Repository và quản lý dữ liệu: generic collection List

Trong bài này chúng ta sẽ học và vận dụng kỹ thuật lập trình tổng quát (generic) và phương thức tổng quát để xây dựng class quản lý dữ liệu. Các ứng dụng quản lý (nói chung) thường sử dụng một (nhóm) class để quản lý tập trung việc truy xuất dữ liệu. Mô hình quản lý dữ liệu tập trung như vậy có tên gọi là Repository. Trong dự án này, chúng ta sẽ vận dụng hình thức đơn giản của mô hình này giúp việc quản lý dữ liệu hiệu quả hơn. Ngoài ra, bạn sẽ học cách sử dụng lớp List<T> để lưu trữ danh sách object dữ liệu thay cho mảng.

NỘI DUNG CỦA BÀI [Ẩn]

Trong các bài trước chúng ta đã xây dựng lớp thực thể để mô tả dữ liệu cần quản lý. Chúng ta cũng xây dựng các lớp giao diện để hiển thị dữ liệu theo các hình thức khác nhau cùng các lớp cho phép người dùng tương tác với dữ liệu.

Tuy nhiên chúng ta chưa thực sự quản lý được dữ liệu. Chúng ta mới chỉ tạo ra một số dữ liệu thử nghiệm trực tiếp trong phương thức của controller.

Cách thức tạo ra và xử lý dữ liệu này không phù hợp với một ứng dụng quản lý, cụ thể:

1. Dữ liệu chỉ tồn tại khi chương trình hoạt động: mọi thay đổi trên dữ liệu sẽ mất đi khi đóng chương trình;
2. Dữ liệu tạo ra ở dạng thức rời rạc: dữ liệu chỉ phục vụ riêng cho mỗi phương thức nhằm mục đích thử nghiệm hoạt động của các view;
3. Chưa thể thực hiện các thao tác quản lý thông thường với dữ liệu: chưa thực hiện được việc thêm, sửa, xóa, cập nhật;
4. Chưa đảm bảo được khả năng thay thế nguồn dữ liệu: cụ thể, cần phải có khả năng thay đổi giữa dữ liệu thử nghiệm là danh sách các object trong bộ nhớ, truy xuất dữ liệu từ file, qua dịch vụ mạng hoặc từ cơ sở dữ liệu quan hệ.

Mô hình repository

Đối với các ứng dụng quản lý chúng ta phải tổ chức code quản lý dữ liệu đảm bảo các yêu cầu: tập trung, có khả năng lưu trữ lâu dài, dễ dàng thực hiện hoặc bổ sung các thao tác quản lý (như lọc, nhóm dữ liệu, v.v.), dễ dàng thay đổi cách thức truy xuất dữ liệu.

Do đó, chúng ta phải xây dựng các lớp riêng để hỗ trợ quản lý dữ liệu.

Thành phần hỗ trợ quản lý dữ liệu như vậy không thuộc về kiến trúc MVC mà có liên quan đến một *mô hình thiết kế* (design pattern) được gọi là *mô hình repository*. Mô hình này được sử dụng đặc biệt phổ biến với các ứng dụng quản lý.

Repository giúp tách rời phần xử lý/truy xuất dữ liệu khỏi giao diện người dùng. Do đó, có thể tái sử dụng repository trong nhiều dự án khác nhau, cho nhiều loại ứng dụng khác nhau về giao diện người dùng.

Trong bài này chúng ta sẽ cùng xây dựng một nhóm class, gọi chung là dịch vụ dữ liệu (data service) cho mục đích vừa nêu trên, cụ thể hơn:

1. Chứa code để quản lý danh sách thực thể (sách/giá sách) với các thao tác xử lý trên dữ liệu (thêm, sửa, xóa, lọc, lưu trữ, truy xuất, v.v.);
2. Chứa code để thực hiện truy xuất đến các nguồn lưu trữ dữ liệu dài hạn (như file, cơ sở dữ liệu quan hệ).

