn nhiều nguồn dữ liệu.

Code ít hơn so với cách tiếp cận truyền thống. Nói cách khác, LINQ giúp làm giảm đáng kể lượng code cho sản phẩm phần mềm.

LINQ cung cấp cơ chế kiểm tra compile time error cũng như tương thích tốt với Visual Studio – công cụ lập trình các .NET phổ biến nhất hiện nay. Điều này giúp làm tăng năng suất của lập trình viên và đồng thời tránh xảy ra lỗi tốt hơn.

LINQ cung cấp nhiều phương thức sẵn có mà các lập trình viên có thể sử dụng để thực hiện các hoạt động khác nhau như filter (lọc), order (sắp xếp), group (gom nhóm), ...

Truy vấn của Lin có thể được tái sử dụng (re-use).

1. *Nhược điểm*

LINQ rất khó để viết các truy vấn phức tạp như SQL.

LINQ hiện không tận dụng tối đa các tính năng của SQL.

Nếu các lập trình viên sử dụng các truy vấn không đúng cách, hiệu suất của phần mềm sẽ rất thấp

Nếu các truy vấn bị thay đổi, lập trình viên cần biên dịch lại ứng dụng và cần triển khai lại dll cho máy chủ.

1. **Các truy vấn cơ bản của LINQ**
2. *Lấy nguồn dữ liệu (datasource)*

Trong truy vấn LINQ, bước đầu tiên là chỉ ra datasource. Một biến phải được khai báo trước khi có thể sử dụng nó. Trong truy vấn LINQ, mệnh đề ***from*** xuất hiện trước để giới thiệu datasource và biến phạm vi.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02. | var queryAllCustomers = from cust in customers  select cust; | |

Trong ví dụ trên, datasource được chỉ ra là *customers*, và biến phạm vi là *cust* – đại diện cho từng records bên trong *customers*.

1. *Lọc dữ liệu (Filtering)*

Bộ lọc làm cho truy vấn chỉ trả về các phần tử thỏa mãn điều kiện lọc. Trong linq, việc chỉ ra điều kiện được thực hiện trong mệnh đề *where*. Bộ lọc có hiệu lực xác định các phần tử cần loại trừ khỏi kết quả trả về. Trong ví dụ sau, chỉ những khách hàng có địa chỉ ở London mới được trả lại.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02. | var queryLondonCustomers = from cust in customers  where cust.City == "London"  select cust; | |

LINQ cũng hỗ trợ các toán tử ***AND*** và ***OR*** trong câu truy vấn của mình.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04. | var queryLondonCustomers = from cust in customers  where cust.City=="London"  && cust.Name == "Devon"  select cust; | |
|  |
| |  |  | | --- | --- | | 01.  02.  03.  04. | var queryLondonCustomers = from cust in customers  where cust.City == "London"  || cust.City == "Paris"  select cust; | |

Ngoài việc lọc theo thuộc tính, LINQ cũng hỗ trợ việc filter theo kiểu dữ liệu của đối tượng. Việc này được thực hiện bởi toán tử ***Oftype***. Ví dụ sau đây mô tả việc lọc các phần tử thuộc kiểu ***String*** của một **IList** bao gồm nhiều phần tử:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16. | IList mixedList = new ArrayList();  mixedList.Add(0);  mixedList.Add("One");  mixedList.Add("Two");  mixedList.Add(3);  mixedList.Add(new Student() { StudentID = 1, StudentName = "Bill" });  var stringResult = from s in mixedList.OfType<string>()  select s;  //Output: stringResult = [“One”,”Two”]  var intResult = from s in mixedList.OfType<int>()  select s;  //Output: intResult =[0,3] | |

1. *Sắp xếp (Ordering)*

Mệnh đề ***orderby*** sẽ làm cho các phần tử trong chuỗi được trả về được sắp xếp theo bộ so sánh mặc định cho loại được sắp xếp. Ví dụ: truy vấn sau sẽ có kết quả trả về được sắp xếp dựa trên thuộc tính ***Name***. Vì ***Name*** là một chuỗi, trình so sánh mặc định thực hiện sắp xếp theo thứ tự bảng chữ cái từ A đến Z.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04. | var queryLondonCustomers3 = from cust in customers  where cust.City == "London"  orderby cust.Name ascending  select cust; | |

