Language Integrated Query (LINQ)

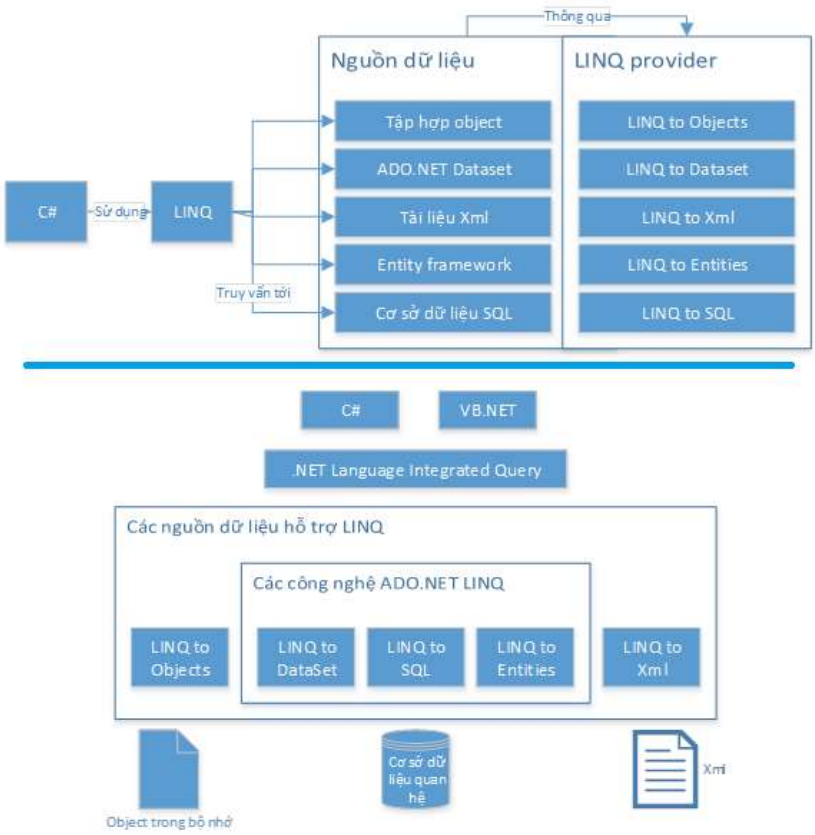
Để sử dụng LINQ cần có ba thành phần: nguồn dữ liệu (data source), truy vấn (query), lời gọi thực hiện truy vấn (query execution).

Nguồn dữ liệu

Các truy vấn LINQ trong C# đều tác động trên nguồn dữ liệu. Để người lập trình có thể sử dụng chung một cách thức viết code cho dù truy vấn từ nhiều loại nguồn dữ liệu khác nhau, LINQ luôn luôn làm việc với object. Do đó, đối với mỗi loại nguồn dữ liệu trong C# cần xây dựng thư viện hỗ trợ LINQ (LINQ provider) riêng giúp chuyển đổi dữ liệu về dạng object và ngược lại.

Vì lý do này, đối với dữ liệu Xml người ta phải xây dựng thêm provider "LINQ to Xml", với cơ sở dữ liệu SQL Server người ta phải xây dựng thêm provider "LINQ to SQL", v.v..

Hình minh họa dưới đây mô tả vai trò của LINQ trong quan hệ với ngôn ngữ lập trình C# và các nguồn dữ liệu.



Kiến trúc và vị trí của LINQ

Nguồn dữ liệu IEnumerable và IQueryable

Từ một góc nhìn khác, LINQ thực chất là một bộ thư viện phương thức mở rộngs (extension method) cho các class thực thi hai giao diện (interface) `IEnumerable` và `IQueryable`. Tất cả các class và interface cho LINQ đều đặt trong namespace `System.Linq`, vốn được sử dụng mặc định khi tạo mới file mã nguồn cho bất kỳ class nào.

