Sorting an IList in C#

Asked 16 years, 4 months ago Modified 2 years, 2 months ago

Viewed 131k times

97

So I came across an interesting problem today. We have a WCF web service that returns an IList. Not really a big deal until I wanted to sort it.

Turns out the IList interface doesn't have a sort method built in.

I ended up using the `ArrayList.Adapter(list).Sort(new MyComparer())` method to solve the problem but it just seemed a bit "ghetto" to me.

I toyed with writing an extension method, also with inheriting from IList and implementing my own Sort() method as well as casting to a List but none of these seemed overly elegant.

So my question is, does anyone have an elegant solution to sorting an IList

c# generics sorting ilist

Share

Improve this question

Follow

asked Aug 19, 2008 at 1:23

lomaxx

116k ● 58 ● 147 ● 180

Why would you return an IList in the first place? From a WCF service? – DaeMoohn Feb 18, 2011 at 15:42

15 Answers

Sorted by: Highest score (default)

78

You can use LINQ:

```
using System.Linq;

IList<Foo> list = new List<Foo>();
IEnumerable<Foo> sortedEnum = list.OrderBy(f=>f.Bar);
IList<Foo> sortedList = sortedEnum.ToList();
```

Share Improve this answer

Follow

answered Aug 19, 2008 at 1:34

Mark Cidade

99.8k ● 33 ● 229 ● 237

67

+100

This question inspired me to write a blog post: http://blog.velir.com/index.php/2011/02/17/ilistt-sorting-a-better-way/

I think that, ideally, the .NET Framework would include a static sorting method that accepts an IList<T>, but the next best thing is to create your own extension method. It's not too hard to create a couple of methods that will allow you to sort an IList<T> as you would a List<T>. As a bonus you can overload the LINQ OrderBy extension method using the same technique, so that whether you're

using List.Sort, IList.Sort, or IEnumerable.OrderBy, you can use the exact same syntax.

```
public static class SortExtensions
{
    //  Sorts an IList<T> in place.
    public static void Sort<T>(this IList<T> list, Com
    {
        ArrayList.Adapter((IList)list).Sort(new Compar
(comparison));
    }

    // Sorts in IList<T> in place, when T is IComparab
    public static void Sort<T>(this IList<T> list) whe
    {
        Comparison<T> comparison = (l, r) => l.Compare
        Sort(list, comparison);

    }

    // Convenience method on IEnumerable<T> to allow p
    // Comparison<T> delegate to the OrderBy method.
    public static IEnumerable<T> OrderBy<T>(this IEnum
Comparison<T> comparison)
    {
        return list.OrderBy(t => t, new ComparisonComp
    }
}

// Wraps a generic Comparison<T> delegate in an ICompa
// to use a lambda expression for methods that take an
IComparer<T>
public class ComparisonComparer<T> : IComparer<T>, ICo
{
    private readonly Comparison<T> _comparison;

    public ComparisonComparer(Comparison<T> comparison
    {
        _comparison = comparison;
    }

    public int Compare(T x, T y)
```

```
    {
        return _comparison(x, y);
    }

    public int Compare(object o1, object o2)
    {
        return _comparison((T)o1, (T)o2);
    }
}
```

With these extensions, sort your IList just like you would a List:

```
IList<string> iList = new []
{
    "Carlton", "Alison", "Bob", "Eric", "David"
};

// Use the custom extensions:

// Sort in-place, by string length
iList.Sort((s1, s2) => s1.Length.CompareTo(s2.Length))

// Or use OrderBy()
IEnumerable<string> ordered = iList.OrderBy((s1, s2) =
s1.Length.CompareTo(s2.Length));
```

There's more info in the post: http://blog.velir.com/index.php/2011/02/17/ilistt-sorting-a-better-way/

