02

Consultas com LINQ

Transcrição

Neste vídeo, resolveremos o mesmo problema do anterior, mas com uma técnica diferente.

Iremos colocar o código do vídeo anterior em comentários, e montar uma nova consulta.

Criaremos uma nova variável `consulta`, ela será do tipo `IEnumerable`.

```
//meses.Sort();
//foreach (var mes in meses)
//{
//        if (mes.Dias == 31)
//        {
//                Console.WriteLine(mes.Nome.ToUpper());
//        }

IEnumerable<Mes> consulta = meses;

foreach (var item in consulta)
{
        Console.WriteLine(item);
```

Utilizaremos o comando "Ctrl + F5" para executar a aplicação. Feito isso, veremos que foi impressa a lista original de meses.

Agora modificaremos a consulta, para obtermos o resultado conforme os parâmetros definidos.

Para filtrarmos os meses que têm 31 dias, utilizaremos o método `Where`. Ele recebe um parâmetro, que é uma expressão lambda, e será utilizado para refinar a busca.

O parâmetro da expressão lambda será a letra "m", e a condição será `m.Dias == 31`, já que queremos somente os meses com este número de dias.

```
IEnumerable<Mes>
            consulta = meses;
                            .Where(m => m.Dias == 31);

foreach (var item in consulta)
{
        Console.WriteLine(item);
```

Executaremos o programa e veremos que, só foram impressos, os nomes dos meses que possuem 31 dias.

Olharemos o método `Where` em maior detalhe, com o comando "Alt + F12".

Percebemos que o método `Where` retorna um `IEnumerable` de mês, mantendo o mesmo tipo da nossa consulta, e recebe como parâmetro a instância da nossa listagem de meses.

Este método pertence a uma classe estática, chamada `Enumerable`, que é, na verdade, uma extensão da classe do tipo `IEnumerable`.

Ela faz parte do pacote `namespace System.Linq`, que é o conjunto de técnicas da ferramenta Linq.

Este, por sua vez, fornece uma série de instrumentos para consultar, ordenar ou agrupar informações de coleções em geral.

É uma ferramenta muito poderosa, que permite fazermos uma série de coisas que não seriam possíveis com os métodos mais básicos das coleções.

Para organizarmos a lista, incluiremos o método de ordenação `Order.By`.

```
//LINQ = CONSULTA INTEGRADA À LINGUAGEM

IEnumerable<Mes>
            consulta = meses;
                            .Where(m => m.Dias == 31);
                            .OrderBy(m => m.Nome);

foreach (var item in consulta)
{
        Console.WriteLine(item);
```

Executando o programa, vemos que os nomes estão em ordem alfabética.

Neste caso, não é exigida a implementação do método `IComparable` na classe `Mes`, para trabalharmos com o `OrderBy`.

Por isso, vamos colocar o `IComparable` em comentários:

```
class Mes //: IComparable
{
            public Mes(string nome, int dias)
            {
                    Nome = nome;
                    Dias = dias;
            }

            public string Nome { get; private set; }
            public int Dias { get; private set; }

            //public int CompareTo(object obj)
            //{
            //        Mes outro = obj as Mes;

            //        return this.Nome.CompareTo(outro.Nome);
            //}

            public override string ToString()
            {
                        return $"{Nome} - {Dias}";
                }
        }
}
```

Executaremos o programa, e vemos que foi possível ordenarmos alfabeticamente os nomes, sem utilizar o `IComparable`.

O `OrderBy` faz parte do `System.Linq`, da classe estática `Enumerable`, ou seja, é um método de extensão da interface para fornecer novas possibilidades, ferramentas.

Ele retorna uma `IOrderedEnumerable`, que também é uma implementação do `IEnumerable` que estamos utilizando na consulta.

Por fim, faremos com que os nomes fiquem todos em caixa alta.

Para transformarmos uma informação utilizando o Linq, utilizaremos um operador chamado `Select(m => m.Nme);`

Com isso, o Visual Studio identifica a presença de um erro, informando que não é possível converter um `IEnumerable<string>` para um `IEnumerable<Mes>`.

Isso porque estamos transformando uma instância da classe `Mes`, para uma string. Não é possível fazer esta conversão.

Para que isso não ocorra, podemos utilizar um `IEnumerable<string>` ou, ainda, um `var` consulta.

Faremos a primeira opção.

Executaremos a aplicação, e veremos que foram impressos somente os nomes dos meses.

O próximo passo será fazer com que eles todos sejam exibidos em caixa alta, utilizando a função `ToUpper`.

```
IEnumerable<string>
        consulta = meses
                        .Where(m => m.Dias == 31)
                        .PrderBy(m => m.Nome)
                        .Select(m => m.Npme.ToUpper());
```

Executando a aplicação, observamos que atingimos o resultado desejado.

Com a consula Linq, vimos que há uma separação entre a montagem da consulta, que são estas linhas, agrupadas:

```
IEnumerable<string>
        consulta = meses
                        .Where(m => m.Dias == 31)
                        .OrderBy(m => m.Nome)
                        .Select(m => m.Nome.ToUpper());
```

E, em seguida, temos a utilização da consulta:

```
foreach (var item in consulta)
{
        Console.WriteLine(item);
}
```

Isso faz com que fique mais fácil de identificar os limites da formação da consulta, ou seja, onde ela começa e onde termina.

Não há dificuldade em perceber que ela está filtrando a lista de meses pelo número de dias, ordenando alfabeticamente, e transformando em letras maiúsculas.

Há uma sequência lógica para modificarmos a lista original, até atingirmos o resultado desejado.

Para aqueles familiarizados com consultas SQL, é possível perceber que é bastante similar. O Linq é baseado em consultas deste tipo, para poder facilitar a leitura do desenvolvedor. Neste curso, continuaremos a trabalhar com o Linq nas aulas seguintes.

Caso queira aprofundar o conhecimento de consultas Linq, a **Alura** oferece o curso "Entity LinQ parte 1: Crie queries poderosas em C#". (https://cursos.alura.com.br/course/linq-c-sharp)