Bài 12: ICOLLECTION TRONG C#

Xem bài học trên website để ủng hộ Kteam: ICollection trong C#

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Ở các bài học trước, chúng ta đã cùng nhau tìm hiểu về TUPLE TRONG C#. Hôm nay chúng ta sẽ cùng tìm hiểu về **ICollection trong C#**.

Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- BIẾN , KIỂU DỮ LIỆU, TOÁN TỬ trong C#
- CÂU ĐIỀU KIỆN trong C#
- Cấu trúc cơ bản của VÒNG LẶP, HÀM trong C#
- MẢNG trong C#
- LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG C#

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- ICollection là gì?
- Thực thi interface ICollection.

ICollection là gì?

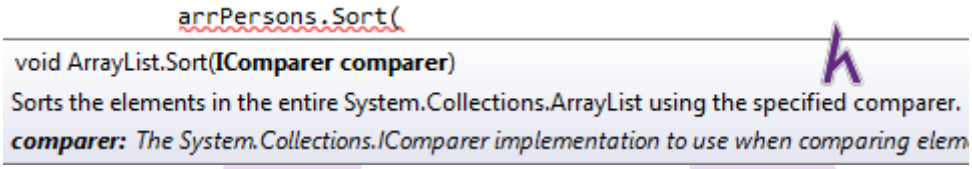
ICollection là một interface trong bộ các interface được định nghĩa sẵn của **.NET Framework**.

Về khái niệm interface cũng như cách sử dụng interface các bạn có thể xem lại bài INTERFACE TRONG C#. Còn trong bài này mình chỉ tập trung vào thực thi **ICollection**.

Trước khi vào nội dung chính chúng ta cùng tìm hiểu tại sao **.NET** lại định nghĩa sẵn nhiều interface đến như vậy.

- Đưa ra những chuẩn mực chung, mỗi interface thể hiện cho một hoặc một vài tính chất nào đó.
- Khi bạn muốn cấu trúc dữ liệu của mình có tính chất nào đó (ví dụ như bạn muốn cấu trúc dữ liệu mình là 1 dạng collection, hay cấu trúc dữ liệu của mình có khả năng duyệt bằng từ khoá foreach,...) thì bạn chỉ cần thực thi đúng interface đã được cung cấp sẵn là được.
- Ngoài ra, việc định nghĩa sẵn các interface như là một cách để lập trình viên có thể tuỳ chỉnh một số chức năng bên trong **.NET**. Nếu bạn để ý thì trong phần ví dụ của bài ARRAYLIST TRONG C# mình có thực thi 1 interface **IComparer** nhằm định nghĩa lại cách sắp xếp giữa các đối tượng.

  Phân tích 1 chút. Ban đầu hàm **Sort** trong lớp ArrayList chỉ thực hiện sắp xếp tăng dần (theo số hoặc theo bảng chữ cái) và để ý kỹ bạn sẽ thấy điều này:



  Hàm **Sort** có thể nhận vào 1 đối tượng có thực thi interface **IComparer**. Đây chính là chìa khoá để chúng ta có thể tuỳ chỉnh code.

  Để thực hiện sắp xếp cho dù là thuật toán nào thì người ta cũng cần so sánh 2 đối tượng xem đối tượng nào bé hơn thì đứng trước. Và việc so sánh này được viết bên trong 1 hàm tên là **Compare** (ý nghĩa hàm này mình đã trình bày trong bài ARRAYLIST TRONG C#).

  Trong hàm **Sort** mặc định thì hàm Compare chỉ so sánh được số hoặc chữ. Vậy nếu như ta có thể tuỳ chỉnh hàm Compare này và đưa ra cách so sánh riêng của mình thì danh sách sẽ được sắp xếp theo ý của mình. Và

cách tuỳ chỉnh hàm Compare chính là tạo một đối tượng có thực thi interface **IComaparer**.

Mình sẽ mô phỏng lại nội dung bên trong hàm **Sort** để các bạn hình dung được tại sao truyền vào 1 đối tượng thực thi interface **IComparer** thì có thể thay đổi được cách sắp xếp:

```
public void Sort(IComparer comparer)
{
/* Mình sử dụng thuật toán sắp xếp đơn giản nhất nhé.
* Còn thực sự hàm Sort của .NET sử dụng thuật toán khác nha.
*/
int count = array.Count;
for (int i = 0; i < count - 1; i++)
{
for (int j = i + 1; j < count; j++)
{
/* Nếu phần tử i bé hơn phần tử j thì thực hiện hoán đổi vị trí 2 phần tử này */
if (comparer.Compare(array[i], array[j]) > 0)
{
object tmp = array[i];
array[i] = array[j];
array[j] = tmp;
}
}
}
}
```

Bạn để ý dòng if thôi là đủ. Rõ ràng họ không quan tâm hàm Compare viết gì trong đó họ chỉ cần mình có hàm đó để họ gọi là được. Và để đảm bảo đối tượng của mình có hàm Compare thì cái này do interface **IComparer** ràng buộc.