Share Improve this answer

Follow

edited May 19, 2020 at 12:43

Robin Davies

7,797 ● 1 ● 39 ● 52

answered Feb 18, 2011 at 5:00

David Mills
2,415 ● 1 ● 24 ● 25

2 The right approach really would have been to offer an `ISortableList<T>` interface (with methods to sort a portion of the list using some particular comparer), have `List<T>` implement it, and have a static method which could sort any `IList<T>` by checking whether it implemented `ISortableList<T>` and, if not, copying it to an array, sorting that, clearing the `IList<T>`, and re-adding the items. – supercat Sep 28, 2012 at 23:32

5 Wonderful answer! However, a word of caution: this approach assumes that the `IList<T> list` can be cast to the non-generic `IList` interface. If you code your own class implementing the `IList<T>` interface, make sure you also implement the non-generic `IList` interface, or the code will fail with a class cast exception. – sstan Dec 15, 2015 at 14:12

1 @supercat: What could `ISortableList<T>` offer that's not already in `IList<T>`? Or, asked differently, why couldn't an `IList<T>` be sorted in-place without re-adding the items by your imagined static method? – O. R. Mapper Nov 22, 2019 at 19:19

@O.R.Mapper: If a list uses an array as a backing store (common, but not required), a sort routine that accesses array elements directly can be much faster than one which has to go through the `IList<T>` interface to access every element. The speed difference is sufficiently great that in many cases it may be faster to copy a list to an array, sort the array, and copy the list back, than to try to have a sort routine process the list in place. – supercat Nov 22, 2019 at 19:29

2 The `ComparisonComparer` class is not necessary. You can use the standard static method

`Comparer<T>.Create(comparison)` instead. – linepogl May 24, 2020 at 8:49

57

How about using LINQ To Objects to sort for you?

Say you have a `IList<Car>`, and the car had an `Engine` property, I believe you could sort as follows:

```
from c in list
orderby c.Engine
select c;
```

Edit: You do need to be quick to get answers in here. As I presented a slightly different syntax to the other answers, I will leave my answer - however, the other answers presented are equally valid.

Share Improve this answer Follow

edited Aug 19, 2008 at 1:41

answered Aug 19, 2008 at 1:34

Brad Leach

17k ● 18 ● 74 ● 88

3 It will create a new enumerable, which may not be desirable in some scenarios. You cannot sort an IList<T> in-place through the interface except by using ArrayList.Adapter method in my knowledge. – Tanveer Badar Aug 28, 2014 at 11:11

9

You're going to have to do something like that i think (convert it into a more concrete type).

Maybe take it into a List of T rather than ArrayList, so that you get type safety and more options for how you implement the comparer.

Share Improve this answer

Follow

answered Aug 19, 2008 at 1:29

Leon Bambrick

26.3k ● 9 ● 52 ● 76

6

The accepted answer by @DavidMills is quite good, but I think it can be improved upon. For one, there is no need to define the `ComparisonComparer<T>` class when the framework already includes a static method `Comparer<T>.Create(Comparison<T>)`. This method can be used to create an `IComparison` on the fly.

Also, it casts `IList<T>` to `IList` which has the potential to be dangerous. In most cases that I have seen, `List<T>` which implements `IList` is used behind the scenes to implement `IList<T>`, but this is not guaranteed and can lead to brittle code.

Lastly, the overloaded `List<T>.Sort()` method has 4 signatures and only 2 of them are implemented.

1. `List<T>.Sort()`
2. `List<T>.Sort(Comparison<T>)`
3. `List<T>.Sort(IComparer<T>)`

