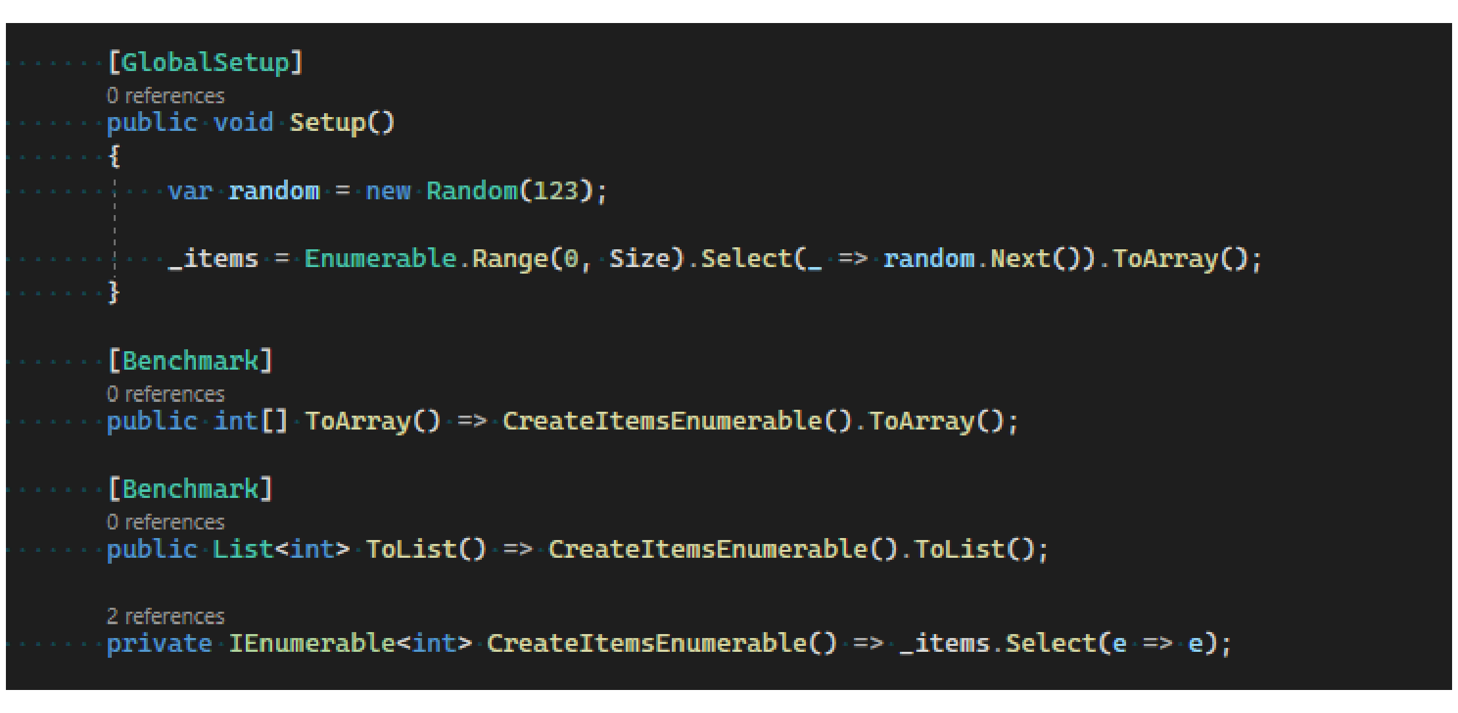
.NET —ToList 和 ToArray 的性能比较

作者：微信公众号：【架构师老卢】 2-2 10:35 7

概述：E自从 Microsoft 将语言集成查询引入 .NET 框架（也称为 LINQ）以来，开发人员一直在广泛使用它来处理集合。从简单的筛选器到聚合，再到转换，LINQ 是保持代码整洁和可读的首选技术。我们甚至有将 LINQ 指令转换为将在某个数据库中运行的 SQL 命令的提供程序。我管理的应用程序的编码准则之一规定了以下内容：在应用程序层之间传递集合时，始终使用 instead 而不是 use，并且应用于强制枚举。IReadOnlyCollectionIEnumerableToArray因为我们开发人员默认是好奇的，需要了解为什么事情是以给定的方式实现的，所以每次我们有新的团队成员时，该编码指南



```
[GlobalSetup]
0 references
public void Setup()
{
    var random = new Random(123);

    _items = Enumerable.Range(0, Size).Select(_ => random.Next()).ToArray();
}

[Benchmark]
0 references
public int[] ToArray() => CreateItemsEnumerable().ToArray();

[Benchmark]
0 references
public List<int> ToList() => CreateItemsEnumerable().ToList();

2 references
private IEnumerable<int> CreateItemsEnumerable() => _items.Select(e => e);
```

E自从 Microsoft 将语言集成查询引入 .NET 框架（也称为 LINQ）以来，开发人员一直在广泛使用它来处理集合。

从简单的筛选器到聚合，再到转换，LINQ 是保持代码整洁和可读的首选技术。我们甚至有将 LINQ 指令转换为将在某个数据库中运行的 SQL 命令的提供程序。

我管理的应用程序的编码准则之一规定了以下内容：

在应用程序层之间传递集合时，始终使用 instead 而不是 use，并且应用于强制枚举。IReadOnlyCollectionIEnumerableToArray

因为我们开发人员默认是好奇的，需要了解为什么事情是以给定的方式实现的，所以每次我们有新的团队成员时，该编码指南通常会导致以下对话：

问：为什么我们在 POCO 中使用而不是？IReadOnlyCollectionIEnumerable

答：嗯，因为我们希望合约明确说明集合在内存中，因此不会发生多次枚举，并且任何映射问题都会发生在相应的层中。

问：很公平，但为什么？该接口由数组和列表实现，我可以使用并获得相同的结果。ToArrayToList

答：结果是一样的，这是事实，但通常比 更快、更省内存，而且由于它是一个不会发生变异的短寿命集合，因此前者是首选。ToArrayToList

