

# Lưu trữ dữ liệu (1): serialization, Binary, Xml, Json

Hướng dẫn tự học lập trình C# toàn tập > Lưu trữ dữ liệu (1): serialization, Binary, Xml, Json

Trong bài học này chúng ta sẽ xem xét vấn đề chuyển đổi dữ liệu (serialization) về các dạng binary, xml và json. Chúng ta sẽ vận dụng các kỹ thuật này để lưu trữ dữ liệu vào file theo các định dạng tương ứng sử dụng FileStream.

Để thực hiện bài thực hành này, bạn cần biết: [Kiến trúc stream](#) và cách sử dụng [FileStream](#); Khái niệm và cách thức thực hiện [serialization](#).

## NỘI DUNG CỦA BÀI [ Ấn ]

1. Thực hành 1: Sử dụng binary serialization để lưu trữ dữ liệu trong file nhị phân
  - 1.1. Bước 1. Xây dựng class BinaryDataAccess
  - 1.2. Bước 2. Điều chỉnh lớp Book
  - 1.3. Bước 3. Điều chỉnh lớp Repository
  - 1.4. Bước 4. Điều chỉnh lớp BookController và ShellController
  - 1.5. Bước 5. Điều chỉnh phương thức ConfigRouter
  - 1.6. Bước 6. Dịch và chạy thử chương trình
2. Thực hành 2: sử dụng xml serialization để lưu trữ dữ liệu trong file xml
3. Thực hành 3: sử dụng json serialization để lưu trữ dữ liệu trong file json
4. Kết luận

## Thực hành 1: Sử dụng binary serialization để lưu trữ dữ liệu trong file nhị phân

### Bước 1. Xây dựng class BinaryDataAccess

Trong thư mục DataServices tạo file mã nguồn BinaryDataAccess.cs cho lớp BinaryDataAccess và viết code cho lớp BinaryDataAccess như sau:

BinaryDataAccess.cs

```
1. using System.Collections.Generic;
2. using System.IO;
3. using System.Runtime.Serialization.Formatters.Binary;
4.
5. namespace BookMan.ConsoleApp.DataServices
6. {
7.     using Models;
8.
9.     public class BinaryDataAccess
10.    {
11.        public List<Book> Books { get; set; } = new List<Book>();
12.        private readonly string _file = "data.dat";
13.
14.        public void Load()
15.        {
16.            if (!File.Exists(_file))
17.            {
18.                SaveChanges();
19.                return;
20.            }
21.
22.            using (FileStream stream = File.OpenRead(_file))
```

```

23.         {
24.             BinaryFormatter formatter = new BinaryFormatter();
25.             Books = formatter.Deserialize(stream) as List<Book>;
26.         }
27.     }
28.
29.     public void SaveChanges()
30.     {
31.         using (FileStream stream = File.OpenWrite(_file))
32.         {
33.             BinaryFormatter formatter = new BinaryFormatter();
34.             formatter.Serialize(stream, Books);
35.         }
36.     }
37. }
38.

```

Ở bước này chúng ta sử dụng BinaryFormatter để thực hiện serialization thẳng vào một FileStream, cũng như deserialization từ FileStream, thay cho biến đổi về mảng byte như trong ví dụ ở phần trước.

## Bước 2. Điều chỉnh lớp Book

```

1. using System;
2.
3. namespace BookMan.ConsoleApp.Models
4. {
5.     /// <summary>
6.     /// Lớp mô tả sách điện tử
7.     /// </summary>
8.     [Serializable]
9.     public class Book
10.    {
11.        private int _id = 1;
12.        /// <summary>
13.        /// số định danh duy nhất cho mỗi object
14.        /// </summary>
15.        public int Id
16.        {
17.            get { return _id; }
18.            set { if (value >= 1) _id = value; } // id chỉ nhận giá trị >= 1
19.        }
20.        ...

```

Ở bước này chúng ta thêm attribute [Serializable] cho lớp Book.