Thực hành: xây dựng repository đơn giản

Bước 1. Xây dựng lớp SimpleDataAccess

Tạo thêm thư mục DataServices trong dự án BookMan.ConsoleApp. Tạo thêm lớp SimpleDataAccess trong thư mục mới này (trong file mã nguồn SimpleDataAccess.cs) và code như sau:

```
using System.Collections.Generic;

namespace BookMan.ConsoleApp.DataServices
{
    using Models;

    public class SimpleDataAccess
    {
        public List<Book> Books { get; set; }

        public void Load()
        {
            Books = new List<Book>
            {
                new Book{Id=1, Title = "A new book 1"},
                new Book{Id=2, Title = "A new book 2"},
                new Book{Id=3, Title = "A new book 3"},
                new Book{Id=4, Title = "A new book 4"},
                new Book{Id=5, Title = "A new book 5"},
                new Book{Id=6, Title = "A new book 6"},
                new Book{Id=7, Title = "A new book 7"},
                new Book{Id=8, Title = "A new book 8"},
                new Book{Id=9, Title = "A new book 9"},
            };
        }

        public void SaveChanges() { }
    }
}
```

Ở đây bạn đã sử dụng kiểu dữ liệu danh sách **List<T>**. Đây là một kiểu thuộc nhóm *generic collection*.

Bước 2. Xây dựng lớp Repository

Tạo file mã nguồn Repository.cs trong thư mục DataServices cho lớp Repository và viết code như sau:

```
using System.Collections.Generic;

namespace BookMan.ConsoleApp.DataServices
{
    using Models;

    public class Repository
    {
        protected readonly SimpleDataAccess _context;

        public Repository(SimpleDataAccess context)
        {
            _context = context;
            _context.Load();
        }

        public void SaveChanges() => _context.SaveChanges();

        public List<Book> Books => _context.Books;

        public Book[] Select() => _context.Books.ToArray();

        public Book Select(int id)
        {
            foreach (var b in _context.Books)
            {
                if (b.Id == id) return b;
            }
            return null;
        }

        public Book[] Select(string key)
        {
            var temp = new List<Book>();
            var k = key.ToLower();
            foreach (var b in _context.Books)
            {
                var logic =
                    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)
                    ;
                if (logic) temp.Add(b);
            }
            return temp.ToArray();
        }

        public void Insert(Book book)
        {
            var lastIndex = _context.Books.Count - 1;
            var id = lastIndex < 0 ? 1 : _context.Books[lastIndex].Id + 1;
            book.Id = id;
            _context.Books.Add(book);
        }

        public bool Delete(int id)
        {
            var b = Select(id);
            if (b == null) return false;
            _context.Books.Remove(b);
            return true;
        }

        public bool Update(int id, Book book)
        {
            var b = Select(id);
```

```
            if (b == null) return false;
            b.Authors = book.Authors;
            b.Description = book.Description;
            b.Edition = book.Edition;
            b.File = book.File;
            b.Isbn = book.Isbn;
            b.Publisher = book.Publisher;
            b.Rating = book.Rating;
            b.Reading = book.Reading;
            b.Tags = book.Tags;
            b.Title = book.Title;
            b.Year = book.Year;
            return true;
        }
    }
}
```

Bước 4. Điều chỉnh BookController để sử dụng Repository

```
namespace BookMan.ConsoleApp.Controllers
{
    using DataServices;
    using Views;
    using Models;

    /// <summary>
    /// lớp điều khiển, giúp ghép nối dữ liệu sách với giao diện
    /// </summary>
    internal class BookController
    {
        protected Repository Repository;

        public BookController(SimpleDataAccess context)
        {
            Repository = new Repository(context);
        }

        /// <summary>
        /// ghép nối dữ liệu 1 cuốn sách với giao diện hiển thị 1 cuốn sách
        /// </summary>
        /// <param name="id">mã định danh của cuốn sách</param>
        public void Single(int id)
        {
            // lấy dữ liệu qua repository
            var model = Repository.Select(id);
            // khởi tạo view
            BookSingleView view = new BookSingleView(model);
            // gọi phương thức Render để thực sự hiển thị ra màn hình
            view.Render();
        }

        /// <summary>
        /// kích hoạt chức năng nhập dữ liệu cho 1 cuốn sách
        /// </summary>
        public void Create()
        {
            BookCreateView view = new BookCreateView();// khởi tạo object
            view.Render(); // hiển thị ra màn hình
        }

        /// <summary>
        /// kích hoạt chức năng hiển thị danh sách
        /// </summary>
        public void List()
        {
            // lấy dữ liệu qua repository
            var model = Repository.Select();
            // khởi tạo view
            BookListView view = new BookListView(model);
            view.Render();
        }

        /// <summary>
        /// kích hoạt chức năng cập nhật
        /// </summary>
        /// <param name="id"></param>
        public void Update(int id)
        {
            // lấy dữ liệu qua repository
```