在本文中，我将比较创建短期集合时的性能。我还将在不同版本的框架（.NET Framework 4.8、.NET 7 和 .NET 8）中执行测试，以便我们还可以看到这些年来性能提高了多少。ToListToArray

我将使用众所周知的 C# 库来运行测试，环境如下：BenchmarkDotNet

```
BenchmarkDotNet v0.13.10, Windows 11 (10.0.22621.2428/22H2/2022Update/SunValley2)
AMD Ryzen 7 3700X, 1 CPU, 16 logical and 8 physical cores
.NET SDK 8.0.100-rc.2.23502.2
  [Host]             : .NET 8.0.0 (8.0.23.47906), X64 RyuJIT AVX2
  .NET 5.0           : .NET 5.0.17 (5.0.1722.21314), X64 RyuJIT AVX2
  .NET 7.0           : .NET 7.0.11 (7.0.1123.42427), X64 RyuJIT AVX2
  .NET 8.0           : .NET 8.0.0 (8.0.23.47906), X64 RyuJIT AVX2
  .NET Framework 4.8 : .NET Framework 4.8.1 (4.8.9181.0), X64 RyuJIT VectorSize=256
```

性能测试

该测试包括创建一个包含随机整数的集合，即参数定义的大小。为了确保随机性不会影响结果，这些值被缓存到一个数组中，在调用任何一个值之前，或者它被转换为一个新的，而不仅仅是可能导致内部优化的强制转换。ToArrayToListIEnumerable

```
[SimpleJob(RuntimeMoniker.Net48)]
[SimpleJob(RuntimeMoniker.Net70)]
[SimpleJob(RuntimeMoniker.Net80)]
[MemoryDiagnoser]
public class ToListVsToArray
{
    [Params(10, 100, 1000, 10000, 100000)]
    public int Size;

    private int[] _items;

    [GlobalSetup]
    public void Setup()
    {
        var random = new Random(123);

        _items = Enumerable.Range(0, Size).Select(_ => random.Next()).ToArray();
    }

    [Benchmark]
    public int[] ToArray() => CreateItemsEnumerable().ToArray();

    [Benchmark]
    public List<int> ToList() => CreateItemsEnumerable().ToList();

    private IEnumerable<int> CreateItemsEnumerable() => _items.Select(e => e);
}