Các phương thức mở rộng của LINQ được xây dựng trong hai class tĩnh `Enumerable` và `Queryable`.

Lớp `Enumerable` chứa các phương thức mở rộng dành cho các class thực thi giao diện `IEnumerable<T>`, bao gồm các kiểu dữ liệu quen thuộc nằm trong không gian tên `System.Collections.Generic` như `List<T>`, `Dictionary<TKey, TValue>` (ngoài ra còn có `SortedList<T>`, `Queue<T>`, `HashSet<T>`, `LinkedList<T>`, v.v.).

Lớp `Queryable` chứa các phương thức mở rộng dành cho các class thực thi giao diện `IQueryable<T>` (ví dụ Entity framework). Vì lý do này, tất cả các class thực thi các giao diện trên đều có thể trở thành nguồn dữ liệu.

Truy vấn LINQ

Cú pháp truy vấn và cú pháp phương thức

Có hai cách viết cho LINQ là cú pháp truy vấn (query syntax) và cú pháp phương thức (method syntax).

- Cú pháp truy vấn (Query Syntax): Cú pháp này nhìn giống như một truy vấn select SQL đảo ngược với từ khóa đầu tiên là *from*, kết thúc là *select*. Cú pháp này có hình thức khác biệt với code C# thông thường với một số từ khóa mới.
- Cú pháp phương thức (Method Syntax, còn gọi là Fluent): giống như cách gọi một phương thức mở rộng bình thường trên object của class. Đây lối viết cơ bản của LINQ.
- Cú pháp pha trộn (mixed syntax): thực tế là một số phương thức LINQ không được hỗ trợ ở dạng query syntax, khi đó người ta có thể sử dụng lối viết pha trộn của cả hai loại cú pháp.

Có một số điểm lưu ý khi lựa chọn cách viết:

1. một số phương thức chỉ viết được theo cú pháp phương thức, không viết được với cú pháp truy vấn;
2. cú pháp truy vấn sẽ được chuyển sang cú pháp phương thức ở giai đoạn biên dịch, vì vậy hai cách viết không có gì khác biệt về hiệu suất ở giai đoạn thực thi;
3. nếu quen thuộc với sử dụng biểu thức lambda, cú pháp phương thức dễ dàng sử dụng hơn và không đòi hỏi phải học thêm một lối viết mới;
4. việc lựa chọn mang tính cá nhân, tuy nhiên nên sử dụng thống nhất.

Lối viết này luôn yêu cầu cung cấp một hàm lambda để gọi đối với mỗi phần tử của chuỗi dữ liệu. Khi quen thuộc với việc sử dụng hàm lambda, lối viết này hoàn toàn giống như gọi hàm thông thường và không yêu cầu phải học thêm gì cả.

Cú pháp truy vấn nhìn giống như truy vấn SQL viết ngược lại. Lối viết này cung cấp một cách khác để mô tả hàm xử lý phần tử thay vì trực tiếp cung cấp hàm lambda làm tham số.

Trong bài học này bạn sẽ chỉ học cách sử dụng cú pháp phương thức. Nếu muốn, bạn có thể tự tìm hiểu cú pháp phương thức. Cú pháp phương thức giúp code "có tính C#" hơn.

Danh sách phương thức truy vấn LINQ

Danh sách phân loại các phương thức truy vấn được liệt kê trong bảng dưới đây:

- Nhóm: Lọc dữ liệu
  - Phương thức: Where, OfType
- Nhóm: Sắp xếp
  - Phương thức: OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
- Nhóm: Nhóm
  - Phương thức: GroupBy, ToLookup
- Nhóm: Ghép nối
  - Phương thức: GroupJoin, Join
- Nhóm: Phép chiếu
  - Phương thức: Select, SelectMany
- Nhóm: Phép gộp
  - Phương thức: Aggregate, Average, Count, LongCount, Max, Min, Sum
- Nhóm: Định lượng
  - Phương thức: All, Any, Contains
- Nhóm: Lấy phần tử
  - Phương thức: ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
- Nhóm: Tập hợp
  - Phương thức: Distinct, Except, Intersect, Union
- Nhóm: Phân đoạn
  - Phương thức: Skip, SkipWhile, Take, TakeWhile
- Nhóm: Ghép dữ liệu
  - Phương thức: Concat
- Nhóm: Đẳng thức
  - Phương thức: SequenceEqual
- Nhóm: Sinh dữ liệu
  - Phương thức: DefaultEmpty, Empty, Range, Repeat
- Nhóm: Biến đổi
  - Phương thức: AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList

Thông qua tên gọi chúng ta có thể hình dung sơ qua về khả năng của các phương thức.

Đặc biệt, nếu quen thuộc với ngôn ngữ SQL, nhiều phương thức trong danh sách trên có tên gọi và khả năng tương đồng với các lệnh của SQL.

Chúng ta tạm thời chưa cần hiểu rõ tất cả các phương thức trên. Trong phần tiếp theo chúng ta sẽ tìm hiểu cách sử dụng cụ thể của một số phương thức phổ biến nhất thông qua các ví dụ.

Thực thi truy vấn LINQ

Một truy vấn LINQ không được thực thi ngay khi chương trình thực hiện đến lệnh đó. Chỉ khi nào có những hoạt động thực sự cần đến dữ liệu từ truy vấn đó (ví dụ như duyệt danh sách dữ liệu), truy vấn mới được thực hiện. Việc trì hoãn thực hiện truy vấn như vậy có tên gọi là *thực thi trễ* (deferred execution).

Việc thực thi trễ có tác dụng lớn đến việc tăng hiệu suất xử lý vì nó hạn chế thực thi những lệnh chưa cần thiết. Cơ chế thực thi trễ áp dụng cho tất cả các nguồn dữ liệu hỗ trợ LINQ hiện tại (LINQ to Objects, LINQ to SQL, LINQ to Entities, LINQ to XML).

Trong một số trường hợp chúng ta cần thực thi truy vấn ngay khi chương trình chạy đến vị trí lệnh đó. Để thực hiện cơ chế này, chúng ta gọi tới một trong số các phương thức biến đổi dữ liệu bắt đầu bằng "To": ToArray, ToList.

Sử dụng một số phương thức LINQ

Ở phần trên chúng ta đã xem xét những vấn đề cơ bản để hiểu tư tưởng chung của LINQ. Trong phần này chúng ta sẽ xem một số ví dụ minh họa cụ thể. Trong các ví dụ, chúng ta sẽ sử dụng lớp thực thể Contact như sau:

```
private class Contact
{
    public int Age { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Address { get; set; }
}
```

Nguồn dữ liệu:

```
var contacts = new List<Contact>
{
    new Contact{ Age = 11, FirstName = "Trump", LastName = "Donald", Address = "Ha Noi"},
    new Contact{ Age = 21, FirstName = "Omaba", LastName = "Barrack", Address = "Sai Gon"},
    new Contact{ Age = 31, FirstName = "Bush", LastName = "George", Address = "Ha Noi"},
    new Contact{ Age = 41, FirstName = "Bill", LastName = "Clinton", Address = "Da Nang"},
    new Contact{ Age = 51, FirstName = "Reagan", LastName = "Ronald", Address = "Da Nang"},
    new Contact{ Age = 61, FirstName = "Jimmy", LastName = "Carter", Address = "Sai Gon"},
    new Contact{ Age = 71, FirstName = "Gerald", LastName = "Ford", Address = "Ha Noi"},
    new Contact{ Age = 81, FirstName = "Nixon", LastName = "Richard", Address = "Ha Noi"},
};
```

Sau đây chúng ta sẽ xem xét cách sử dụng một số phương thức cụ thể của LINQ thông qua các ví dụ.

