Cải tiến repository: LINQ (Language Integrated Query)

Trong bài học này chúng ta sẽ xem xét cách sử dụng một công cụ khác của .NET framework để xử lý dữ liệu: LINQ (Language Integrated Query). Chúng ta sẽ vận dụng LINQ để cải tiến các phương thức truy xuất dữ liệu của lớp Repository giúp đơn giản hóa code và tăng hiệu suất của ứng dụng.

NỘI DUNG CỦA BÀI [Ẩn]

Đặt vấn đề

Ở các bài đầu khi xây dựng lớp Repository để xử lý dữ liệu chúng ta đã xây dựng các phương thức sau:

```
1.  public Book Select(int id)
2.  {
3.      foreach (var b in _context.Books)
4.      {
5.          if (b.Id == id) return b;
6.      }
7.      return null;
8.  }
9.
10. public Book[] Select(string key)
11. {
12.     var temp = new List<Book>();
13.     var k = key.ToLower();
14.     foreach (var b in _context.Books)
15.     {
16.         var logic =
17.             b.Title.ToLower().Contains(k) ||
18.             b.Authors.ToLower().Contains(k) ||
19.             b.Publisher.ToLower().Contains(k) ||
20.             b.Tags.ToLower().Contains(k) ||
21.             b.Description.ToLower().Contains(k)
22.             ;
23.         if (logic) temp.Add(b);
24.     }
25.     return temp.ToArray();
26. }
27.
28. public Book[] SelectMarked()
29. {
30.     var temp = new List<Book>();
31.     foreach (var b in _context.Books)
32.     {
33.         if (b.Reading) temp.Add(b);
34.     }
35.     return temp.ToArray();
```

```
36.  }
37.  public void Insert(Book book)
38.  {
39.      var lastIndex = _context.Books.Count - 1;
40.      var id = lastIndex < 0 ? 1 : _context.Books[lastIndex].Id + 1;
41.      book.Id = id;
42.      _context.Books.Add(book);
43.  }
```

Trong đó phương thức `Select` lựa chọn một cuốn sách từ kho theo giá trị Id. Một overload khác của Select lựa chọn tất cả những cuốn sách mà tiêu đề, tác giả, nhà xuất bản, tags và mô tả chứa một từ khóa nhất định. Phương thức `SelecteMarked` lựa chọn những cuốn sách mà trường Reading có giá trị true.

Cả 3 phương thức này đều có đặc điểm chung là phải dò tìm trong danh sách dữ liệu và trích ra những object đáp ứng yêu cầu. Chúng ta cũng có thể thấy là logic của các phương thức này rất giống nhau: (1) duyệt danh sách dữ liệu => (2) ứng với mỗi object check xem có thỏa mãn yêu cầu hay không => (3) nếu thỏa mãn, trả lại object này.

Nhu cầu viết những phương thức như vậy rất phổ biến khi xây dựng các ứng dụng quản lý. Để hỗ trợ người lập trình giải quyết những bài toán tương tự, .NET framework 3.5 (C# 3) đưa vào một bộ thư viện mở rộng có tên gọi *LINQ* (Language Integrated Query).

LINQ là bộ thư viện của các phương thức mở rộng cung cấp thêm khả năng xử lý dữ liệu dạng tập hợp. LINQ cung cấp các phương thức mở rộng (extension method) cho tất cả các kiểu dữ liệu tập hợp mà ta đã biết. Các phương thức của LINQ tập trung chủ yếu vào việc truy xuất dữ liệu từ các object của dữ liệu tập hợp (như mảng, List, Dictionary, v.v.). Vì lý do này, các phương thức của LINQ cũng thường được gọi là *truy vấn LINQ* (LINQ query).

Như trong trường hợp 3 phương thức của lớp Repository, chúng ta chỉ cần sử dụng một truy vấn LINQ là có thể chọn ra 1 object hoặc một mảng object đáp ứng các yêu cầu, thay vì phải duyệt danh sách và tự kiểm tra từng object như trên.

Thực hành 1: Cải tiến lớp Repository sử dụng LINQ

Bước 1. Bổ sung không gian tên System.Linq trong lớp Repository

```
1.  using System.Linq;
```

Trên thực tế, khi xây dựng một class mới, không gian tên `System.Linq` luôn được Visual Studio thêm vào một cách tự động. Nếu chưa thấy không gian tên này trong khối *using* thì bổ sung vào.

