What is the least amount of code needed to update one list with another list?

Asked 16 years, 2 months ago Modified 16 years, 2 months ago Viewed 478 times

2

Suppose I have one list:

```
IList<int> originalList = new List<int>();
originalList.add(1);
originalList.add(5);
originalList.add(10);
```

And another list...

```
IList<int> newList = new List<int>();
newList.add(1);
newList.add(5);
newList.add(7);
newList.add(11);
```

How can I update originalList so that:

1. If the int appears in newList, keep
2. If the int does not appear in newList, remove
3. Add any ints from newList into originalList that aren't there already

Thus - making the contents of originalList:

```
{ 1, 5, 7, 11 }
```

The reason I'm asking is because I have an object with a collection of children. When the user updates this collection, instead of just deleting all children, then inserting their selections, I think it would be more efficient if I just acted on the children that were added or removed, rather than tearing down the whole collection, and inserting the newList children as if they are all new.

EDIT - Sorry - I wrote a horrible title... I should have written 'least amount of code' instead of 'efficient'. I think that threw off alot of the answers I've gotten. They are all great... thank you!

c# generics merge generic-list

Share

Improve this question

Follow

edited Sep 29, 2008 at 14:39

David Basarab

73.2k ● 43 ● 130 ● 157

asked Sep 29, 2008 at 13:35

EvilSyn

2,472 ● 7 ● 24 ● 22

How much time is your application spending copying values? – Mats Fredriksson Sep 29, 2008 at 13:52

Can there be duplicate values? e.g. { 1, 5, 5, 7, 11 } – RickL Sep 29, 2008 at 16:26

By the way, if you're using NHibernate and dealing with nonduplicate entities, you may want to be using Iesi.Collections.ISet<T>, not IList<T>. ISet<T> enforces having no duplicates. – Ryan Lundy Sep 29, 2008 at 16:54

@rick - No, I don't want duplicate values. – EvilSyn Sep 29, 2008 at 18:11

@Kyralessa - Yes, this is the precise reason I'm writing this question - it all comes back to nhibernate. I'm passing in a parent along with its child as JSON to my codebehind. Because it is serialized without a 'parent' property on the child, I need to assign it in .NET manually. :(Hence, this ? – EvilSyn Sep 29, 2008 at 18:13

10 Answers

Sorted by: Highest score (default)

5

```
originalList = newList;
```

Or if you prefer them being distinct lists:

```
originalList = new List<int>(newList);
```

But, either way does what you want. By your rules, after updating, originalList will be identical to newList.

UPDATE: I thank you all for the support of this answer, but after a closer reading of the question, I believe my other response (below) is the correct one.

Share

Improve this answer

Follow

edited Sep 29, 2008 at 14:48

answered Sep 29, 2008 at 13:41

James Curran

103k ● 37 ● 185 ● 262

I'm going to kill myself if its that simple. I will add this to my code and see! – EvilSyn Sep 29, 2008 at 13:44

It is still too early on Monday morning ;) – Vivek Sep 29, 2008 at 14:02

If you use some LINQ extension methods, you can do it in two lines:

```
originalList.RemoveAll(x => !newList.Contains(x));
originalList.AddRange(newList.Where(x => !originalList.Contains(x)));
```

This assumes (as do other people's solutions) that you've overridden Equals in your original object. But if you can't override Equals for some reason, you can create an IEqualityOperator like this:

```
class EqualThingTester : IEqualityComparer<Thing>
{
    public bool Equals(Thing x, Thing y)
    {
        return x.ParentID.Equals(y.ParentID);
    }

    public int GetHashCode(Thing obj)
    {
        return obj.ParentID.GetHashCode();
    }
}
```

Then the above lines become:

```
originalList.RemoveAll(x => !newList.Contains(x, new EqualThingTester()));
originalList.AddRange(newList.Where(x => !originalList.Contains(x, new
EqualThingTester())));
```

And if you're passing in an IEqualityOperator anyway, you can make the second line even shorter:

```
originalList.RemoveAll(x => !newList.Contains(x, new EqualThingTester()));
originalList.AddRange(newList.Except(originalList, new EqualThingTester()));
```

