Aula 15 - LINQ Avançado + Desafio 7

Docupedia Export

Author:Balem Luis (CtP/ETS)
Date:24-May-2023 18:28

Table of Contents

- Agrupamento
- Objetos Anônimos
- Ordenamento
- LINQ como queries
- Desafio 7

1 Agrupamento

O Linq permite que nós agrupemos valores de uma coleção e tratemos cada coleção de forma diferente, Para isso usaremos o GroupBy, o exemplo a seguir mostra como essa operação funciona. Ela pode ser confusa para os iniciantes com LINQ em geral.

```
1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4
5  List<Pessoa> list = new List<Pessoa>();
6  list.Add(new Pessoa()
7  {
8      Nome = "Gilmar",
9      AnoNascimento = 1970,
10     MesNascimento = 4
11 });
12 list.Add(new Pessoa()
13 {
14     Nome = "Xispita",
15     AnoNascimento = 1999,
16     MesNascimento = 2
17 });
18 list.Add(new Pessoa()
19 {
20     Nome = "Trevisan",
21     AnoNascimento = 1999,
22     MesNascimento = 4
23 });
24 list.Add(new Pessoa()
25 {
26     Nome = "Pamella",
27     AnoNascimento = 2000,
28     MesNascimento = 6
29 });
30 list.Add(new Pessoa()
31 {
32     Nome = "Bernardo",
33     AnoNascimento = 2000,
34     MesNascimento = 1
```

```
});

// Agrupa por Ano de Nascimento
var query = list.GroupBy(p => p.AnoNascimento);

// Cada vez que o foreach roda ele pega um grupo diferente
foreach (var group in query)
{
    // g.Key pega a chave que foi usada para separar os grupos, no caso o ano de nascimento
    Console.WriteLine($"As seguintes pessoas nasceram no ano de {group.Key}:");
    foreach (var pessoa in group)
    {
        Console.WriteLine(pessoa.Nome);
    }
    Console.WriteLine();
}
// Resultado:
// As seguintes pessoas nasceram no ano de 1970:
// Gilmar
//
// As seguintes pessoas nasceram no ano de 1999:
// Xispita
// Trevisan
//
// As seguintes pessoas nasceram no ano de 2000:
// Pamella
// Bernardo
//

public class Pessoa
{
    public string Nome { get; set; }
    public int AnoNascimento { get; set; }
    public int MesNascimento { get; set; }
}
```

Note que 'group' é uma coleção de elementos que estão no mesmo grupo. Considerando o código acima, poderíamos escrever o seguinte:

```
list.GroupBy(p => p.AnoNascimento)
```

```
    .Select(g => "As seguintes pessoas nasceram no ano de {g.Key}:\n" + string.Concat(g.Select(p => p.Nome + "\n")));

foreach (var textin query)
    Console.WriteLine(text);
```

Para cada grupo que obtivemos processamos usando um Select. O Select retorna o texto 'As seguintes pessoas nasceram no ano de...' seguido de '\n' (quebra de linha) somado a um string.Concat. O string.Concat junta uma coleção de valores em uma única string. Para determinar essas strings, simplesmente usamos um Select dentro de um Select que para cada pessoa dentro do grupo seleciona seu nome seguida de uma quebra de linha. Analisando vemos que o resultado do código acima é igual ao resultado do primeiro.

2 Objetos Anônimos

Um objeto anônimo é um objeto que você pode criar sem uma classe e usá-lo por inferência dos seus campos. Basta usar a palavra reservada 'new' seguida de uma espécie de JSON que é uma estrutura nome valor. Note que tudo funciona por inferência, não é necessário dizer os tipos dos objetos.

```
1  using static System.Console;
2
3  var obj = new { nome = "Pamella", AnoNascimento = 2000 };
4
5  WriteLine(obj.nome);
6  WriteLine(obj.AnoNascimento);
```