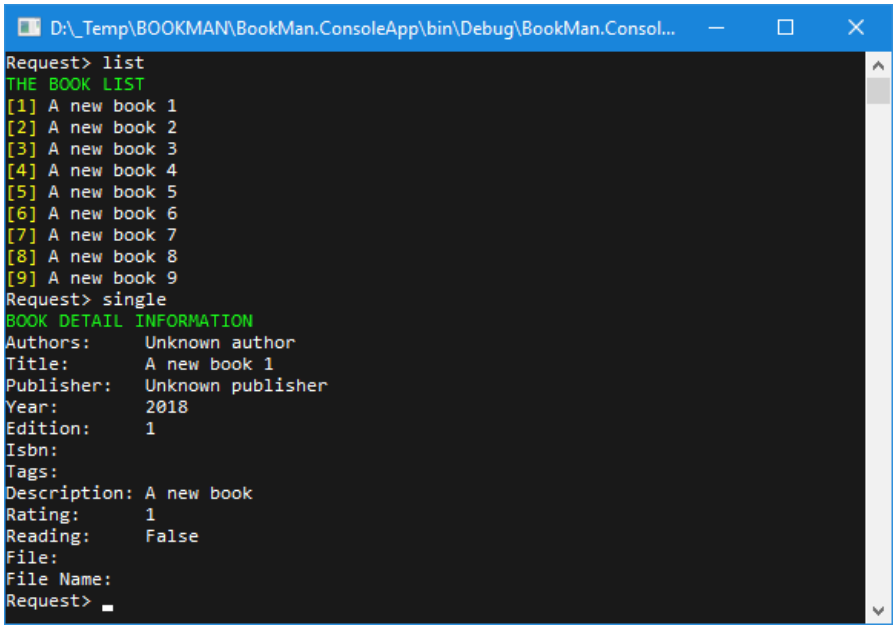
```
            var model = Repository.Select(id);
            var view = new BookUpdateView(model);
            view.Render();
        }
    }
}
```

Bước 5. Điều chỉnh phương thức Main

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BookMan.ConsoleApp
{
    using Controllers;
    using DataServices;

    internal class Program
    {
        private static void Main(string[] args)
        {
            SimpleDataAccess context = new SimpleDataAccess();
            BookController controller = new BookController(context);
            while (true)
            {
                Console.Write("Request> ");
                string request = Console.ReadLine();

                switch (request.ToLower())
                {
                    case "single":
                        controller.Single(1);
                        break;

                    case "create":
                        controller.Create();
                        break;

                    case "update":
                        controller.Update(1);
                        break;

                    case "list":
                        controller.List();
                        break;