Attribute này là bắt buộc để BinaryFormatter có thể hoạt động. Attribute này cho phép BinaryFormatter truy xuất các thành viên của lớp Book để sử dụng trong quá trình serialization. Nếu thiếu attribute này, ở giai đoạn runtime sẽ báo lỗi (nhưng ở giai đoạn compile time sẽ không báo lỗi gì).

## Bước 3. Điều chỉnh lớp Repository

```

1. public class Repository
2. {
3.     protected readonly BinaryDataAccess _context;
4.
5.     public Repository(BinaryDataAccess context)
6.     {
7.         _context = context;
8.         _context.Load();
9.     }
10.    ...

```

Ở bước này chúng ta thay SimpleDataAccess bằng BinaryDataAccess vừa tạo.

## Bước 4. Điều chỉnh lớp BookController và ShellController

```
BookController.cs ShellController.cs
1. internal class BookController : ControllerBase
2. {
3.     protected Repository Repository;
4.     public BookController(BinaryDataAccess context)
5.     {
6.         Repository = new Repository(context);
7.     }
```

Ở bước này chúng ta cũng thay SimpleDataAccess bằng BinaryDataAccess.

**B**ổ sung phương thức sau vào ShellController

```
1. public void Save()
2. {
3.     Repository.SaveChanges();
4.     Success("Data save!");
5. }
```

## Bước 5. Điều chỉnh phương thức ConfigRouter

```
1. private static void ConfigRouter()
2. {
3.     BinaryDataAccess context = new BinaryDataAccess();
4.
5.     BookController controller = new BookController(context);
6.     ShellController shell = new ShellController(context);
7.
8.     Router r = Router.Instance;
```

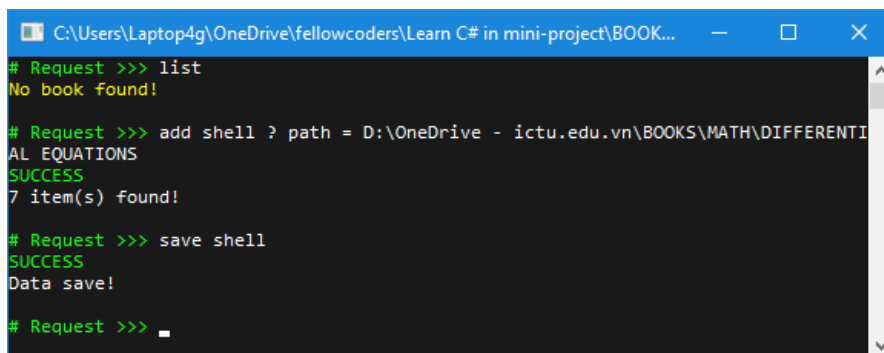
Tương tự hai bước trên, ở đây chúng ta cũng thay SimpleDataAccess bằng BinaryDataAccess.

Bổ sung route sau vào ConfigRouter

```
1. r.Register(route: "save shell",
2.     action: p => shell.Save(),
3.     help: "[save shell]");
```

## Bước 6. Dịch và chạy thử chương trình

Dịch và chạy thử với các lệnh: Add shell ? path = <value>, Save shell



```
C:\Users\Laptop4g\OneDrive\fellows\Learn C# in mini-project\BOOK...
# Request >>> list
No book found!

# Request >>> add shell ? path = D:\OneDrive - ictu.edu.vn\BOOKS\MATH\DIFFERENTIAL EQUATIONS
SUCCESS
7 item(s) found!

# Request >>> save shell
SUCCESS
Data save!

# Request >>> _
```

