Bài: SortedList trong C#

Xem bài học trên website để ủng hộ Kteam: SortedList 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 HASTABLE TRONG C#. Hôm nay chúng ta sẽ cùng tìm hiểu về **SortedList 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:

- SortedList là gì?
- Một số thuộc tính và phương thức hỗ trợ sẵn trong SortedList.

SortedList là gì?

SortedList cũng là một Collections lưu trữ dữ liệu dưới dạng cặp **Key - Value**. **Key** đại diện cho 1 khoá giống như chỉ số phần tử của mảng và **Value** chính là giá trị tương ứng của khoá đó.

Đặc điểm của SortedList

- Là 1 Hashtable nhưng các giá trị được sắp xếp theo **Key**. Việc sắp xếp này được thực hiện một cách tự động mỗi khi thêm 1 phần tử mới vào SortedList.
- Có thể truy xuất đến các phần tử trong SortedListthông qua **Key**(như Hashtable) hoặc thông qua chỉ số phần tử (như ArrayList).
- SortedList chính là sự kết hợp giữa ArrayList với Hashtable .

Do SortedList cũng là 1 Collections nên để sử dụng ta cần thêm thư viện System.Collections bằng câu lệnh:

> **using System.Collections;**

Trước khi sử dụng ta cần khởi tạo vùng nhớ bằng toán tử new:

C#:

```
// khởi tạo 1 SortedList rỗng
SortedList MySL = new SortedList();
```

Bạn cũng có chỉ định sức chứa (Capacity) ngay lúc khởi tạo bằng cách thông qua **constructor** được hỗ trợ sẵn:

C#:

```
// khởi tạo 1 SortedList và chỉ định Capacity ban đầu là 5
SortedList MySL2 = new SortedList(5);
```

Bạn cũng có thể khởi tạo 1 SortedList chứa các phần tử được sao chép từ một SortedList khác:

C#:

```
/*
 * Khởi tạo 1 SortedList có kích thước bằng với MySL2.
 * Sao chép toàn độ phần tử trong MySL2 vào MySL3.
 */
SortedList MySL3 = new SortedList(MySL2);
```

Vì các phần tử của SortedList được sắp xếp tự động theo **Key** nên ta cũng có thể chỉ ra cách sắp xếp do mình tự định nghĩa (trong bài ARRAYLIST TRONG C# mình đã có trình bày về việc định nghĩa lại cách sắp xếp) thông qua **constructor** có sẵn:

C#:

```
/*
 * Mình định nghĩa 1 lớp PersonComparer có thực thi 1 interface IComparer
 * Sau đó override lại phương thức Compare.
 * Sử dụng lớp trên để truyền vào constructor của SortedList.
 */
SortedList MySL4 = new SortedList(new PersonComparer());
```

Ngoài ra bạn cũng có thể khởi tạo 1 SortedList chứa các phần tử được sao chép từ 1 SortedList khác đồng thời sắp xếp lại các phần tử theo 1 cách sắp xếp khác:

C#:

```
/*
 * Tạo 1 SortedList mới và sao chép các phần tử từ MySL3 đồng thời sắp xếp các phần tử lại
 * theo cách sắp xếp được định nghĩa trong lớp PersonComparer.
 */
SortedList MySL5 = new SortedList(MySL3, new PersonComparer());
```

Một số thuộc tính và phương thức hỗ trợ sẵn trong SortedList

Vì SortedList là sự kết hợp giữa ArrayList và Hashtable nên nó sẽ mang các thuộc tính, phương thức giống 2 **Collections** trên và một vài phương thức mới. Ở đây **Kteam** xin giới thiệu lại để những bạn nào chưa theo dõi những bài trước cũng có thể nắm.

Một số thuộc tính thông dụng trong SortedList:

TÊN THUỘC TÍNH	Ý NGHĨA
Count	Trả về 1 số nguyên là **số phần tử hiện có** trong SortedList .
Capacity	Trả về 1 số nguyên cho biết số phần tử mà SortedList **có thể chứa** (sức chứa). Nếu số phần tử được thêm vào chạm sức chứa này thì hệ thống sẽ tự động tăng lên. Ngoài ra ta có thể gán 1 sức chứa bất kỳ cho SortedList.
Keys	Trả về 1 danh sách chứa các **Key** trong SortedList.
Values	Trả về 1 danh sách chứa các **Value** trong SortedList.

Một số phương thức thông dụng trong SortedList:

TÊN PHƯƠNG THỨC	Ý NGHĨA
Add(object Key, object Value)	Thêm 1 cặp **Key - Value** vào SortedList.

Clear()	Xoá tất cả các phần tử trong SortedList.
Clone()	Tạo 1 bản sao từ SortedList hiện tại.
ContainsKey(object Key)	Kiểm tra đối tượng **Key** có tồn tại trong SortedList hay không.
ContainsValue(object Value)	Kiểm tra đối tượng **Value** có tồn tại trong SortedList hay không.
CopyTo(Array array, int Index)	Thực hiện sao chép tất cả phần tử trong SortedList sang mảng một chiều array từ vị trí **Index** của array. **Lưu ý:** array phải là mảng các object hoặc mảng các **DictionaryEntry**.
GetByIndex(int Index)	Trả về giá trị **Value** tại vị trí **Index** trong SortedList.
GetKey(int Index)	Trả về giá trị **Key** tại vị trí **Index** trong SortedList.
GetKeyList()	Trả về 1 List các **Key** trong SortedList. (xem thêm LIST TRONG C#)
GetValueList()	Trả về 1 List các **Value** trong SortedList.
IndexOfKey(object Key)	Trả về 1 số nguyên là chỉ số phần tử của 1 **Key** trong SortedList.
Remove(object Key)	Xoá đối tượng có **Key** xuất hiện đầu tiên trong SortedList.
RemoveAt(int Index)	Xoá đối tượng tại vị trí **Index** trong SortedList.
SetByIndex(int Index, object Value)	Gán giá trị **Value** mới tại vị trí **Index** trong SortedList.