Giờ chúng ta vào vấn đề chính. **ICollection** là 1 interface có ý nghĩa "**xác định kích thước, enumerator (bộ liệt kê) và những phương thức đồng bộ cho những tập hợp không phải kiểu generic**" hay nói ngắn gọn đây là 1 interface thể hiện tính chất của 1 collection.

Những collections mình đã tìm hiểu đều thực thi interface này:

```
public class ArrayList : IList, ICollection, IEnumerable, ICloneable
{
    public ArrayList();
    public ArrayList(ICollection c);
    public ArrayList(int capacity);
```

```
    public class Hashtable : IDictionary, ICollection,
ICloneable
    {
        public Hashtable();
        public Hashtable(IDictionary d);
```

```
public class Stack : ICollection, IEnumerable, ICloneable
{
    public Stack();
    public Stack(ICollection col);
    public Stack(int initialCapacity);
```

```
public class Queue : ICollection, IEnumerable, ICloneable
{
    public Queue();
    public Queue(ICollection col);
    public Queue(int capacity);
```

Interface **ICollection** yêu cầu chúng ta thực thi những Property, phương thức sau:

- **Count**: trả về số lượng phần tử của tập hợp.
- **IsSynchronized** và **SyncRoot**: 2 property để làm cho thao tác đa luồng với tập hợp an toàn hơn.
- **CopyTo(Array** array**, int** index**)**: phương thức thực hiện copy tập hợp ra 1 mảng, bắt đầu từ vị trí index trong tập hợp.
- **GetEnumerator()**: Trả về 1 đối tượng kiểu IEnumerator (sẽ được trình bày trong bài IENUMERABLE VÀ IENUMERATOR TRONG C#)

Thực thi interface ICollection

Chúng ta sẽ cùng tổ chức 1 **collection** đơn giản giống ArrayList bằng cách thực thi các interface cần thiết. Ví dụ này sẽ được mình thực hiện xuyên suốt cho các bài sau về interface.

Hôm nay chúng ta đã tìm hiểu về **ICollection** vậy mình sẽ bắt đầu thực thi interface này đầu tiên.

```
public class MyArrayList : ICollection
   {
      private object[] lstObj; // mảng giá trị
      private int count; // số lượng phần tử
      private const int MAXCOUNT = 100; // số lượng phần tử tối đa

      public MyArrayList()
      {
         count = -1;
         lstObj = new object[MAXCOUNT];
      }

      public MyArrayList(int count)
      {
         this.count = count;
         lstObj = new object[count];
      }

      public MyArrayList(Array array)
      {
         array.CopyTo(lstObj, 0);
         count = array.Length;
      }

      public void CopyTo(Array array, int index)
      {
         // thực hiện copy các phần tử trong lstObj từ vị trí index đến cuối sang
mảng array.
         lstObj.CopyTo(array, index);
      }

      public int Count
      {
         get { return count; }
      }

      public bool IsSynchronized
      {
         get { throw new NotImplementedException(); }
      }
```

```
        public object SyncRoot
        {
            get { throw new NotImplementedException(); }
        }

        public IEnumerator GetEnumerator()
        {
            throw new NotImplementedException();
        }
    }
```

Ở đây chúng ta sẽ tạo 1 lớp MyArrayList và thực thi interface **ICollection**. Trong lớp MyArrayList có 1 mảng các object và 1 biến lưu trữ số lượng phần tử của mảng. Sau đó thực hiện viết lại những property hoặc phương thức mà chúng ta biết bao gồm property **Count**, phương thức **CopyTo()** và một số **constructor** cơ bản.

Vì đây chưa phải là 1 collection hoàn thiện nên chúng ta không thể chạy chương trình xem thử được. Những bài tiếp theo chúng ta sẽ tiếp tục tìm hiểu về các interface có sẵn qua đó từng bước hoàn thiện MyArrayList hơn.

Kết luận

Nội dung bài này giúp các bạn nắm được:

- ICollection là gì?
- Thực thi interface ICollection.

Bài học sau chúng ta sẽ cùng tìm hiểu về ILIST TRONG C#.

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".