4. `List<T>.Sort(Int32, Int32, IComparer<T>)`

The below class implements all 4 `List<T>.Sort()` signatures for the `IList<T>` interface:

```
using System;
using System.Collections.Generic;

public static class IListExtensions
{
    public static void Sort<T>(this IList<T> list)
    {
        if (list is List<T> listImpl)
        {
            listImpl.Sort();
        }
        else
        {
            var copy = new List<T>(list);
            copy.Sort();
            Copy(copy, 0, list, 0, list.Count);
        }
    }

    public static void Sort<T>(this IList<T> list, Com
    {
        if (list is List<T> listImpl)
        {
            listImpl.Sort(comparison);
        }
        else
        {
            var copy = new List<T>(list);
            copy.Sort(comparison);
            Copy(copy, 0, list, 0, list.Count);
        }
    }

    public static void Sort<T>(this IList<T> list, ICo
    {
        if (list is List<T> listImpl)
        {
```

```
            listImpl.Sort(comparer);
        }
        else
        {
            var copy = new List<T>(list);
            copy.Sort(comparer);
            Copy(copy, 0, list, 0, list.Count);
        }
    }

    public static void Sort<T>(this IList<T> list, int
        IComparer<T> comparer)
    {
        if (list is List<T> listImpl)
        {
            listImpl.Sort(index, count, comparer);
        }
        else
        {
            var range = new List<T>(count);
            for (int i = 0; i < count; i++)
            {
                range.Add(list[index + i]);
            }
            range.Sort(comparer);
            Copy(range, 0, list, index, count);
        }
    }

    private static void Copy<T>(IList<T> sourceList, i
        IList<T> destinationList, int destinationIndex
    {
        for (int i = 0; i < count; i++)
        {
            destinationList[destinationIndex + i] = so
i];
        }
    }
}
```

Usage:

```
class Foo
{
    public int Bar;

    public Foo(int bar) { this.Bar = bar; }
}

void TestSort()
{
    IList<int> ints = new List<int>() { 1, 4, 5, 3, 2
    IList<Foo> foos = new List<Foo>()
    {
        new Foo(1),
        new Foo(4),
        new Foo(5),
        new Foo(3),
        new Foo(2),
    };

    ints.Sort();
    foos.Sort((x, y) => Comparer<int>.Default.Compare(
}
```

The idea here is to leverage the functionality of the underlying `List<T>` to handle sorting whenever possible. Again, most `IList<T>` implementations that I have seen use this. In the case when the underlying collection is a different type, fallback to creating a new instance of `List<T>` with elements from the input list, use it to do the sorting, then copy the results back to the input list. This will work even if the input list does not implement the `IList` interface.

Share Improve this answer edited Oct 22, 2022 at 16:24

Follow

answered Aug 19, 2016 at 5:08

dana

18.1k ● 7 ● 68 ● 90

2

```
try this  **USE ORDER BY** :

   public class Employee
    {
        public string Id { get; set; }
        public string Name { get; set; }
    }

 private static IList<Employee> GetItems()
        {
            List<Employee> lst = new List<Employee>();

            lst.Add(new Employee { Id = "1", Name = "E
            lst.Add(new Employee { Id = "2", Name = "E
            lst.Add(new Employee { Id = "7", Name = "E
            lst.Add(new Employee { Id = "4", Name = "E
            lst.Add(new Employee { Id = "5", Name = "E
            lst.Add(new Employee { Id = "6", Name = "E
            lst.Add(new Employee { Id = "3", Name = "E

            return lst;
        }

**var lst = GetItems().AsEnumerable();

            var orderedLst = lst.OrderBy(t => t.Id).To

            orderedLst.ForEach(emp => Console.WriteLin
emp.Id, emp.Name));**
```

Share Improve this answer

Follow

answered Aug 31, 2012 at 8:02

Dhanasekar Murugesan

3,219 ● 1 ● 20 ● 23

What if you want it to be DESC instead of ASC? – sam byte Nov 13, 2020 at 7:13

@sambyte OrderByDescending probably. I'm not sure this answer even meets the requirements though – Joe Phillips Feb 5, 2021 at 16:18

1

Found this thread while I was looking for a solution to the exact problem described in the original post. None of the answers met my situation entirely, however. Brody's answer was pretty close. Here is my situation and solution I found to it.

I have two ILists of the same type returned by NHibernate and have emerged the two IList into one, hence the need for sorting.