                    default:
                        Console.WriteLine("Unknown command");
                        break;
                }
            }
        }
    }
}
```

Bước 5. Dịch và chạy thử chương trình

Chạy thử với các lệnh đã có (single, list, update):



Kết quả thực hiện chương trình

> Mời bạn tham khảo bài viết chi tiết hơn về lập trình generic trong C#.

Repository và sơ đồ class tổng thể

Repository là một kiểu thiết kế class giúp tách rời code của chương trình ứng dụng (tổ chức theo mô hình MVC) với phần xử lý và truy xuất dữ liệu. Theo đó, tất cả các thao tác xử lý dữ liệu cần sử dụng trong ứng dụng đều được tập trung ở repository.

Bản thân repository sau đó cũng gọi đến một class chuyên dụng để tải/lưu dữ liệu (ví dụ, vào file, cơ sở dữ liệu quan hệ, dịch vụ dữ liệ). Repository cho phép dễ dàng chuyển đổi giữa các nguồn dữ liệu khác nhau (ví dụ, để test hoặc để chạy thử ứng dụng).

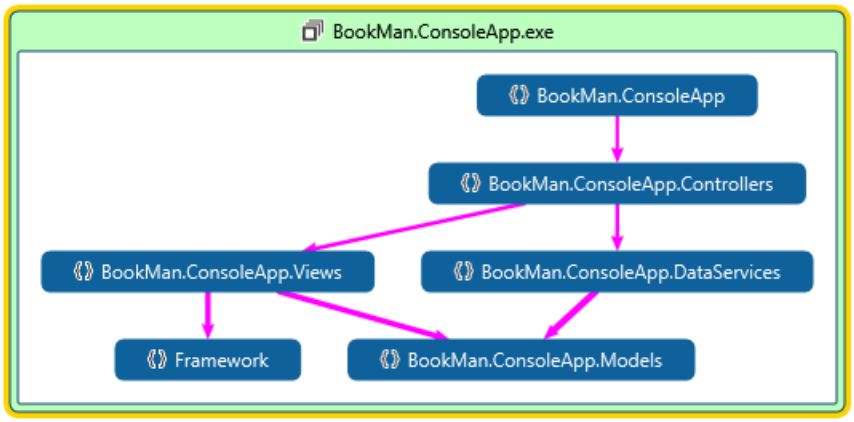
Như phần đầu đã nói, repository không phải là một phần của kiến trúc MVC mà thuộc hướng tiếp cận DDD (Domain-Driven Design).

Sơ đồ lớp

Qua các bài học từ đầu đến giờ chúng ta đã xây dựng nhiều class và chia vào các nhóm theo các không gian tên, bao gồm:

- BookMan.ConsoleApp,
- BookMan.ConsoleApp.Controllers,
- BookMan.ConsoleApp.Views,
- BookMan.ConsoleApp.DataServices,
- BookMan.ConsoleApp.Models,
- Framework.

Để thấy sự phụ thuộc giữa các nhóm có thể xem sơ đồ code dưới đây:



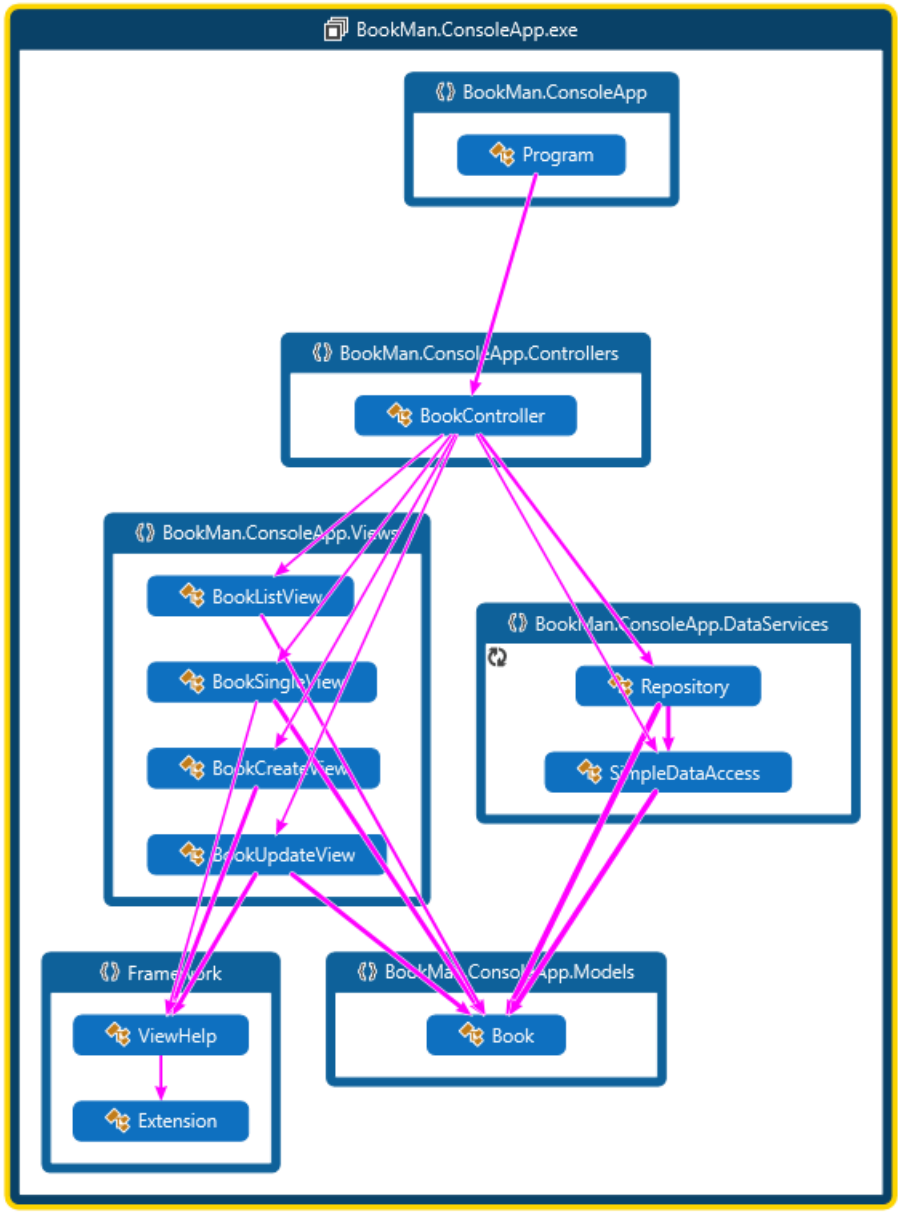
Sơ đồ lớp tổng quan

Mũi tên biểu diễn quan hệ phụ thuộc giữa các class ("lời gọi phương thức", "khởi tạo object", "tham chiếu tới").

Chúng ta có thể thấy, khối DataServices bây giờ nằm ở vị trí trung gian giữa Controllers và Models. Điều này có nghĩa là Controller không trực tiếp làm các công việc xử lý dữ liệu (như khởi tạo, thêm, sửa, xóa, lọc, nhóm, cập nhật) mà đẩy toàn bộ các công việc này cho nhóm DataServices, trong đó có Repository và DataAccess.

Sơ đồ dưới đây biểu diễn chi tiết hơn nữa các thành phần trong dự án và mối quan hệ giữa chúng.

> Các sơ đồ này được tạo ra bởi công cụ Code Map của Visual Studio (Ultimate). Các phiên bản Community và Professional không tạo ra được code map mà chỉ có thể đọc được sơ đồ này.



