

.NET 6: LINQ improvement

Basics and features

A little before the start

Data used in the code examples

```
public static List<Person> People => new()  
{  
    new Person {Id = 1, Name = "Gregor", Age = 33},  
    new Person {Id = 2, Name = "Roman", Age = 32},  
    new Person {Id = 3, Name = "Roma", Age = 28},  
    new Person {Id = 4, Name = "Pavel", Age = 29},  
    new Person {Id = 5, Name = "Ghost", Age = 33},  
};
```

A little before the start

Data used in the code examples

```
public static IEnumerable<Person> PeopleEnumerable
{
    get
    {
        yield return new Person {Id = 1, Name = "Gregor", Age = 33};
        yield return new Person {Id = 2, Name = "Roman", Age = 32};
        yield return new Person {Id = 3, Name = "Roma", Age = 28};
        yield return new Person {Id = 4, Name = "Pavel", Age = 29};
        yield return new Person {Id = 5, Name = "Ghost", Age = 33};
    }
}
```

New methods

MinBy()

```
public static void DemonstrateMinBy(IEnumerable<Person> source)
{
    Console.WriteLine("----MinBy----");

    //old way
    Person oldWay = source
        .OrderBy(person => person.Age)
        .First();
    //Youngest Person: Roma
}
```

New methods

MinBy()

```
public static void DemonstrateMinBy(IEnumerable<Person> source)
{
    Console.WriteLine( "----MinBy----" );

    //new way
    Person newWay = source.MinBy(person => person.Age);
    //Youngest Person: Roma
}
```

New methods

MaxBy()

```
public static void DemonstrateMaxBy(IEnumerable<Person> source)
{
    Console.WriteLine( "----MaxBy----" );

    //old way
    Person oldWay = source
        .OrderByDescending(person => person.Age)
        .First();
    //Oldest Person: Gregor
}
```

New methods

MaxBy()

```
public static void DemonstrateMaxBy(IEnumerable<Person> source)
{
    Console.WriteLine( "----MaxBy----" );

    //new way
    Person newWay = source.MaxBy(person => person.Age);
    //Oldest Person: Gregor
}
```

New methods

Chunk()

```
private static IEnumerable<IEnumerable<Person>> Chunk1(IEnumerable<Person> source, int chunkSize)
{
    IEnumerable<IEnumerable<Person>> cluster = source
        .Select((x, i) => new {Index = i, Value = x})
        .GroupBy(x => x.Index / chunkSize)
        .Select(x => x.Select(v => v.Value));
    return cluster;
}
//or
private static IEnumerable<IEnumerable<Person>> Chunk2(IEnumerable<Person> source, int chunkSize)
{
    List<List<Person>> altCluster = new List<List<Person>>();
    int count = source.Count();
    for (int i = 0; i < count; i++)
    {
        int chunkIndex = i / chunkSize;
        if (altCluster.Skip(chunkIndex).FirstOrDefault() is null)
            altCluster.Add(new List<Person>());
        altCluster[chunkIndex].Add(source.Skip(i).FirstOrDefault());
    }

    return altCluster;
}
```


New method

Chunk()

```
IEnumerable<IEnumerable<Person>> cluster = source.Chunk(chunkSize);
```

New methods

*By() methods

```
public static void Demonstrate(IEnumerable<Person> source)
{
    Console.WriteLine("----DistinctBy, UnionBy, IntersectBy, ExceptBy----");

    IEnumerable<Person> evenAgedPeople = source.Where(person => person.Age % 2 == 0);
    //Roma, Ghost
    IEnumerable<Person> personAbove30 = source.Where(person => person.Age > 30);
    //Gregor,Roman, Ghost

    UnionBy(evenAgedPeople, personAbove30);
    IntersectBy(evenAgedPeople, personAbove30);
    DistinctBy(evenAgedPeople, personAbove30);
    ExceptBy(evenAgedPeople, personAbove30);
}
```

New methods

*By() methods

```
private static void UnionBy(IEnumerable<Person> evenAgedPeople, IEnumerable<Person> personAbove30)
{
    //What we did before:
    IEnumerable<Person> union = evenAgedPeople.Union(personAbove30, new PersonByAgeComparer());
    //Roman, Roma, Gregor
}
```