Like Brody said I implemented an ICompare on the object (ReportFormat) which is the type of my IList:

```
public class FormatCcdeSorter:IComparer<ReportFormat>
   {
      public int Compare(ReportFormat x, ReportFormat
       {
          return x.FormatCode.CompareTo(y.FormatCode)
       }
   }
```

I then convert the merged IList to an array of the same type:

```
ReportFormat[] myReports = new ReportFormat[reports.Co
merged IList
```

Then sort the array:

```
Array.Sort(myReports, new FormatCodeSorter());//sortin
```

Since one-dimensional array implements the interface `System.Collections.Generic.IList<T>`, the array can be used just like the original IList.

Share Improve this answer

Follow

answered Jul 13, 2010 at 22:20

John

711 ● 1 ● 9 ● 24

1

Useful for grid sorting this method sorts list based on property names. As follow the example.

```
    List<MeuTeste> temp = new List<MeuTeste>();

    temp.Add(new MeuTeste(2, "ramster", DateTime.Now))
    temp.Add(new MeuTeste(1, "ball", DateTime.Now));
    temp.Add(new MeuTeste(8, "gimm", DateTime.Now));
    temp.Add(new MeuTeste(3, "dies", DateTime.Now));
    temp.Add(new MeuTeste(9, "random", DateTime.Now));
    temp.Add(new MeuTeste(5, "call", DateTime.Now));
    temp.Add(new MeuTeste(6, "simple", DateTime.Now));
    temp.Add(new MeuTeste(7, "silver", DateTime.Now));
    temp.Add(new MeuTeste(4, "inn", DateTime.Now));

    SortList(ref temp, SortDirection.Ascending, "MyPro

    private void SortList<T>(
    ref List<T> lista
```

```
    , SortDirection sort
    , string propertyToOrder)
    {
        if (!string.IsNullOrEmpty(propertyToOrder)
        && lista != null
        && lista.Count > 0)
        {
            Type t = lista[0].GetType();

            if (sort == SortDirection.Ascending)
            {
                lista = lista.OrderBy(
                    a => t.InvokeMember(
                        propertyToOrder
                        , System.Reflection.BindingFla
                        , null
                        , a
                        , null
                    )
                ).ToList();
            }
            else
            {
                lista = lista.OrderByDescending(
                    a => t.InvokeMember(
                        propertyToOrder
                        , System.Reflection.BindingFla
                        , null
                        , a
                        , null
                    )
                ).ToList();
            }
        }
    }
```

Share Improve this answer

Follow

answered Dec 8, 2010 at 13:55

Bruno

11 ● 1

Here's an example using the stronger typing. Not sure if it's necessarily the best way though.

```
static void Main(string[] args)
{
    IList list = new List<int>() { 1, 3, 2, 5, 4, 6, 9
    List<int> stronglyTypedList = new List<int>(Cast<i
    stronglyTypedList.Sort();
}

private static IEnumerable<T> Cast<T>(IEnumerable list
{
    foreach (T item in list)
    {
        yield return item;
    }
}
```

The Cast function is just a reimplementation of the extension method that comes with 3.5 written as a normal static method. It is quite ugly and verbose unfortunately.

Share Improve this answer

Follow

answered Aug 19, 2008 at 11:38

ICR

14.2k ● 4 ● 51 ● 78

In VS2008, when I click on the service reference and select "Configure Service Reference", there is an option to choose how the client de-serializes lists returned from the service.

Notably, I can choose between System.Array, System.Collections.ArrayList and

System.Collections.Generic.List

Share Improve this answer

Follow

answered Sep 17, 2008 at 13:33

Amy B

110k ● 21 ● 139 ● 190

0

```
using System.Linq;

var yourList = SomeDAO.GetRandomThings();
yourList.ToList().Sort( (thing, randomThing) =>
thing.CompareThisProperty.CompareTo( randomThing.Compa
```

That's pretty !ghetto.