Bước 2. Điều chỉnh các phương thức như sau của lớp Repository:

```
1.  public Book Select(int id)
2.  {
3.      return _context.Books.FirstOrDefault(b => b.Id == id);
4.  }
5.
6.  public Book[] Select(string key)
7.  {
8.      var k = key.ToLower();
9.      return _context.Books.Where(b =>
10.             b.Title.ToLower().Contains(k) ||
11.             b.Authors.ToLower().Contains(k) ||
```

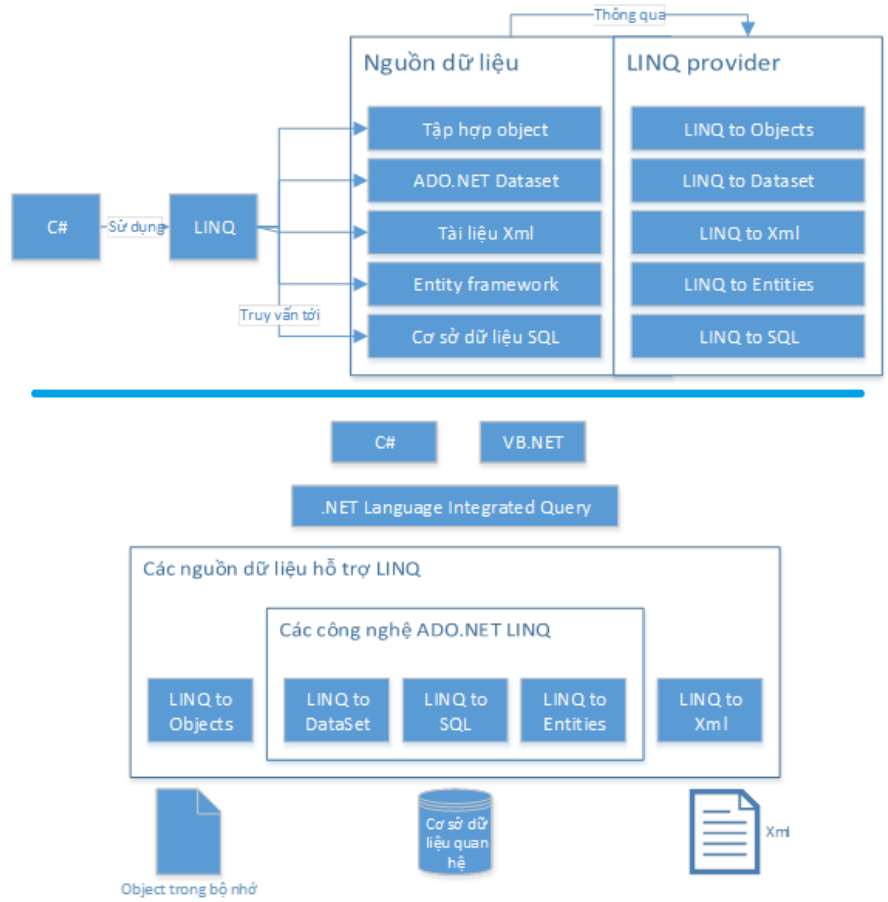
```
            b.Publisher.ToLower().Contains(k) ||
            b.Tags.ToLower().Contains(k) ||
            b.Description.ToLower().Contains(k)).ToArray();
    }

    public Book[] SelectMarked()
    {
        return _context.Books.Where(b => b.Reading == true).ToArray();
    }
    public void Insert(Book book)
    {
        var id = _context.Books.Count == 0 ? 1 : _context.Books.Max(b => b.Id) + 1;
        book.Id = id;
        _context.Books.Add(book);
    }
```

Ở phần thực hành này chúng ta đã thay thế các lệnh thông thường bằng truy vấn LINQ với hiệu quả tương tự. Khối lượng code đã được giảm đi đáng kể.

Cách áp dụng LINQ trong project

Như bạn đã biết, 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). Hình minh họa dưới đây mô tả vai trò của LINQ trong quan hệ với ngôn ngữ lập trình và các nguồn dữ liệu.



Kiến trúc và vị trí của LINQ

Trong phần thực hành trên, **nguồn dữ liệu** là `_context.Books` mà chúng ta đã có sẵn. Đây là một tập hợp các object trong bộ nhớ thuộc kiểu `List<Book>` (thực thi giao diện `IEnumerable<Book>`). **Provider** cho nó là *LINQ to Objects*.

Trong lớp `Repository` ở trên chúng ta chỉ sử dụng LINQ to Objects. LINQ to Objects cho phép sử dụng LINQ với các loại dữ liệu tập hợp quen thuộc mà chúng ta đã sử dụng trong dự án như `List<T>`, `Dictionary<TKey, TValue>`, mảng.

Các lệnh đã sử dụng ở phần thực hành

```
_context.Books.Where(b =>
            b.Title.ToLower().Contains(k) ||
            b.Authors.ToLower().Contains(k) ||
            b.Publisher.ToLower().Contains(k) ||
            b.Tags.ToLower().Contains(k) ||
            b.Description.ToLower().Contains(k))
_context.Books.Where(b => b.Reading == true)
```

là **truy vấn** được viết theo *cú pháp phương thức*. 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.

Các truy vấn trên nếu viết theo *cú pháp truy vấn* sẽ có dạng như sau:

```
from b in _context.Books
where b.Title.ToLower().Contains(k) ||
      b.Authors.ToLower().Contains(k) ||
      b.Publisher.ToLower().Contains(k) ||
      b.Tags.ToLower().Contains(k) ||
      b.Description.ToLower().Contains(k)
select b

from b in _context.Books
where b.Reading == true
select b
```

Lưu ý rằng, 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`.