```

性能结果

因为对于给定的应用程序，我们想要在性能之间或基于性能来决定，所以让我们首先分析每个框架版本的结果。ToArrayToList

.NET 框架 4.8

```
| Method  | Size   | Mean          | Error        | StdDev       | Gen0     | Gen1     | Gen2     | Allocated |
|-------- |------- |--------------:|-------------:|-------------:|---------:|---------:|---------:|----------:|
| ToArray | 10     |     166.31 ns |     1.013 ns |     0.947 ns |   0.0484 |        - |        - |     305 B |
| ToList  | 10     |     193.17 ns |     1.248 ns |     1.168 ns |   0.0446 |        - |        - |     281 B |
| ToArray | 100    |     965.26 ns |     8.436 ns |     7.479 ns |   0.2594 |        - |        - |    1637 B |
| ToList  | 100    |   1,032.99 ns |     4.144 ns |     3.876 ns |   0.1984 |        - |        - |    1252 B |
| ToArray | 1000   |   8,377.61 ns |    39.382 ns |    36.838 ns |   1.9836 |        - |        - |   12509 B |
| ToList  | 1000   |   8,665.62 ns |    57.777 ns |    54.045 ns |   1.3428 |   0.0153 |        - |    8514 B |
| ToArray | 10000  |  81,576.33 ns |   923.478 ns |   863.822 ns |  26.9775 |   5.3711 |        - |  171755 B |
| ToList  | 10000  |  83,476.64 ns |   410.694 ns |   342.948 ns |  20.7520 |   4.0283 |        - |  131606 B |
| ToArray | 100000 | 830,624.20 ns | 4,536.991 ns | 4,021.924 ns | 399.4141 | 399.4141 | 399.4141 | 1452144 B |
| ToList  | 100000 | 945,017.84 ns | 7,921.731 ns | 6,615.004 ns | 285.1563 | 285.1563 | 285.1563 | 1051184 B |
```

平均而言，该方法比 .NET Framework 4.8 快 10%。ToArrayToList

.NET 7.NET 7

```
| Method  | Size   | Mean          | Error        | StdDev       | Gen0     | Gen1     | Gen2     | Allocated |
|-------- |------- |--------------:|-------------:|-------------:|---------:|---------:|---------:|----------:|
| ToArray | 10     |      50.06 ns |     0.916 ns |     1.254 ns |   0.0134 |        - |        - |     112 B |
| ToList  | 10     |      56.20 ns |     1.022 ns |     0.906 ns |   0.0172 |        - |        - |     144 B |
| ToArray | 100    |     231.18 ns |     1.597 ns |     1.494 ns |   0.0563 |        - |        - |     472 B |
| ToList  | 100    |     261.38 ns |     2.523 ns |     2.236 ns |   0.0601 |        - |        - |     504 B |
| ToArray | 1000   |   2,029.47 ns |    28.534 ns |    25.295 ns |   0.4845 |        - |        - |    4072 B |
| ToList  | 1000   |   2,291.63 ns |    13.328 ns |    11.815 ns |   0.4883 |        - |        - |    4104 B |
| ToArray | 10000  |  17,322.99 ns |   176.548 ns |   165.143 ns |   4.7607 |        - |        - |   40072 B |
| ToList  | 10000  |  22,781.69 ns |   200.720 ns |   177.933 ns |   4.7607 |        - |        - |   40104 B |
| ToArray | 100000 | 306,976.29 ns | 2,525.016 ns | 2,361.901 ns | 124.5117 | 124.5117 | 124.5117 |  400114 B |
| ToList  | 100000 | 337,437.79 ns | 2,441.397 ns | 2,283.684 ns | 124.5117 | 124.5117 | 124.5117 |  400146 B |
```