New methods

*By() methods

```
private static void UnionBy(IEnumerable<Person> evenAgedPeople, IEnumerable<Person> personAbove30)
{
    //What we did before:
    IEnumerable<Person> union = evenAgedPeople.Union(personAbove30, new PersonByAgeComparer());
    //Roman, Roma, Gregor
}
```

```
class PersonByAgeComparer : IEqualityComparer<Person>
{
    public bool Equals(Person x, Person y)
    {
        if (ReferenceEquals(x, null)) return false;
        if (ReferenceEquals(y, null)) return false;
        return x.Age == y.Age;
    }

    public int GetHashCode(Person obj) => GetHashCode.Combine(obj.Age);
}
```

New methods

*By() methods

```
private static void UnionBy(IEnumerable<Person> evenAgedPeople, IEnumerable<Person> personAbove30)
{
    //And now:

    IEnumerable<Person> unionBy = evenAgedPeople.UnionBy(personAbove30, x => x.Age);
    //Roman, Roma, Gregor
}
```

New methods

*By() methods

```
private static void IntersectBy(IEnumerable<Person> evenAgedPeople, IEnumerable<Person> personAbove30)
{
    IEnumerable<Person> intersection = evenAgedPeople.Intersect(personAbove30, new PersonByAgeComparer());

    IEnumerable<Person> intersectionBy = evenAgedPeople.IntersectBy(personAbove30, x => x, new PersonByAgeComparer());
    //Roman
}
```

New methods

*By() methods

```
private static void DistinctBy(IEnumerable<Person> evenAgedPeople, IEnumerable<Person> personAbove30)
{
    IEnumerable<Person> distinct = evenAgedPeople.Distinct(new PersonByAgeComparer());

    IEnumerable<Person> distinctBy = evenAgedPeople.DistinctBy(x => x, new PersonByAgeComparer());
    //Ghost
}
```

New methods

*By() methods

```
private static void ExceptBy(IEnumerable<Person> evenAgedPeople, IEnumerable<Person> personAbove30)
{
    IEnumerable<Person> except = evenAgedPeople.Except(personAbove30, new PersonByAgeComparer());

    IEnumerable<Person> exceptBy =evenAgedPeople.ExceptBy(personAbove30, x => x, new PersonByAgeComparer());
    //Ghost
}
```


New features

Index and Ranges

```
Person secondLastPersonOld = source.TakeLast(2).FirstOrDefault();  
Person secondLastPerson = source.ElementAt(^2);  
Console.WriteLine(secondLastPerson.Name);  
//Pavel
```

New features

Index and Ranges

```
IEnumerable<Person> take3PeopleOld = source.Take(3);  
IEnumerable<Person> take3People = source.Take(..3);  
Console.WriteLine(string.Join(", ", take3People.Select(person => person.Name)));  
//Gregor, Roman, Roma
```

New features

Index and Ranges

```
IEnumerable<Person> skip1PersonOld = source.Skip(1);  
IEnumerable<Person> skip1Person = source.Take(1..);  
Console.WriteLine(string.Join(", ", skip1Person.Select(person => person.Name)));  
//Roman, Roma, Pavel, Ghost
```

New features

Index and Ranges

```
IEnumerable<Person> take3Skip1PeopleOld = source.Take(3).Skip(1);  
IEnumerable<Person> take3Skip1People = source.Take(1..3);  
Console.WriteLine(string.Join(", ", take3Skip1People.Select(person => person.Name)));  
//Roman, Roma
```

New features

Index and Ranges

```
IEnumerable<Person> takeLast2PeopleOld = source.TakeLast(2);  
IEnumerable<Person> takeLast2People = source.Take(^2..);  
Console.WriteLine(string.Join(", ", takeLast2People.Select(person => person.Name)));  
//Pavel, Ghost
```

New features

Index and Ranges

```
IEnumerable<Person> skipLast3PeopleOld = source.SkipLast(3);  
IEnumerable<Person> skipLast3People = source.Take(..^3);  
Console.WriteLine(string.Join(", ", skipLast3People.Select(person => person.Name)));  
//Gregor, Roman
```

