Difference between foreach and for loops over an IEnumerable class in C#

Asked 16 years, 3 months ago Modified 10 years, 11 months ago Viewed 22k times

13

I have been told that there is a performance difference between the following code blocks.

```
foreach (Entity e in entityList)
{
 ....
}
```

and

```
for (int i=0; i<entityList.Count; i++)
{
   Entity e = (Entity)entityList[i];
   ...
}
```

where

```
List<Entity> entityList;
```

I am no CLR expect but from what I can tell they should boil down to basically the same code. Does anybody have concrete (heck, I'd take packed dirt) evidence one way or the other?

c# performance loops

Share Improve this question Follow

asked Sep 4, 2008 at 17:20

Craig

11.9k ● 13 ● 45 ● 62

7 Answers

Sorted by: Highest score (default)

foreach creates an instance of an enumerator (returned from GetEnumerator) and that enumerator also keeps state throughout the course of the foreach loop. It then

10

repeatedly calls for the Next() object on the enumerator and runs your code for each object it returns.

They don't boil down to the same code in any way, really, which you'd see if you wrote your own enumerator.

Share Improve this answer Follow

answered Sep 4, 2008 at 17:22

Daniel Jennings

6,490 ● 3 ● 33 ● 44

9

Here is a good article that shows the IL differences between the two loops.

Foreach is technically slower however much easier to use and easier to read. Unless performance is critical I prefer the foreach loop over the for loop.

Share Improve this answer Follow

answered Sep 4, 2008 at 17:24

David Basarab

73.2k ● 43 ● 130 ● 157

> The dnspy debugger - handy for debugging .NET on remote webservers without source code - currently will not enter a foreach-loop using F11 (VS2017 debugger will tho') - and that can be a deal breaker, esp. when foreach is slower too. Also, in some languages like Javascript there are 2 variants of foreach(e.g. array & for...in) and the behaviour of the latter is not always so intuitive, which can significantly reduce any "easier to use and easier read" benefit. – Zeek2 Jul 26, 2018 at 9:36

7

The foreach sample roughly corresponds to this code:

```
using(IEnumerator<Entity> e = entityList.GetEnumerator()) {
    while(e.MoveNext()) {
        Entity entity = e.Current;
        ...
    }
}
```

There are two costs here that a regular for loop does not have to pay:

1. The cost of allocating the enumerator object by entityList.GetEnumerator().
2. The cost of two virtual methods calls (MoveNext and Current) for each element of the list.

Share Improve this answer Follow

answered Sep 4, 2008 at 17:32

Igor Ostrovsky

3

One point missed here: A List has a Count property, it internally keeps track of how many elements are in it.

An IEnumerable DOES NOT.

If you program to the interface IEnumerable and use the count extention method it will enumerate just to count the elements.

A moot point though since in the IEnumerable you cannot refer to items by index.

So if you want to lock in to Lists and Arrays you can get small performance increases.

If you want flexability use foreach and program to IEnumerable. (allowing the use of linq and/or yield return).

Share Improve this answer Follow

answered Sep 4, 2008 at 17:56

Brian Leahy

35.4k ● 12 ● 46 ● 60

1

In terms of allocations, it'd be better to look at this blogpost. It shows in exactly in what circumstances an enumerator is allocated on the heap.

Share Improve this answer Follow

answered Sep 4, 2008 at 17:36

Patrick

92.4k ● 11 ● 52 ● 61

0

I think one possible situation where you *might* get a performance gain is if the enumerable type's size and the loop condition is a constant; for example:

```
const int ArraySize = 10;
int[] values = new int[ArraySize];

//...

for (int i = 0; i
```

In this case, depending on the complexity of the loop body, the compiler might be able to replace the loop with inline calls. I have no idea if the .NET

compiler does this, and it's of limited utility if the size of the enumerable type is dynamic.

One situation where `foreach` might perform better is with data structures like a linked list where random access means traversing the list; the enumerator used by `foreach` will probably iterate one item at a time, making each access O(1) and the full loop O(n), but calling the indexer means starting at the head and finding the item at the right index; O(N) each loop for O(n^2).

> `Personally I don't usually worry about it and use foreach any time I need all items and don't care about the index of the item. If I'm not working with all of the items or I really need to know the index, I use for`. The only time I could see it being a big concern is with structures like linked lists.

Share Improve this answer Follow

answered Sep 4, 2008 at 17:31

OwenP

25.4k ● 13 ● 70 ● 105

0

```
For Loop
for loop is used to perform the opreration n times
for(int i=0;i<n;i++)
{
l=i;
}
foreach loop

int[] i={1,2,3,4,5,6}
foreach loop is used to perform each operation value/object in IEnumarable
foreach(var k in i)
{
l=k;
}
```

Share Improve this answer Follow

answered Jan 13, 2014 at 6:14

kavitha Reddy

3,333 ● 26 ● 14