Để sắp xếp các kết quả theo thứ tự ngược lại, từ Z đến A, sử dụng mệnh đề orderby… descending

1. *Gom nhóm (Grouping)*

Mệnh đề ***group*** cho phép lập trình viên nhóm kết quả trả về dựa trên thuộc tính được chỉ định. Ví dụ: truy vấn sau sẽ có kết quả trả về được góm nhóm theo thuộc tính ***City***.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03. | var queryCustomersByCity = from cust in customers  group cust by cust.City  select cust; | |

1. *Kết hợp (Joining)*

Tương tự như SQL, kết hợp (join) dữ liệu xảy ra giữa các tập đối tượng dữ liệu mà chưa được mô hình rõ ràng trong nguồn dữ liệu. Ví dụ, chúng ta tìm tất cả khách hàng (***customers***) và các nhà phân phối (***distributors***) ở cùng thành phố. Mệnh đề join trong LINQ cho phép kết hợp dữ liệu trên các tập đối tượng theo vì dùng bảng cơ sở dữ liệu trực tiếp.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07. | var innerJoinQuery = from cust in customers  join dist in distributors  on cust.City equals dist.City  select new {  CustomerName = cust.Name,  DistributorName = dist.Name  }; | |

Trong ví dụ trên, chúng ta join 2 tập dữ liệu ***customers*** và ***distributors*** trên thuộc tính ***City*** sau đó xuất ra tập dữ liệu chứa ***CustomerName*** và ***DistributorName***.

Trong LINQ, chúng ta không cần phải dùng mệnh đề join thường xuyên như trong SQL bởi vì các khóa ngoại trong LINQ thông thường được mô tả trong mô hình đối tượng (object model) như là các thuôc tính liên kết 1 tập các phần tử. Ví dụ, một đối tượng ***Customer*** chứa 1 tập các đối tượng ***Order***. Thay vì dùng join, chúng ta có thể truy xuất danh sách orders bằng dấu chấm.

1. **LINQ Query và LINQ Method**

Có hai cách cơ bản để viết truy vấn LINQ:

* Dùng cú pháp truy vấn LINQ (LINQ Query Syntax)
* Dùng cú pháp phương thức LINQ (LINQ Method Syntax)

1. *LINQ Query Syntax*

Cú pháp truy vấn tương tự như SQL cho cơ sở dữ liệu. Nó được định nghĩa trong code C # hoặc VB.

Cú pháp truy vấn LINQ bắt đầu bằng từ khóa ***from*** và kết thúc bằng từ khóa ***select***. Ví dụ sau là một truy vấn LINQ trả về một tập hợp các chuỗi có chứa một từ "Tutorials" dùng cú pháp LINQ query.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13. | IList<string> stringList = new List<string>() {  "C# Tutorials",  "VB.NET Tutorials",  "Learn C++",  "Java"  };  // LINQ Query Syntax  var result = from s in stringList  where s.Contains("Tutorials")  select s;  //Output: result = [“C# Tutorials”,”VB.NET Tutorials”] | |

Cấu trúc của LINQ query

|  |
| --- |
| *Hình 5. Cú pháp của LINQ Query* |

1. *LINQ Method Syntax*

LINQ Method sử dụng các phương thức mở rộng (extension method) được định nghĩ trong class **Enumerable** hoặc **Queryable**, tương tự như cách gọi phương thức mở rộng của bất kỳ lớp nào. Ví dụ sau là một truy vấn LINQ trả về một tập hợp các chuỗi có chứa một từ "Tutorials" dùng cú pháp LINQ method.

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11. | IList<string> stringList = new List<string>() {  "C# Tutorials",  "VB.NET Tutorials",  "Learn C++",  "Java"  };  // LINQ Method Syntax  var result = stringList.Where(s => s.Contains("Tutorials"));  //Output: result = [“C# Tutorials”,”VB.NET Tutorials”] | |

Cấu trúc của LINQ Method

|  |
| --- |
| *Hình 6. Cú pháp của LINQ Method* |

Như mô tả ở hình trên, cú pháp của LINQ Method bao gồm các phương thức mở rộng (extension method) và biểu thức Lambda.

