.NET 6: LINQ improvements

New features

Gashkov Roman

@macdeath667

A little before the start

Data used in the code examples

```
public static List<Person> Cyclists => 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> CyclistsEnumerable
{
    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};
    }
}
```

MinBy() method

```
public static void DemonstrateMinBy(IEnumerable<Person> source)
{
    Person oldWay = source.OrderBy(person => person.Age).First();

    Person newWay = source.MinBy(person => person.Age);

    //Youngest Person: Roma
}
```

MinBy() method

```
public static void DemonstrateMinBy(IEnumerable<Person> source)
{
    Person oldWay = source.OrderBy(person => person.Age).First();

    Person newWay = source.MinBy(person => person.Age);

    //Youngest Person: Roma
}
```

MinBy() method

```
public static void DemonstrateMinBy(IEnumerable<Person> source)
{
    Person oldWay = source.OrderBy(person => person.Age).First();

    Person newWay = source.MinBy(person => person.Age);

    //Youngest Person: Roma
}
```

MaxBy() method

```
public static void DemonstrateMaxBy(IEnumerable<Person> source)
{
    Person oldWay = source.OrderByDescending(person => person.Age).First();

    Person newWay = source.MaxBy(person => person.Age);

    //Oldest Person: Gregor
}
```

MaxBy() method

```
public static void DemonstrateMaxBy(IEnumerable<Person> source)
{
    Person oldWay = source.OrderByDescending(person => person.Age).First();

    Person newWay = source.MaxBy(person => person.Age);

    //Oldest Person: Gregor
}
```

MaxBy() method

```
public static void DemonstrateMaxBy(IEnumerable<Person> source)
{
    Person oldWay = source.OrderByDescending(person => person.Age).First();

    Person newWay = source.MaxBy(person => person.Age);

    //Oldest Person: Gregor
}
```

Chunk() method

```
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;
}


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;
}
```

Chunk() method

```
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;
}
```

```
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;
}
```

Chunk() method

```
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;
}
```

```
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;
}
```

Chunk() method

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

*By() methods

Operating data

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

IEnumerable<Person> personAbove30 = source.Where(person => person.Age > 30);
//Gregor,Roman, Ghost
```

*By() methods

UnionBy()

```
//What we did before:

IEnumerable<Person> union = evenAgedPeople.Union(personAbove30, new PersonByAgeComparer());
//Roman, Roma, Gregor
```

*By() methods
UnionBy()

```
//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) => HashCode.Combine(obj.Age);
}
```

*By() methods

UnionBy()

```
IEnumerable<Person> unionBy = evenAgedPeople.UnionBy(personAbove30, x => x.Age);
```

*By() methods

IntersectBy()

```
IEnumerable<Person> intersection = evenAgedPeople.Intersect(personAbove30, new PersonByAgeComparer());

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

*By() methods

ExceptBy()

```
IEnumerable<Person> except = evenAgedPeople.Except(personAbove30, new PersonByAgeComparer());

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

*By() methods
DistinctBy()

```
IEnumerable<Person> distinct = evenAgedPeople.Distinct(new PersonByAgeComparer());

IEnumerable<Person> distinctBy = evenAgedPeople.DistinctBy(x => x.Age);
//Ghost
```

Index and Ranges

```
Person secondLastPersonOld = source.TakeLast(2).FirstOrDefault();

//Pavel
```

Index and Ranges

```
Person secondLastPerson = source.ElementAt(^2);

//Pavel
```

Index and Ranges

```
IEnumerable<Person> take3PeopleOld = source.Take(3);
IEnumerable<Person> take3People = source.Take(..3);

//Gregor, Roman, Roma
```

Index and Ranges

```
IEnumerable<Person> skip1PersonOld = source.Skip(1);
IEnumerable<Person> skip1Person = source.Take(1..);

//Roman, Roma, Pavel, Ghost
```

Index and Ranges

```
IEnumerable<Person> take3Skip1PeopleOld = source.Take(3).Skip(1);
IEnumerable<Person> take3Skip1People = source.Take(1..3);

//Roman, Roma
```

Index and Ranges

```
IEnumerable<Person> takeLast2PeopleOld = source.TakeLast(2);
IEnumerable<Person> takeLast2People = source.Take(^2..);

//Pavel, Ghost
```

Index and Ranges

```
IEnumerable<Person> skipLast3PeopleOld = source.SkipLast(3);
IEnumerable<Person> skipLast3People = source.Take(..^3);

//Gregor, Roman
```

Index and Ranges

```
IEnumerable<Person> takeLast3SkipLast2Old = source.TakeLast(3).SkipLast(2);
IEnumerable<Person> takeLast3SkipLast2 = source.Take(^3..^2);

//Roma
```

Single/Last/FirstOrDefault(T default) overloads

```
private static void OrDefaultReferenceType(IEnumerable<Cyclist> enumerable)
{
    IEnumerable<Cyclist> emptyCyclists = new List<Cyclist>();

    Cyclist firstOrDefault = emptyCyclists.FirstOrDefault();
    Cyclist overloadedFirstOrDefault = emptyCyclists.FirstOrDefault(Cyclist.Empty);

}

//Overloaded FirstOrDefault: None; Old FirstOrDefault:
```

```
public class Cyclist
{
    …
    public static readonly Cyclist Empty = new Cyclist(){Id = -1, Name = "None", Age = -1};
    …
}
```

Single/Last/FirstOrDefault(T default) overloads

```
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
```

Zip(…) takes 3 arguments

```
public static void Demonstrate(IEnumerable<Person> source)
{
    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));

}
```

Zip(…) takes 3 arguments

```
public static void Demonstrate(IEnumerable<Person> source)
{
    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);

    //new way to zip three collections
    IEnumerable<(int Id, string name, int Age)> zipped = ids.Zip(allNames, allAges);
}
```

TryGetNotEnumeratedCount()

```
public static void Demonstrate(IEnumerable<Person> source)
{
    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
```

References

Docs

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

Examples

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

Thank you!