Trong phần thực hành bên trên chúng ta đã sử dụng cơ chế thực thi này:

```
public Book[] Select(string key)
{
    var k = key.ToLower();
    return _context.Books.Where(b =>
            b.Title.ToLower().Contains(k) ||
            b.Authors.ToLower().Contains(k) ||
            b.Publisher.ToLower().Contains(k) ||
            b.Tags.ToLower().Contains(k) ||
            b.Description.ToLower().Contains(k)).ToArray();
}

public Book[] SelectMarked()
{
```

```
        return _context.Books.Where(b => b.Reading == true).ToArray();
    }
```

Thực hành 2: Bổ sung chức năng thống kê

Trong phần thực hành này chúng ta sẽ vận dụng LINQ để xây dựng một chức năng thống kê đơn giản: in danh sách theo nhóm.

Bước 1. Bổ sung phương thức Stats vào lớp Repository

```
public IEnumerable<IGrouping<string, Book>> Stats(string key = "folder")
{
    return _context.Books.GroupBy(b => System.IO.Path.GetDirectoryName(b.File));
}
```

Ở bước này chúng ta vận dụng phương thức GroupBy của LINQ để nhóm dữ liệu sách theo tên thư mục chứa file. Cấu trúc

```

```

làm nhiệm vụ trích phần thông tin về thư mục của file sách (phương thức GetDirectoryName), sau đó gom tất cả dữ liệu sách mà file của nó nằm trong cùng một thư mục vào một nhóm. Kết quả thực hiện của truy vấn này là một danh sách nhóm, trong đó tên của mỗi nhóm là tên một thư mục, phần tử của mỗi nhóm là tất cả các cuốn sách nằm trong thư mục đó.

Bước 2. Xây dựng lớp view mới BookStatsView trong thư mục Views

```
using BookMan.ConsoleApp.Models;
using Framework;
using System;
using System.Collections.Generic;
using System.Linq;

namespace BookMan.ConsoleApp.Views
{
    internal class BookStatsView : ViewBase<IEnumerable<IGrouping<string, Book>>>
    {
        public BookStatsView(IEnumerable<IGrouping<string, Book>> model) : base(model)
        {
        }

        public override void Render()
        {
            foreach (var g in Model)
            {
                ViewHelp.WriteLine($"# {g.Key}", ConsoleColor.Magenta);
                foreach (var b in g)
                {
                    ViewHelp.Write($"[{b.Id}] ", ConsoleColor.Yellow);
                    ViewHelp.WriteLine(b.Title, b.Reading ? ConsoleColor.Cyan: ConsoleCo
                }
            }
        }
    }
}
```

Ở đây để ý rằng, dữ liệu controller cung cấp cho view có kiểu IEnumerable<IGrouping<string, Book>>.

Để duyệt kiểu dữ liệu này chúng ta phải sử dụng hai vòng lặp: vòng lặp thứ nhất để duyệt danh sách nhóm, vòng lặp thứ hai để duyệt danh sách phần tử trong mỗi nhóm.

Thông tin nhóm được truy xuất qua thuộc tính Key, trong trường hợp này thuộc kiểu string. Tự bản thân mỗi nhóm là một danh sách các phần tử kiểu Book, do đó, có thể truy xuất như một danh sách bình thường.

Bước 3. Bổ sung phương thức vào lớp BookController

```
1.  public void Stats()
2.  {
3.      var model = Repository.Stats();
4.      var view = new BookStatsView(model);
5.      Render(view);
6.  }
```

Bước 4. Bổ sung route sau vào ConfigRouter

```
1.  r.Register(route: "show stats",
2.      action: p => controller.Stats(),
3.      help: "[show stats]");
```

Bước 5. Dịch và chạy thử chương trình với lệnh “show stats”

Kết quả thực hiện lệnh `show stats`

Chúng ta vừa bổ sung chức năng thống kê sách theo thư mục. Lệnh “show stats” sẽ hiển thị các cuốn sách theo từng thư mục. Theo logic này chúng ta hoàn toàn có thể xây dựng các tính năng thống kê theo các tiêu chí khác như tác giả, nhà xuất bản, năm xuất bản.

Kết luận

Trong bài này chúng ta đã xem xét cách sử dụng bộ thư viện LINQ và vận dụng vào việc điều chỉnh lớp Repository, cũng như xây dựng thêm tính năng thống kê dữ liệu dựa trên LINQ.

Đây cũng là bài gần kết thúc của chuỗi bài tự học lập trình C# qua dự án mini. Đến giai đoạn này, tất cả các chức năng theo phân tích đã hoàn thành. Trong bài tiếp theo chúng ta sẽ xem xét cách xuất bản và triển khai ứng dụng đến người dùng cuối.

> + Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
> + Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
> + Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
> Cảm ơn bạn!