Kết quả chạy lệnh add shell và save shell

## Thực hành 2: sử dụng xml serialization để lưu trữ dữ liệu trong file xml

Trong thư mục *DataServices* tạo file mã nguồn *XmlDataAccess.cs* cho lớp *XmlDataAccess*. Code cho lớp *XmlDataAccess* như sau:

```
XmlDataAccess.cs
1. using System.Collections.Generic;
2. using System.IO;
3. using System.Xml;
4. using System.Xml.Serialization;
5.
6.
7. namespace BookMan.ConsoleApp.DataServices
8. {
9.     using Models;
10.    public class XmlDataAccess
11.    {
12.        public List<Book> Books { get; set; } = new List<Book>();
13.        private readonly string _file = "data.xml";
14.
15.        public void Load()
16.        {
17.            if (!File.Exists(_file))
18.            {
19.                SaveChanges();
20.                return;
21.            }
22.            var serializer = new XmlSerializer(typeof(List<Book>));
23.            using (var reader = XmlReader.Create(_file))
24.            {
25.                Books = (List<Book>)serializer.Deserialize(reader);
26.            }
27.        }
28.
29.        public void SaveChanges()
30.        {
31.            var serializer = new XmlSerializer(typeof(List<Book>));
32.            using (var writer = XmlWriter.Create(_file))
33.            {
34.                serializer.Serialize(writer, Books);
35.            }
36.        }
37.    }
38. }
```

Để chạy thử nghiệm có thể lặp lại bước 3, 4, 5 như trong phần thực hành 1 để thay *BinaryDataAccess* bằng *XmlDataAccess*.

## Thực hành 3: sử dụng json serialization để lưu trữ dữ liệu trong file json

Trong thư mục *DataServices* tạo file mã nguồn *JsonDataAccess.cs* cho lớp *JsonDataAccess*.

Code cho lớp *JsonDataAccess* như sau:

```
1. using Newtonsoft.Json;
2. using System.Collections.Generic;
3. using System.IO;
4.
5. namespace BookMan.ConsoleApp.DataServices
6. {
7.     using Models;
8.     public class JsonDataAccess
9.     {
```

```

10. public List<Book> Books { get; set; } = new List<Book>();
11. private readonly string _file = "data.json";
12.
13. public void Load()
14. {
15.     if (!File.Exists(_file))
16.     {
17.         SaveChanges();
18.         return;
19.     }
20.
21.     JsonSerializer serializer = new JsonSerializer();
22.     using (StreamReader sReader = new StreamReader(_file))
23.     using (JsonReader jReader = new JsonTextReader(sReader))
24.     {
25.         Books = serializer.Deserialize<List<Book>>(jReader);
26.     }
27.     //var jsonString = File.ReadAllText(_file);
28.     //Books = JsonConvert.DeserializeObject<List<Book>>(jsonString);
29. }
30.
31. public void SaveChanges()
32. {
33.     JsonSerializer serializer = new JsonSerializer();
34.     using (StreamWriter sWriter = new StreamWriter(_file))
35.     using (JsonWriter jWriter = new JsonTextWriter(sWriter))
36.     {
37.         serializer.Serialize(jWriter, Books);
38.     }
39.     //var jsonString = JsonConvert.SerializeObject(Books);
40.     //File.WriteAllText(_file, jsonString);
41. }
42. }
43. }

```

Để chạy thử nghiệm có thể lặp lại bước 3-4-5 như trong phần thực hành 1 để thay BinaryDataAccess bằng JsonDataAccess.

## Kết luận

Trong bài này chúng ta tập trung chính vào phương pháp làm việc với file và cách thức biến đổi dữ liệu (serialization). Chúng ta thấy rằng cơ chế làm việc với các nguồn dữ liệu (nói chung) và file (nói riêng) trong .NET framework rất thống nhất.

.NET framework cũng hỗ trợ việc chuyển đổi dữ liệu tự động với nhiều loại định dạng khác nhau, như nhị phân, xml, json.

Thông qua phần thực hành chúng ta đã xây dựng 3 lớp khác nhau để hỗ trợ lưu trữ dữ liệu vào file nhị phân, file xml và file json.

Ở bài tiếp theo chúng ta sẽ xem xét một tính năng khác của C# và .NET framework để có thể dễ dàng chuyển đổi qua lại giữa các loại dữ liệu này.

- + 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!