New features

Index and Ranges

```
IEnumerable<Person> takeLast3SkipLast2Old = source.TakeLast(3).SkipLast(2);  
IEnumerable<Person> takeLast3SkipLast2 = source.Take(^3..^2);  
Console.WriteLine(string.Join(", ", takeLast3SkipLast2.Select(person =>  
person.Name)));  
//Roma
```

New methods

TryGetNotEnumeratedCount()

```
public static void Demonstrate(IEnumerable<Person> source)
{
    Console.WriteLine("----TryGetNonEnumeratedCount----");
    bool doneWithoutEnumerating = source.TryGetNonEnumeratedCount(out var sourceCount);
    var actionCount = source.Count();
    Console.WriteLine($"Was enumerated: {!doneWithoutEnumerating}, SourceCount: {sourceCount}, ActionCount: {actionCount}");
}

//Was enumerated: False, SourceCount: 5, ActionCount: 5
//Was enumerated: True, SourceCount: 0, ActionCount: 5
```


New overloads

Single/Last/FirstOrDefault(T default)

```
void OrDefaultReferenceType()
{
    VmSize[] vmSizesA =
    {
        new("Medium", 300),
        new("Small", 100)
    };

    VmSize[] vmSizesB =
    {
        new("Huge", 1500),
        new("Large", 500)
    };

    IEnumerable<VmSize> vmSizes = vmSizesA.IntersectBy(vmSizesB.Select(x => x.Name), x => x.Name);
    VmSize firstOrDefault = vmSizes.FirstOrDefault();
    VmSize overloadedFirstOrDefault = vmSizes.FirstOrDefault(VmSize.Empty);

    Console.WriteLine(
        $"Overloaded FirstOrDefault: {overloadedFirstOrDefault}; Old FirstOrDefault: {firstOrDefault}");
}

//Overloaded FirstOrDefault: Empty; Old FirstOrDefault:
```

```
private class VmSize
{
    public static readonly VmSize Empty = new VmSize("Empty", 0);

    public VmSize(string name, int cost)
    {
        Name = name;
        Cost = cost;
    }

    public string Name { get; }
    public int Cost { get; }
}
```

New overloads

Single/Last/FirstOrDefault(T default)

```
void OrDefaultValueType(IEnumerable<Person> enumerable)
{
    var firstOrDefault = enumerable.Where(x => x.Age < 10).Select(x => x.Age).FirstOrDefault();
    var overloadedFirstOrDefault = enumerable.Where(x => x.Age < 10).Select(x => x.Age).FirstOrDefault(-1);

    Console.WriteLine(
        $"Overloaded FirstOrDefault: {overloadedFirstOrDefault}; Old FirstOrDefault: {firstOrDefault}");
}

//Overloaded FirstOrDefault: -1; Old FirstOrDefault: 0
```

New overloads

Zip(...) takes 3 arguments

```
public static void Demonstrate(IEnumerable<Person> source)
{
    Console.WriteLine("----Zip with 3 parameters----");

    const string Separator = "-";
    IEnumerable<int> ids = source.Select(x=>x.Id).SkipLast(1);
    IEnumerable<string> allNames = source.Select(x=>x.Name);
    IEnumerable<int> allAges = source.Select(person => person.Age);

    //old way to zip three collections
    IEnumerable<(int Id, string name, int Age)> zippedOld = ids
        .Zip(allNames)
        .Zip(allAges)
        .Select(x=>(x.First.First, x.First.Second, x.Second));

    //new way to zip three collections
    IEnumerable<(int Id, string name, int Age)> zipped = ids.Zip(allNames, allAges);
    Console.WriteLine(string.Join(", ", zipped.Select(x => x.Id + Separator + x.Name + Separator + x.Age)));
}
```

References

Docs

- <https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6>
- <https://devblogs.microsoft.com/dotnet/announcing-net-6/>

References

Examples

- <https://github.com/MacDeath667/.NET-6-LINQ-review>

Thank you!