平均而言，该方法比 .NET 7 快 13%。ToArrayToList

.NET 8.NET 8

```
| Method  | Size   | Mean          | Error        | StdDev       | Gen0     | Gen1     | Gen2     | Allocated |
|-------- |------- |--------------:|-------------:|-------------:|---------:|---------:|---------:|----------:|
| ToArray | 10     |      33.89 ns |     0.727 ns |     0.778 ns |   0.0134 |        - |        - |     112 B |
| ToList  | 10     |      40.17 ns |     0.668 ns |     0.625 ns |   0.0172 |        - |        - |     144 B |
| ToArray | 100    |      90.30 ns |     1.104 ns |     0.922 ns |   0.0564 |        - |        - |     472 B |
| ToList  | 100    |      97.87 ns |     1.257 ns |     1.176 ns |   0.0602 |        - |        - |     504 B |
| ToArray | 1000   |     615.97 ns |     6.819 ns |     5.695 ns |   0.4864 |        - |        - |    4072 B |
| ToList  | 1000   |     615.44 ns |     5.195 ns |     4.056 ns |   0.4902 |        - |        - |    4104 B |
| ToArray | 10000  |   5,335.01 ns |    86.179 ns |    76.395 ns |   4.7607 |        - |        - |   40072 B |
| ToList  | 10000  |   5,427.51 ns |    86.063 ns |    80.503 ns |   4.7836 |        - |        - |   40104 B |
| ToArray | 100000 | 169,711.96 ns | 2,405.080 ns | 2,249.713 ns | 124.7559 | 124.7559 | 124.7559 |  400114 B |
| ToList  | 100000 | 162,577.72 ns | 1,634.437 ns | 1,364.829 ns | 124.7559 | 124.7559 | 124.7559 |  400146 B |
```

该方法比 .NET 8 中的速度快一点，但在较大的集合上慢了 4%。ToArrayToList

.NET 性能演变

由于我们获得了每个框架版本的结果，因此让我们比较一下，看看 .NET 性能多年来是如何演变的。

ToArray（托阵列）

```
| Runtime            | Size   | Mean          | Error        | StdDev       | Gen0     | Gen1     | Gen2     | Allocated |
|------------------- |------- |--------------:|-------------:|-------------:|---------:|---------:|---------:|----------:|
| .NET Framework 4.8 | 10     |     166.31 ns |     1.013 ns |     0.947 ns |   0.0484 |        - |        - |     305 B |
| .NET 7.0           | 10     |      50.06 ns |     0.916 ns |     1.254 ns |   0.0134 |        - |        - |     112 B |
| .NET 8.0           | 10     |      33.89 ns |     0.727 ns |     0.778 ns |   0.0134 |        - |        - |     112 B |
| .NET Framework 4.8 | 100    |     965.26 ns |     8.436 ns |     7.479 ns |   0.2594 |        - |        - |    1637 B |
| .NET 7.0           | 100    |     231.18 ns |     1.597 ns |     1.494 ns |   0.0563 |        - |        - |     472 B |
| .NET 8.0           | 100    |      90.30 ns |     1.104 ns |     0.922 ns |   0.0564 |        - |        - |     472 B |
| .NET Framework 4.8 | 1000   |   8,377.61 ns |    39.382 ns |    36.838 ns |   1.9836 |        - |        - |   12509 B |
| .NET 7.0           | 1000   |   2,029.47 ns |    28.534 ns |    25.295 ns |   0.4845 |        - |        - |    4072 B |
| .NET 8.0           | 1000   |     615.97 ns |     6.819 ns |     5.695 ns |   0.4864 |        - |        - |    4072 B |
| .NET Framework 4.8 | 10000  |  81,576.33 ns |   923.478 ns |   863.822 ns |  26.9775 |   5.3711 |        - |  171755 B |
| .NET 7.0           | 10000  |  17,322.99 ns |   176.548 ns |   165.143 ns |   4.7607 |        - |        - |   40072 B |
| .NET 8.0           | 10000  |   5,335.01 ns |    86.179 ns |    76.395 ns |   4.7607 |        - |        - |   40072 B |
| .NET Framework 4.8 | 100000 | 830,624.20 ns | 4,536.991 ns | 4,021.924 ns | 399.4141 | 399.4141 | 399.4141 | 1452144 B |
| .NET 7.0           | 100000 | 306,976.29 ns | 2,525.016 ns | 2,361.901 ns | 124.5117 | 124.5117 | 124.5117 |  400114 B |
| .NET 8.0           | 100000 | 169,711.96 ns | 2,405.080 ns | 2,249.713 ns | 124.7559 | 124.7559 | 124.7559 |  400114 B |
```