O código acima funciona e temos um objeto de uma classe que nesse contexto não existe. Ele não pode ser exatamente retornado na maiorias das vezes pois não conseguiremos utilizar em outros contextos visto que não conhecemos o tipo do objeto contido de obj. Nem mesmo podemos mandá-los em parâmetros. Mas podemos usar isso em consultas LINQ que ficam contidas em um único escopo:

```
1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4
5  string texto = "Tles platos de tligo pala Tles Tligles Tlistes";
6
7  var query = texto
8      .GroupBy(c => c) // Agrupa as letra do texto por elas mesmas, ou seja, agrupa caracteres
9      .Where(g => g.Key != ' ') // Ignora o grupo de espaço
10     .Select(g => new { // Seleciona um objeto anônimo composto da letra maiúscula (ToUpper) com a quantidade de vezes que
   ela aparece (tamanho do grupo)
11         letra = g.Key.ToString().ToUpper(),
12         quantidade = g.Count()
13     })
14     .Select(x => $"A letra {x.letra} foi escrita {x.quantidade} vezes"); // Formata em texto para apresentação
15
16 foreach (var s in query)
17     Console.WriteLine(s);
```

3 Ordenamento

Por fim, é possível, também, ordenar:

```
using System;
using System.Linq;
using System.Collections.Generic;

string texto = "Tles platos de tligo pala Tles Tligles Tlistes";

var query = texto
    .GroupBy(c => c)
    .Where(g => g.Key != ' ')
    .Select(g => new {
        letra = g.Key.ToString().ToUpper(),
        quantidade = g.Count()
    })
    .OrderBy(x => x.letra) // Ordena pela letra, ou seja, deixa em ordem alfabética
    .Select(x => $"A letra {x.letra} foi escrita {x.quantidade} vezes");

foreach (var s in query)
    Console.WriteLine(s);
```

4 LINQ como queries

Uma coisa muito bonitas no LINQ é que podemos transformar o código LINQ em consultas SQL. Ou seja, usar consultas de banco de dados (levemente diferentes) no C# é válido e o mesmo é convertido para as funções que estamos acostumados. O código abaixo gera um código idêntico ao código acima e que tem o mesmo comportamento. Só que usando uma query alto nível de fácil compreensão para humanos:

```
1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4
5  string texto = "Tles platos de tligo pala Tles Tligles Tlistes";
6
7  var query =
8      from c in texto
9      group c by c into g
10     where g.Key != ' '
11     select new {
12         letra = g.Key.ToString().ToUpper(),
13         quantidade = g.Count()
14     } into x
15     orderby x.letra
16     select $"A letra {x.letra} foi escrita {x.quantidade} vezes";
17
18 foreach (var s in query)
19     Console.WriteLine(s);
```

Ela se traduz da seguinte forma: Para cada c na variável texto use a função GroupBy, agrupando a variável c por c (ela mesma) e criando um grupo chamado g (usando into). Depois um where que é equivalente a nossa função Where. O mesmo podemos dizer do select que cria os objetos anônimos e usa novamente a palavra reservada into para dizer que esses objetos podem ser acessados usando a letra x. Ordenamos por x.letra usando o 'orderby' e por fim mais um select em x.
É importante notar que muitas vezes esse código é mais legível mas é menos poderoso pois não conseguimos usar algumas funções como Zip por exemplo. Mas sempre podemos usar ambas as notações juntas. Veja mais alguns exemplos de conversões de notação:

```
1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4
5  int[] arr = new int[]
6  { 1, 3, 5, 7, 9 };
7
```

```
var query1a = from n in arr select n * n;

var query1b = arr.Select(n => n * n);


var query2a =
    from n in arr
    where n > 4
    select n * n into x
    where x < 50
    select x * x;

var query2b = arr
    .Where(n => n > 4)
    .Select(n => n * n)
    .Where(x => x < 50)
    .Select(x => x * x);
```

5 Desafio 7

Busque pelo SRAG 2021 (Síndrome Respiratória Aguda Grave). Esses dados trazem todos os casos de entrada no hospital com uma doença respiratória. Olhe o dicionário de dados para identificar cada coluna e poder ler os dados. Filtre os casos de Covid-19 e responda a seguinte pergunta: Onde a mortalidade é maior, nos vacinados ou nos não vacinados? Use consultas LINQ para responder essa pergunta. Dicas:

1. Não acesse os dados de um servidor remoto, caso contrário a aplicação poderá ficar muito lenta. Tenha o .csv dos dados no Desktop.
2. Use um método iterador para ler os dados linha a linha.
3. Ao terminar tudo busque pelo fenômeno de Simpson.