Lưu ý sử dụng không gian tên `System.Linq`

Phương thức Where

`Where` là phương thức dùng để lọc dữ liệu theo các tiêu chí nào đó và trả về một danh sách dữ liệu khác (chỉ chứa những dữ liệu phù hợp tiêu chí). Chúng ta xem xét việc sử dụng `Where` qua một số ví dụ:

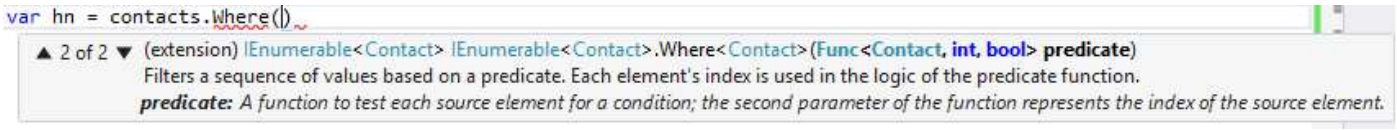
Yêu cầu 1: lập danh sách những người có địa chỉ là "Ha Noi".

Đây là yêu cầu lọc dữ liệu theo tiêu chí trường Address của object phải chứa giá trị "Ha Noi". Chúng ta sử dụng phương thức lọc Where trên nguồn dữ liệu contacts.

Như chúng ta thấy, Intellisense cung cấp mô tả của phương thức này với hai overload:

phương thức linq where trong C#

overload thứ nhất của phương thức Where



overload thứ hai của phương thức Where

overload thứ hai của phương thức Where

Overload thứ nhất tiếp nhận một biến thuộc kiểu delegate Func<Contact, bool>.

Overload thứ hai tương tự như vậy nhưng trong phương thức tham số có thêm một biến đầu vào kiểu int dùng để chứa index của phần tử.

Như ở phần trên đã giải thích, `Func<Contact, bool>` là kiểu delegate tương ứng với phương thức có đầu vào kiểu `Contact` (kiểu dữ liệu cơ sở của biến `contacts`) và trả về kết quả kiểu `bool`. Phương thức này sẽ được gọi trên từng phần tử của `contacts`.

Nếu phương thức trả về kết quả true thì phần tử đó sẽ được thêm vào danh sách kết quả. Kết quả trả về của phương thức `Where` là một biến thuộc kiểu `IEnumerable<Contact>`, vốn có thể truy xuất tương tự như mảng.

Bạn nên học cách đọc mô tả phương thức LINQ để có thể chủ động tìm hiểu cách sử dụng của các phương thức. Có hơn 40 phương thức LINQ. Trong bài giảng này chúng ta không thể hướng dẫn tất cả mà chỉ hướng dẫn cách đọc và cách học là chính. Khi cần sử dụng đến phương thức nào có thể tự học được.

Với yêu cầu trên chúng ta có thể viết truy vấn như sau:

```
var hn = contacts.Where(c => c.Address == "Ha Noi");
foreach (var c in hn) Console.WriteLine($"{c.FirstName} {c.LastName}");
```

Như đã biết, ở vị trí tham số này có thể cung cấp hàm lambda, phương thức cục bộ, phương thức instance, phương thức tĩnh, phương thức vô danh. Ở đây chúng ta sử dụng hàm lambda cho gọn.

Do truy vấn LINQ sử dụng cơ chế thực thi trễ, chỉ đến khi gọi lệnh duyệt danh sách, truy vấn trên mới được thực thi để trả dữ liệu về cho biến hn.

Yêu cầu 2: hãy lọc tất cả các object nằm ở vị trí lẻ

Theo yêu cầu này, chúng ta cần lọc ra tất cả các object có index 1, 3, 5, v.v.. Để thực hiện yêu cầu này, chúng ta sử dụng overload thứ hai của phương thức Where:

```
var odd = contacts.Where((c, i) => i % 2 != 0);
foreach (var c in odd) Console.WriteLine($"{c.FirstName} {c.LastName}");
```

Overload thứ hai có thêm tham số đầu vào thứ hai thuộc kiểu int. Tham số này chứa index của phần tử đang được duyệt. Khi có giá trị index chúng ta dễ dàng tính ra những phần tử ở vị trí lẻ.