在某些情况下，.NET 8 比 .NET Framework 4.8 快 90%，比 .NET 7 快 50% 以上，同时分配的内存更少，因此 .NET 8 显然是赢家。

待办事项清单

```
| Runtime            | Size   | Mean          | Error        | StdDev       | Gen0     | Gen1     | Gen2     | Allocated |
|------------------- |------- |--------------:|-------------:|-------------:|---------:|---------:|---------:|----------:|
| .NET Framework 4.8 | 10     |     193.17 ns |     1.248 ns |     1.168 ns |   0.0446 |        - |        - |     281 B |
| .NET 7.0           | 10     |      56.20 ns |     1.022 ns |     0.906 ns |   0.0172 |        - |        - |     144 B |
| .NET 8.0           | 10     |      40.17 ns |     0.668 ns |     0.625 ns |   0.0172 |        - |        - |     144 B |
| .NET Framework 4.8 | 100    |   1,032.99 ns |     4.144 ns |     3.876 ns |   0.1984 |        - |        - |    1252 B |
| .NET 7.0           | 100    |     261.38 ns |     2.523 ns |     2.236 ns |   0.0601 |        - |        - |     504 B |
| .NET 8.0           | 100    |      97.87 ns |     1.257 ns |     1.176 ns |   0.0602 |        - |        - |     504 B |
| .NET Framework 4.8 | 1000   |   8,665.62 ns |    57.777 ns |    54.045 ns |   1.3428 |   0.0153 |        - |    8514 B |
| .NET 7.0           | 1000   |   2,291.63 ns |    13.328 ns |    11.815 ns |   0.4883 |        - |        - |    4104 B |
| .NET 8.0           | 1000   |     615.44 ns |     5.195 ns |     4.056 ns |   0.4902 |        - |        - |    4104 B |
| .NET Framework 4.8 | 10000  |  83,476.64 ns |   410.694 ns |   342.948 ns |  20.7520 |   4.0283 |        - |  131606 B |
| .NET 7.0           | 10000  |  22,781.69 ns |   200.720 ns |   177.933 ns |   4.7607 |        - |        - |   40104 B |
| .NET 8.0           | 10000  |   5,427.51 ns |    86.063 ns |    80.503 ns |   4.7836 |        - |        - |   40104 B |
| .NET Framework 4.8 | 100000 | 945,017.84 ns | 7,921.731 ns | 6,615.004 ns | 285.1563 | 285.1563 | 285.1563 | 1051184 B |
| .NET 7.0           | 100000 | 337,437.79 ns | 2,441.397 ns | 2,283.684 ns | 124.5117 | 124.5117 | 124.5117 |  400146 B |
| .NET 8.0           | 100000 | 162,577.72 ns | 1,634.437 ns | 1,364.829 ns | 124.7559 | 124.7559 | 124.7559 |  400146 B |
```

.NET 8 再一次夺得金牌！

结论

在本文中，我们比较了 vs 的性能，并得出结论，前者在大多数时候表现更好，速度更快，内存效率更高，因此在创建短期集合时必须强制枚举时考虑使用它。ToArrayToList

我们还得出结论，.NET 8 版本将在这方面带来出色的性能升级。它不仅比旧版本快得多，而且非常接近，以至于应该使用哪种方法几乎无关紧要。ToArrayToList