Phương thức mở rộng trong ví dụ trên là phương thức **Where()**, được định nghĩa trong class **Enumerable**. Phương thức **Where()** là một delegate. Điều này có nghĩa là lập trình viên có thể truyền những tham số khác nhau để lọc kết quả trả về.

|  |
| --- |
| *Hình 7. Mô tả cấu trúc của phương thức where()* |

Một số toán tử phổ biến thường được sử dụng với LINQ Method:

|  |  |
| --- | --- |
| Toán tử | Mô tả |
| ElementAt | Trả về phần tử tại một vị trí được chỉ định trong collection |
| ElementAtOrDefault | Trả về phần tử tại một vị trí được chỉ định trong collection hoặc một giá trị mặc định nếu giá trị được chỉ định nằm ngoài phạm vi. |
| First | Trả về phần tử đầu tiên của collection hoặc phần tử đầu tiên thỏa mãn điều kiện trong collection. |
| FirstOrDefault | Trả về phần tử đầu tiên của collection hoặc phần tử đầu tiên thỏa mãn điều kiện trong collection hoặc một giá trị mặc định nếu giá trị được chỉ định nằm ngoài phạm vi. |
| Last | Trả về phần tử cuối cùng của collection hoặc phần tử cuối cùng thỏa mãn điều kiện trong collection. |
| LastOrDefault | Trả về phần tử cuối cùng của collection hoặc phần tử cuối cùng thỏa mãn điều kiện trong collection hoặc một giá trị mặc định nếu giá trị được chỉ định nằm ngoài phạm vi. |
| Single | Trả về phần tử duy nhất của collection hoặc phần tử duy nhất thỏa mãn điều kiện. |
| SingleOrDefault | Trả về phần tử duy nhất của collection hoặc phần tử duy nhất thỏa mãn điều kiện. Trả về giá trị mặc định nếu không có phần tử nào thỏa điều kiện tồn tại hoặc collection không chứa phần tử nào. |

Ví dụ:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.  45.  46.  47.  48.  49.  50.  51.  52.  53.  54.  55.  56.  57.  58.  59.  60.  61.  62.  63. | IList<int> intList = new List<int>() { 10, 21, 30,5, 45, 50, 87 };  IList<string> emptyList = new List<string>();  Console.WriteLine("1st Element: {0}",intList.ElementAt(0));  Console.WriteLine("3rd Element: {0}",intList.ElementAtOrDefault(2));  //1st Element: 10  //3rd Element: 30  Console.WriteLine("9th Element: {0}",intList.ElementAt(9));  Console.WriteLine("9th Element: {0}",intList.ElementAtOrDefault(9));  //10th Element: 0 – default int value  //Run-time exception: Index was out of range....  Console.WriteLine("1st Element: {0}", intList.First());  Console.WriteLine("1st Even Element: {0}", intList.  First(i => i % 2 == 0));  Console.WriteLine("1st Element: {0}", emptyList.First());  //1st Element: 10  //1st Even Element: 10  //1st Element: Sequence contains no elements…  Console.WriteLine("1st Element: {0}", intList.FirstOrDefault());  Console.WriteLine("1st Even Element: {0}", intList.  FirstOrDefault(i => i % 2 == 0));  Console.WriteLine("1st Element: {0}", emptyList.FirstOrDefault());  //1st Element: 10  //1st Even Element: 10  //1st Element:  Console.WriteLine("Last Element: {0}", intList.Last());  Console.WriteLine("Last Even Element: {0}",  intList.Last(i => i % 2 == 0));  Console.WriteLine("Last Element: {0}", emptyList.Last ());  //Last Element: 87  //Last Even Element: 50  //Last Element: Sequence contains no elements…  Console.WriteLine("Last Element in intList: {0}",  intList.LastOrDefault());  Console.WriteLine("Last Even Element in intList: {0}",  intList.LastOrDefault(i => i % 2 == 0));  Console.WriteLine("Last Element: {0}", emptyList.LastOrDefault());  //Last Element: 87  //Last Even Element: 50  //Last Element:  Console.WriteLine("The element which is less than 10 in intList: {0}",  intList.Single(i => i < 10));  Console.WriteLine("The element which is less than 10 in intList: {0}",    //The Element which is less than 10 in intList: 5  // The Element which is less than 10 in intList: 5 | |