Sơ đồ lớp chi tiết

Thực thi mô hình repository

Khi sử dụng mô hình repository có hai cách thức thực thi phổ biến:

1. Xây dựng cho mỗi lớp thực thể một lớp repository riêng: ví dụ, với lớp Book sẽ phải xây dựng BookRepository, nếu có thêm lớp thực thể Shell (giá sách) sẽ phải xây dựng thêm ShellRepository, v.v.. Mỗi lớp repository này chỉ xây dựng những phương thức xử lý dữ liệu cần thiết để sử dụng trong controller.
2. Xây dựng một lớp repository generic: lớp này chứa đầy đủ các phương thức xử lý dữ liệu chung nhất (thêm, sửa, xóa, lọc, nhóm dữ liệu), các lớp thực thể đều sử dụng generic repository này.

Nếu số lượng lớp thực thể ít, phương án thứ nhất sẽ phù hợp hơn; nếu số lượng lớp thực thể lớn, phương án thứ hai phù hợp hơn.

Ngoài hai phương thức này còn có thể xây dựng một lớp repository chung cho cả dự án. Tất cả các phương thức xử lý dữ liệu đều đặt trong lớp này. Phương thức này chỉ áp dụng cho những bài toán nhỏ để tránh làm phức tạp code. Chúng ta đang vận dụng phương án này trong dự án.

Ngoài ra, do repository phải đọc và ghi dữ liệu với nhiều loại nguồn dữ liệu khác nhau trong các giai đoạn phát triển dự án, việc tương tác với các nguồn dữ liệu khác nhau (file, dịch vụ, cơ sở dữ liệu) để đọc và ghi dữ liệu được chuyển sang một lớp trung gian (tạm gọi là lớp data access).

Ví dụ, ở giai đoạn test và thử nghiệm trong lúc code cần nguồn dữ liệu là danh sách các object trong bộ nhớ; ở giai đoạn chạy thử nghiệm có thể sử dụng đến nguồn dữ liệu từ dịch vụ mạng hoặc dữ liệu từ cơ sở dữ liệu quan hệ.

Ở giai đoạn này chúng ta chỉ xây dựng một lớp truy xuất dữ liệu đơn giản (SimpleDataAccess) chứa dữ liệu là một danh sách các object. Đến phần sau khi học cách làm việc với file và cơ sở dữ liệu chúng ta sẽ xây dựng thêm các lớp data access khác.

Với cách tổ chức class như trên chúng ta có thể dễ dàng thay đổi thành phần data access để làm việc với các nguồn dữ liệu khác nhau. Controller có thể quyết định sử dụng data access nào.

Kết luận

Trong bài học này chúng ta đã sử dụng lớp generic List<T> để xây dựng class theo mô hình Repository để quản lý dữ liệu. Chúng ta cũng đã xem xét mô hình repository và xây dựng lớp Repository làm nơi tập trung các thao tác xử lý dữ liệu.