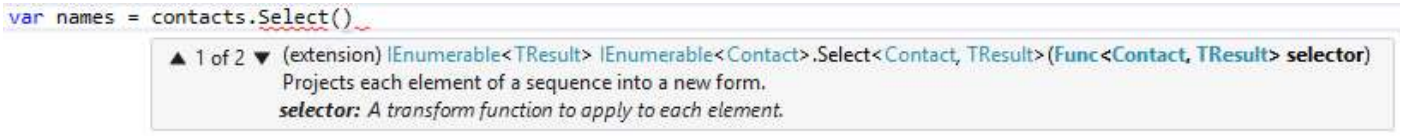
Phương thức Select

Phương thức Select có vai trò rất giống với truy vấn select của ngôn ngữ SQL. Nhiệm vụ của nó là biến đổi object từ dạng này sang dạng khác, gọi là "projection" – phép chiếu.

Yêu cầu 1: hãy tạo ra một danh sách họ tên của các contact (chứ không cần lấy trọn vẹn dữ liệu của mỗi contact) theo dạng: "Donald Trump", "Barrack Obama", v.v.

Chúng ta thấy dữ liệu theo yêu cầu không còn giống như dữ liệu gốc nữa. Chúng ta phải duyệt qua danh sách dữ liệu. Với mỗi phần tử phải trích lấy họ và tên rồi ghép lại với nhau tạo ra một object mới. Các object mới sẽ tập trung vào một danh sách riêng.

Phương thức Select được sử dụng cho những mục đích biến đổi như vậy:



truy vấn linq Select trong c#

Danh sách tham số của truy vấn Select

Khi đọc mô tả này có thể xác định: đầu vào của Select là một biến kiểu delegate `Func<Contact, TResult>` (gọi là selector – hàm chọn). Delegate này tương ứng với các phương thức có kiểu đầu vào là `Contact` (kiểu cơ sở của dữ liệu), kiểu đầu ra (`TResult`) do người lập trình tự xác định.

Phương thức tham số này có nhiệm vụ chuyển những dự liệu cần thiết từ biến kiểu Contact sang biến kiểu `TResult`. Kết quả thực hiện của phương thức Select là một danh sách object thuộc kiểu đích `TResult`.

```
var names = contacts.Select(c => $"{c.FirstName} {c.LastName}");
foreach (string name in names) Console.WriteLine(name);
```

Phương thức `Select` thường được sử dụng với kiểu vô danh và từ khóa `var` và có thể áp dụng lọc dữ liệu (với phương thức `Where`).

Yêu cầu 2: Hãy lấy ra danh sách họ tên của những contact có tuổi nhỏ hơn 50.

```
var youngs = contacts
    .Where(c => c.Age < 50)
    .Select(c => new { FullName = $"{c.FirstName} {c.LastName}", c.Age });
foreach (var c in youngs)
{
    Console.WriteLine($"Full name: {c.FullName}; Age: {c.Age}");
}
```

Ở đây, chúng ta áp dụng phương thức Where để lọc lấy những contact có tuổi dưới 50. Trong danh sách này, chúng ta lại áp dụng projection để tạo ra kết quả là danh sách của các object thuộc kiểu vô danh chứa hai thuộc tính FullName và Age, được tạo thành từ dữ liệu của mỗi contact.

Qua hai hướng dẫn trên chúng ta thấy, việc học cách sử dụng các phương thức LINQ là tương đối đơn giản nếu như nắm chắc các khái niệm thường dùng. Bạn có thể tự mình tìm hiểu thêm cách sử dụng các phương thức khác bằng cách đọc mô tả của phương thức như đã thực hiện ở các ví dụ trên.