1. **Chuyển đổi dữ liệu với LINQ**

LINQ không chỉ dùng để rút trích hay truy vấn dữ liệu mà còn là công cụ mạnh để chuyển đổi dữ liệu. Bằng cách dùng LINQ, bạn có thể sử dụng 1 tập dữ liệu nguồn làm đầu vào và thay đổi tập này theo nhiều cách để tạo ra các tập dữ liệu mới (đầu ra). Thậm chí là bạn có thay đổi các phần tử của tập dữ liệu nguồn bằng cách sắp xếp hoặc gom nhóm. Tuy nhiên, chức năng mạnh nhất của LINQ vẫn là khả năng tạo ra các dạng tập dữ liệu mới và được thực hiện bằng mệnh đề select. Một số ví dụ mà LINQ có thể thực hiện được trong việc chuyển đổi dữ liệu là:

* Trộn 2 tập dữ liệu đầu vào thành 1 tập đầu ra với dạng mới.
* Tạo tập dữ liệu đầu ra chỉ chứa 1 hoặc vài thuộc tính của mỗi phần tử trong tập dữ liệu nguồn.
* Tạo tập dữ liệu đầu ra với các phần tử chứa các kết quả thực thi trên tập dữ liệu nguồn.
* Tạo tập dữ liệu đầu ra theo 1 định dạng khác.

1. *Kết hợp nhiều tập dữ liệu đầu vào thành 1 tập dữ liệu đầu ra*

LINQ cho phép tạo một tập dữ liệu đầu ra chứa các phần tử tử nhiều tập dữ liệu đầu vào. Ví dụ sau mô tả cách kết hợp 2 cấu trúc dữ liệu với cùng 1 nguyên lý mà có thể áp dụng cho kết hợp dữ liệu từ XML, SQL hay các nguồn DataSet. Giả sử chúng ta có 2 class object:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13. | class Student   {       public string Name { get; set; }       public int ID { get; set; }       public string City { get; set; }   }     class Teacher   {       public string Name{ get; set; }       public int ID { get; set; }       public string City{ get; set; }   } | |

Trong hàm Main chúng ta tạo 2 danh sách kiểu List chứa danh sách Teacher và danh sách Student, sau đó ta tạo ra một list với kiểu dữ liệu hỗn hợp của hai class này:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36. | class DataTransformations      {          static void Main()          {  // Tạo danh sách sinh viên có kiểu List,  //với từng đối tượng kiểu Student  List<Student> students = new List<Student>()  {  new Student {Name="Svetlana", ID=111, City="Đà Lạt"},  new Student {Name="Claire", ID=112, City="Nha Trang"},  new Student {Name="Sven", ID=113, City="Hà Nội"}  };  // Tạo danh sách giáo viên cũng có kiểu List,  // với từng đối tượng kiểu Teacher  List<Teacher> teachers = new List<Teacher>()  {  new Teacher {Name="Ann", ID=945, City = "Tp.HCM"},  new Teacher {Name="Alex", ID=956, City = "Hà Nội"},  new Teacher {Name="Michiyo", ID=972, City = "Nha Trang"}  };  // Tạo câu truy vấn  var peopleInNhaTrang = (from student in students  where student.City == "Nha Trang"  select student.Name)  .Concat(from teacher in teachers  where teacher.City == "Nha Trang"  select teacher.Name);  Console.WriteLine("Các sinh viên và giáo viên sống ở Nha Trang:");  // Thực hiện truy vấn  foreach (var person in peopleInNhaTrang)  {  Console.WriteLine(person);  }  }      }  /\* Output:  Các sinh viên và giáo viên sống ở Nha Trang:  Claire  Michiyo  \*/ | |