Share Improve this answer

Follow

answered Jul 6, 2009 at 13:50

vai

0

Found a good post on this and thought I'd share. [Check it out HERE]

Basically.

You can create the following class and IComparer Classes

```
public class Widget {
    public string Name = string.Empty;
    public int Size = 0;

    public Widget(string name, int size) {
    this.Name = name;
    this.Size = size;
```

```
}
}

public class WidgetNameSorter : IComparer<Widget> {
    public int Compare(Widget x, Widget y) {
        return x.Name.CompareTo(y.Name);
}
}

public class WidgetSizeSorter : IComparer<Widget> {
    public int Compare(Widget x, Widget y) {
    return x.Size.CompareTo(y.Size);
}
}
```

Then If you have an IList, you can sort it like this.

```
List<Widget> widgets = new List<Widget>();
widgets.Add(new Widget("Zeta", 6));
widgets.Add(new Widget("Beta", 3));
widgets.Add(new Widget("Alpha", 9));

widgets.Sort(new WidgetNameSorter());
widgets.Sort(new WidgetSizeSorter());
```

But Checkout this site for more information... Check it out HERE

Share Improve this answer

Follow

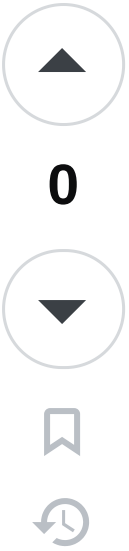
edited Aug 23, 2010 at 9:14

Mizipzor

52.2k ● 23 ● 98 ● 138

answered Feb 4, 2009 at 18:52

Brody



Is this a valid solution?

```
        IList<string> ilist = new List<string>();
        ilist.Add("B");
        ilist.Add("A");
        ilist.Add("C");

        Console.WriteLine("IList");
        foreach (string val in ilist)
            Console.WriteLine(val);
        Console.WriteLine();

        List<string> list = (List<string>)ilist;
        list.Sort();
        Console.WriteLine("List");
        foreach (string val in list)
            Console.WriteLine(val);
        Console.WriteLine();

        list = null;

        Console.WriteLine("IList again");
        foreach (string val in ilist)
            Console.WriteLine(val);
        Console.WriteLine();
```

The result was: IList B A C

List A B C

IList again A B C

Share Improve this answer

Follow

answered Sep 11, 2010 at 14:10

Yoav

2,077 ● 6 ● 27 ● 48

1 Valid if it's really a List<T>. In some cases, you have other types implementing IList<T> (for example, a plain array) where the downcast wouldn't work. Too bad that the Sort() method is not an extension method to IList<T>. – Cygon Jan 5, 2012 at 13:32

0

This looks MUCH MORE SIMPLE if you ask me. This works PERFECTLY for me.

You could use Cast() to change it to IList then use OrderBy():

```
    var ordered = theIList.Cast<T>().OrderBy(e => e);
```

WHERE T is the type eg. Model.Employee or Plugin.ContactService.Shared.Contact

Then you can use a for loop and its DONE.

```
  ObservableCollection<Plugin.ContactService.Shared.Co
ObservableCollection<Contact>();

    foreach (var item in ordered)
    {
       ContactItems.Add(item);
    }
```

Share Improve this answer

Follow

answered Nov 2, 2019 at 9:13

Momodu Deen Swarray

159 ● 1 ● 3

-1

Convert your `IList` into `List<T>` or some other generic collection and then you can easily query/sort it using `System.Linq` namespace (it will supply bunch of extension methods)

Share Improve this answer

Follow

edited Sep 20, 2015 at 12:41

H. Pauwelyn

14.3k ● 28 ● 91 ● 159

answered Aug 19, 2008 at 1:31

lubos hasko

25k ● 10 ● 57 ● 62

9 `IList<T>` implements `IEnumerable<T>` and therefore doesn't need to be converted to use Linq operations.
– Steve Guidi Jul 13, 2010 at 22:43