Về cách sử dụng thì bạn thao tác hoàn toàn giống với Hashtable .

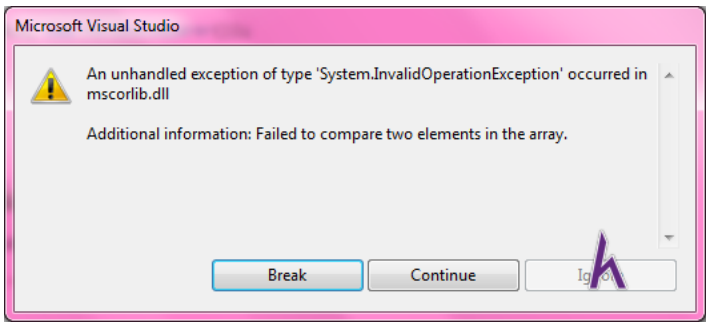
Một số lưu ý

Nếu bạn muốn các giá trị **Key** là các đối tượng thuộc 1 lớp nào đó thì bạn phải định nghĩa cách so sánh đối tượng đó. Nếu không chương trình sẽ báo lỗi vì nó không biết phải sắp xếp các **Key** này như thế nào. Ví dụ đoạn chương trình sau:

C#:

```
SortedList MySL6 = new SortedList();
MySL6.Add(new Person("HowKteam", 20), 10);
MySL6.Add(new Person("Kteam", 2), 15);
```

Khi chạy chương trình trên sẽ nhận được lỗi sau:



Để khắc phục điều này ta có thể định nghĩa 1 lớp thực thi interface IComparer và định nghĩa cách sắp xếp trong hàm Comparer:

C#:

```
/// <summary>
/// Định nghĩa 1 lớp thực thi interface IComparer.
/// override phương thức Comparer và định nghĩa cách sắp xếp trong đó.
/// Chi tiết bạn có thể xem lại bài ArrayList trong C#.
/// </summary>
class PersonComparer : IComparer
    {
        public int Compare(object x, object y)
        {
            Person a = x as Person;
            Person b = y as Person;

            if (a == null || b == null)
            {
                throw new InvalidOperationException();
            }
            else
            {
                if (a.Age > b.Age)
                {
                    return 1;
                }
                else if (a.Age == b.Age)
                {
                    return 0;
                }
                else
                {
                    return -1;
                }
            }
        }
    }
```

Sau đó sử dụng constructor của SortedList để truyền lớp này vào:

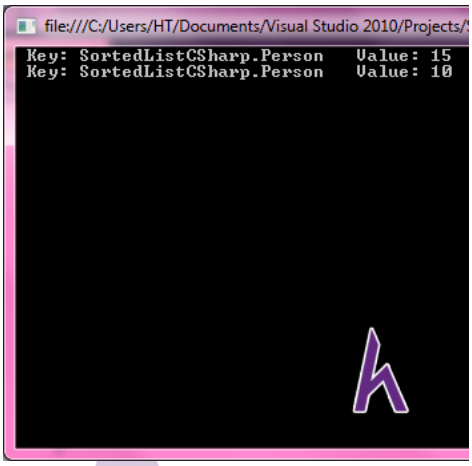
C#:

```
// tạo 1 SortedList và truyền vào cách sắp xếp các Key trong SortedList này.
            SortedList MySL6 = new SortedList(new PersonComparer());
```

Khi đó chương trình sẽ căn cứ vào hàm **Comparer** để sắp xếp các **Key**.

Khi các **Key** hoặc **Value** là các đối tượng thuộc 1 lớp nào đó thì thì ta nên override lại phương thức **ToString** để việc in ra **Key** và **Value** không bị lỗi:

Ví dụ với đoạn chương trình trên khi chưa override phương thức ToString thì kết quả hiển thị là:

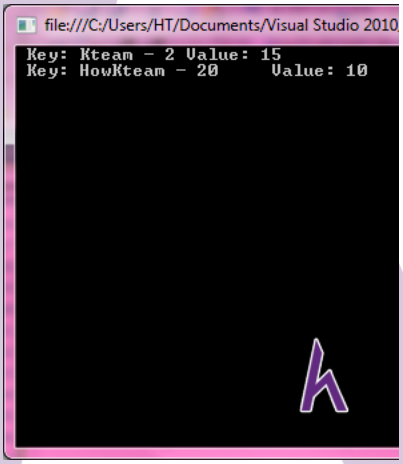


Ta thử override phương thức ToString trong lớp **Person**:

C#:

```
public override string ToString()
{
     return Name + " : " + Age;
}
```

Kết quả:



Kết luận

Nội dung bài này giúp các bạn nắm được:

- SortedList là gì?
- Một số thuộc tính và phương thức hỗ trợ sẵn trong SortedList.

Bài học sau chúng ta sẽ cùng tìm hiểu về STACK 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**".