1. *Chuyển đổi các đối tượng thành XML*

LINQ cho phép chuyển đổi dữ liệu giữa các cấu trúc đối tượng, cơ sở dữ liệu SQL, ADO.NET Dataset và XML hay các tài liệu với nhau một cách dễ dàng. Ví dụ sau mô tả cách chuyển đổi các cấu trúc đối tượng thành các phần tử XML:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25. | class XMLTransform  {  static void Main()  {  // tạo nguồn dữ liệu bằng cách dùng 1 tập hợp khởi tạo  // Lớp Student tương tự phần trên  List<Student> students = new List<Student>()  {  new Student {Name="Svetlana", ID=111},  new Student {Name="Claire", ID=112}  };  // Tạo câu truy vấn  var studentsToXML = new XElement("Root",  from student in students  select new XElement("student",  new XElement("Name", student.Name),  new XElement("ID", student.ID)  ) // kết thúc "student"  ); // kết thúc "Root"  // Thực thi câu truy vấn.  Console.WriteLine(studentsToXML);  }  } | |

Trong ví dụ trên, linq trích xuất data từ dữ liệu người là list ***students***, tạo 1 một thành phần Root mới và đổ dữ liệu vừa trích xuất được vào. Kết quả như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10. | <Root>  <student>  <Name>Svetlana</Name>  <ID>111</ID>  </student>  <student>  <Name>Claire</Name>  <ID>112</ID>  </student>  </Root> | |

1. *Thực thi các tương tác trên dữ liệu nguồn*

Tập dữ liệu đầu ra có thể không chứa bất kỳ phần tử hay thuộc tính nào của tập dữ liệu đầu vào, thay vào đó nó có thể chứa các giá trị dựa trên kết quả tính toán theo các giá trị đầu vào. Như trong ví dụ sau đây:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15. | class FormatQuery  {  static void Main()  {  // Nguồn dữ liệu  double[] listNumber = { 1, 2, 3 };  // Query.  IEnumerable<string> query =  from number in listNumber  select String.Format("Square = {0}", number \* number);  // Thực thi truy vấn  foreach (string s in query)  Console.WriteLine(s);  }  }  /\* Kết quả:  Square = 1  Square = 4  Square = 9  \*/ | |

# **PHẦN 3. CÁC THÀNH PHẦN CỦA LINQ**

1. LINQ to Object

LINQ to Object cung cấp việc sử dụng bất kỳ truy vấn LINQ nào hỗ trợ **IEnumerable<T>** để truy cập các data collection trong bộ nhớ mà không cần LINQ Provider (API) như trong trường hợp LINQ sang SQL hoặc LINQ sang XML. Các collection có thể do người dùng định nghĩa hoặc có thể được trả về bởi API .NET Framework.

Về cơ bản, LINQ to Object thể hiện một cách tiếp cận mới cho các collection. Theo cách cũ, lập trình viên phải viết các vòng lặp foreach phức tạp chỉ định cách lấy dữ liệu từ collection. Trong phương pháp LINQ, lập trình viên chỉ cần viết những đoạn code khai báo mô tả những gì họ muốn lấy.

Ngoài ra, các truy vấn LINQ cung cấp ba lợi thế chính so với các vòng lặp foreach truyền thống:

* Ngắn gọn và dễ đọc hơn, đặc biệt là khi lọc nhiều điều kiện.
* Cung cấp khả năng lọc, sắp xếp và gom nhóm mạnh mẽ với lượng code ngắn.
* Có thể được chuyển giữa các nguồn dữ liệu với nhau với rất ít hoặc không có sửa đổi.

Nói chung, thao tác trên dữ liệu càng phức tạp, cách sử dụng LINQ càng chứng minh được sự mạnh mẽ của nó so với các kỹ thuật lặp truyền thống.