Share

Improve this answer

Follow

edited Sep 29, 2008 at 17:15

answered Sep 29, 2008 at 16:01

Ryan Lundy

210k ● 41 ● 184 ● 216

Incidentally, using Except makes it even shorter, but then you have to create and pass in a custom IEqualityComparer if you're using reference types. Which is a lot more code.
– Ryan Lundy Sep 29, 2008 at 16:19

Lets say I was using reference types... and instead of ints, I was using an object that had a guid ParentId, then some other fields. How would I then go about comparing on the ParentId property of each object? – EvilSyn Sep 29, 2008 at 16:48

Simplest way is to override Equals. The above lines will both use the override. (I've tested it, and that's why I didn't use Except; it doesn't use the Equals override.) If you *can't* override Equals, you can create an IEqualityComparer and pass it in as a second argument to Contains. – Ryan Lundy Sep 29, 2008 at 16:58

Thank you for the code sample I will try it out in my project tonite (because in fact I am using reference types, but didn't want to muck up my question and figured 'keep it simple'. Thanks! – EvilSyn Sep 29, 2008 at 17:06

Sorry, wrote my first response before I saw your last paragraph.

```
for(int i = originalList.length-1; i >=0; --i)
{
     if (!newList.Contains(originalList[i])
            originalList.RemoveAt(i);
}

foreach(int n in newList)
{
     if (!originaList.Contains(n))
           originalList.Add(n);
}
```

Share Improve this answer Follow

answered Sep 29, 2008 at 13:49

James Curran
103k ● 37 ● 185 ● 262

Yeppers, this is exactly what I needed.. I needed the 'unchanged' answers to stay the same. So I learned two things today, oldList = newList merges them and is super sweet.. however, this answer is exactly what I needed, and has a good amount less code than what I had for my initial solution! thanks – EvilSyn Sep 29, 2008 at 15:07

Note that this is an O(n*n) complex method (regarded the Contains call is O(n)) – xtofl Dec 30, 2008 at 12:56

Note that after merging with the algorithm above, the items order in `OriginalList` will not necessarily match the order in `NewList`. – Lightman Oct 1, 2015 at 20:53

If you are not worried about the eventual ordering, a Hashtable/HashSet will likely be the fastest.

1

Share Improve this answer Follow

answered Sep 29, 2008 at 13:50

leppie

117k • 18 • 200 • 300

LINQ solution:

1

```
originalList = new List<int>(
                      from x in newList
                      join y in originalList on x equals y into z
                      from y in z.DefaultIfEmpty()
                      select x);
```

Share Improve this answer Follow

answered Sep 29, 2008 at 13:50

Ben Hoffstein

103k • 8 • 106 • 121

My initial thought was that you could call originalList.AddRange(newList) and then remove the duplicates - but i'm not sure if that would be any more efficient than clearing the list and repopulating it.

0

Share Improve this answer Follow

answered Sep 29, 2008 at 13:44

Chris Parkinson

216 • 4 • 11

0

```
List<int> firstList = new List<int>() {1, 2, 3, 4, 5};
List<int> secondList = new List<int>() {1, 3, 5, 7, 9};

List<int> newList = new List<int>();

foreach (int i in firstList)
{
  newList.Add(i);
}

foreach (int i in secondList)
{
  if (!newList.Contains(i))
  {
    newList.Add(i);
```

```
    }
}
```

Not very clean -- but it works.

Share Improve this answer Follow

answered Sep 29, 2008 at 13:48

Ian P

13k ● 6 ● 50 ● 70

Yea this is how I am doing it now, and like you, I felt it wasn't very clean and might be done easier. – EvilSyn Sep 29, 2008 at 13:48

0

There is no built in way of doing this, the closest I can think of is the way DataTable handles new and deleted items.

What @James Curran suggests is merely replace the originalList object with the newList object. It will dump the oldList, but keep the variable (i.e. the pointer is still there).