Xét ví dụ sau, giả sử ta có object ***Department*** được định nghĩa như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22. | class Department  {  public int DepartmentId { get; set; }  public string Name { get; set; }  }  class LinqToObjects  {  static void Main(string[] args)  {  List<Department> departments = new List<Department>();  departments.Add(new Department  { DepartmentId = 1, Name = "Account" });  departments.Add(new Department  { DepartmentId = 2, Name = "Sales" });  departments.Add(new Department  { DepartmentId = 3, Name = "Marketing" });  }  } | |

Để lọc ra các *Department có Id là 2 hoặc 3,* ta cần viết vòng lặp như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14. | List<Department> departmentList = new List<Department>();  foreach (var dept in departments)  {  if (dept.DepartmentId == 2 || dept.DepartmentId == 3)  departmentList.Add(dept);  }  foreach (var dept in departmentList)  {  Console.WriteLine("Department Id = {0} , Department Name = {1}",  dept.DepartmentId, dept.Name);  } | |

Với LINQ to Object, ta chỉ đơn giản viết nó dưới một câu lệnh:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09. | var departmentList = from dept in departments  where dept.DepartmentId == 2 || dept.DepartmentId == 3  select dept;  foreach (var dept in departmentList)  {  Console.WriteLine("Department Id = {0} , Department Name = {1}",  dept.DepartmentId, dept.Name);  } | |

1. LINQ to XML

XML là một ngôn ngữ được ứng dụng rất nhiều trong lập trình, xử lý dữ liệu cũng như truyền tải thông tin. Trước đây việc thao tác và tạo ra các tài liệu XML thông qua DOM XML API trong .Net rất dài dòng và rắc rối. Vì thế để khắc phục điều này, Microsoft đã cho ra đời LINQ cùng với LINQ to XML API.

LINQ to XML là một interface trong bộ nhớ hỗ trợ LINQ cho phép lập trình viên làm việc với XML bằng các ngôn ngữ lập trình .NET Framework.

LINQ to XML giống như DOM (Document Object Model) ở chỗ nó đưa tài liệu XML vào bộ nhớ. Lập trình viên có thể truy vấn và sửa đổi tài liệu, và sau đó, có thể lưu nó vào một tệp và gửi nó qua Internet. Tuy nhiên, LINQ to XML khác với DOM: Nó cung cấp một mô hình đối tượng mới có trọng lượng nhẹ hơn và dễ làm việc hơn và tận dụng các tính năng ngôn ngữ trong C #.

Ưu điểm quan trọng nhất của LINQ to XML là tích hợp với Truy vấn tích hợp ngôn ngữ (LINQ). Việc tích hợp này cho phép bạn viết các truy vấn trên tài liệu XML trong bộ nhớ để truy xuất các collection các phần tử và thuộc tính. Khả năng truy vấn của LINQ to XML có thể so sánh về chức năng (mặc dù không theo cú pháp) với XPath và XQuery. Việc tích hợp LINQ trong C # cung cấp khả năng gõ mạnh hơn, kiểm tra thời gian biên dịch và hỗ trợ trình gỡ lỗi được cải thiện.

Ví dụ, để tạo file XML cơ bản như sau:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12. | <Blog>    <Title>YinYang's Programming Blog</Title>    <Author>      <FirstName>Yin</FirstName>      <LastName>Yang</LastName>    </Author>    <Categories>      <Category>LINQ</Category>      <Category>WPF</Category>      <Category>ADO.NET</Category>    </Categories>  </Blog> | |

Sử dụng cách cũ với DOM XML API (namespace System.Xml):

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34. | XmlDocument doc = new XmlDocument();  XmlElement blog = doc.CreateElement("Blog");  XmlElement title = doc.CreateElement("Title");  title.InnerText = "YinYang's Programming Blog";    XmlElement author = doc.CreateElement("Author");  XmlElement firstName = doc.CreateElement("FirstName");  firstName.InnerText = "Yin";  XmlElement lastName = doc.CreateElement("LastName");  lastName.InnerText = "Yang";    XmlElement categories = doc.CreateElement("Categories");  XmlElement category1 = doc.CreateElement("Category");  category1.InnerText = "LINQ";  XmlElement category2 = doc.CreateElement("Category");  category2.InnerText = "WPF";  XmlElement category3 = doc.CreateElement("Category");  category3.InnerText = "ADO.NET";    categories.AppendChild(category1);  categories.AppendChild(category2);  categories.AppendChild(category3);    author.AppendChild(firstName);  author.AppendChild(lastName);    blog.AppendChild(title);  blog.AppendChild(author);  blog.AppendChild(categories);    doc.AppendChild(blog); | |

Đoạn mã trên mặc dù dễ hiểu nhưng cú pháp lại quá dài dòng. Việc tạo ra một cấu trúc XML theo cách trên chỉ tập trung vào hai phương thức: ***CreateElement()*** để tạo một phần tử mới và ***AppendChild()*** để thêm phần tử con vào phần tử hiện tại. Nếu khối lượng data lớn, việc coding sẽ rất khó kiểm soát.

LINQ to XML cung cấp một cách tiếp cận hoàn toàn khác:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12. | XElement root =  new XElement("Blog",  new XElement("Title", "YinYang's Programming Blog"),  new XElement("Author",  new XElement("FirstName", "Yin"),  new XElement("LastName", "Yang")),  new XElement("Categories",  new XElement("Category", "LINQ"),  new XElement("Category", "WPF"),  new XElement("Category", "ADO.NET")  )  ); | |

Rất đơn giản và dễ hiểu, chỉ cần duy nhất một lớp XElement để tạo ra một tài liệu XML như trên.

1. LINQ to SQL

LINQ to SQL cung cấp một kiến trúc (run-time) để quản lý dữ liệu quan hệ dưới dạng đối tượng. Nó là một thành phần của .NET Framework 3.5 và thực hiện việc dịch các truy vấn tích hợp ngôn ngữ của mô hình đối tượng sang SQL. Các truy vấn này sau đó được gửi đến cơ sở dữ liệu để thực hiện. Sau khi có được kết quả từ cơ sở dữ liệu, LINQ to SQL lại dịch chúng sang các đối tượng.

|  |
| --- |
| *Hình 8. Kiến trúc của LINQ to SQL* |

LINQ to SQL cho phép truy vấn dữ liệu trong cơ sở dữ liệu SQL Server bằng cách sử dụng các biểu thức LINQ thông thường. Nó cũng cho phép cập nhật, xóa và chèn dữ liệu, nhưng nhược điểm duy nhất và lớn nhất của nó là chỉ hoạt động được với SQL Server. Tuy nhiên, có nhiều lợi ích của LINQ đối với SQL so với ADO.NET như giảm độ phức tạp, giảm mã nguồn và nhiều lợi ích khác.

Sử dụng LINQ to SQL:

* Tạo một kết nối tới Database

|  |
| --- |
| *Hình 9. Tạo Database Connection* |

* Tạo một LINQ to SQL class

|  |
| --- |
| *Hình 10. Tạo mới class LINQ to SQL* |

* Chọn bảng từ database và thả vào class LINQ to SQL

|  |
| --- |
| *Hình 11. Thêm table từ database vào LINQ to SQL class* |

Sau khi thực hiện các bước trên, LINQ to SQL đã xây dựng một đường kết nối giữa database tới các phần mềm. Để truy vấn dữ liệu, lập trình viên chỉ cần viết những câu query bằng LINQ (như mô tả ở Phần 2).

Ví dụ cho câu lệnh **Insert:**

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 01.  02.  03.  04.  05.  06.  07.  08.  09.  10.  11.  12.  13.  14.  15.  16.  17.  18. | //Chỉ ra Connection String  string connectString = System.Configuration.ConfigurationManager.  ConnectionStrings["LinqToSQLDBConnectionString"].ToString();  LinqToSQLDataContext db = new LinqToSQLDataContext(connectString);  //Tạo một nhân viên mới  Employee newEmployee = new Employee();  newEmployee.Name = "Michael";  newEmployee.Email = "yourname@companyname.com";  newEmployee.ContactNo = "343434343";  newEmployee.DepartmentId = 3;  newEmployee.Address = "Michael - USA";  //Thêm nhân viên vừa tạo vào database  db.Employees.InsertOnSubmit(newEmployee);  //Save changes  db.SubmitChanges(); | |

# **PHẦN 4. SẢN PHẨM DEMO**

# **PHẦN 5. KẾT LUẬN**