Regardless, you should consider if optimising this is time well spent. Is the majority of the run time spent copying values from one list to the next, it might be worth it. If it's not, but rather some premature optimising you are doing, you should ignore it.

Spend time polishing the GUI or profile the application before you start optimising is my $.02.

Share

Improve this answer

Follow

edited May 23, 2017 at 11:44

Community Bot

1 ● 1

answered Sep 29, 2008 at 13:51

Mats Fredriksson

20.1k ● 6 ● 38 ● 57

Actually, I'm asking this because of persistence issues with NHibernate... Instead of ints, I have Entities representing database records. I should have reworded my title to be least amount of code instead of 'efficient' as I think that may have thrown some people off. – EvilSyn Sep 29, 2008 at 13:56

0

This is a common problem developers encounter when writing UIs to maintain many-to-many database relationships. I don't know how efficient this is, but I wrote a helper class to handle this scenario:

```
public class IEnumerableDiff<T>
{
    private delegate bool Compare(T x, T y);

    private List<T> _inXAndY;
```

```
    private List<T> _inXNotY;
    private List<T> _InYNotX;

    /// <summary>
    /// Compare two IEnumerables.
    /// </summary>
    /// <param name="x"></param>
    /// <param name="y"></param>
    /// <param name="compareKeys">True to compare objects by their keys using
Data.GetObjectKey(); false to use object.Equals comparison.</param>
    public IEnumerableDiff(IEnumerable<T> x, IEnumerable<T> y, bool
compareKeys)
    {
        _inXAndY = new List<T>();
        _inXNotY = new List<T>();
        _InYNotX = new List<T>();
        Compare comparer = null;
        bool hit = false;

        if (compareKeys)
        {
            comparer = CompareKeyEquality;
        }
        else
        {
            comparer = CompareObjectEquality;
        }


        foreach (T xItem in x)
        {
            hit = false;
            foreach (T yItem in y)
            {
                if (comparer(xItem, yItem))
                {
                    _inXAndY.Add(xItem);
                    hit = true;
                    break;
                }
            }
            if (!hit)
            {
                _inXNotY.Add(xItem);
            }
        }

        foreach (T yItem in y)
        {
            hit = false;
            foreach (T xItem in x)
            {
                if (comparer(yItem, xItem))
                {
                    hit = true;
                    break;
                }
            }
            if (!hit)
            {
                _InYNotX.Add(yItem);
            }
```

```
        }
    }

    /// <summary>
    /// Adds and removes items from the x (current) list so that the contents
match the y (new) list.
    /// </summary>
    /// <param name="x"></param>
    /// <param name="y"></param>
    /// <param name="compareKeys"></param>
    public static void SyncXList(IList<T> x, IList<T> y, bool compareKeys)
    {
        var diff = new IEnumerableDiff<T>(x, y, compareKeys);
        foreach (T item in diff.InXNotY)
        {
            x.Remove(item);
        }
        foreach (T item in diff.InYNotX)
        {
            x.Add(item);
        }
    }

    public IList<T> InXAndY
    {
        get { return _inXAndY; }
    }

    public IList<T> InXNotY
    {
        get { return _inXNotY; }
    }

    public IList<T> InYNotX
    {
        get { return _InYNotX; }
    }

    public bool ContainSameItems
    {
        get { return _inXNotY.Count == 0 && _InYNotX.Count == 0; }
    }

    private bool CompareObjectEquality(T x, T y)
    {
        return x.Equals(y);
    }

    private bool CompareKeyEquality(T x, T y)
    {
        object xKey = Data.GetObjectKey(x);
        object yKey = Data.GetObjectKey(y);
        return xKey.Equals(yKey);
    }

}
```

Share Improve this answer Follow

answered Sep 29, 2008 at 14:11

Jamie Ide

0

if your using .Net 3.5

```
var List3 = List1.Intersect(List2);
```

Creates a new list that contains the intersection of the two lists, which is what I believe you are shooting for here.

Share Improve this answer Follow

answered Sep 29, 2008 at 17:17

FlySwat

175k • 75 • 248 • 314