01

# Covariância de IEnumerable

### Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://github.com/marcelooliveira/csharp-collections-2/archive/2dd14403e95d9e852be727e5e43a4c50f294d217.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Neste vídeo, veremos a conversão de coleções, bem como a de tipo mais básicos.

Faremos primeiro a conversão implícita de uma `string`, para um tipo `object`.

Declararemos uma variável `string titulo = "meses"` e, em seguida, uma do tipo `object`.

```
namespace A5._1_Covariancia
{
        class Program
        {
                static void Main(string[] args)
                {
                        Console.WriteLine("string para object");
                        string titulo = "meses";
                        object obj = titulo;
                        Console.WriteLine(obj);
```

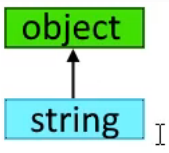
Em seguida, executaremos o programa, para confirmar que foi possível realizar a conversão implícita.

Veremos que foi impresso no console o seguinte texto:

string para object meses

Ou seja, a conversão foi realizada com sucesso.

Isto foi possível porque a classe `string`, assim como qualquer classe do .NET Framework, deriva de `object`, como vemos abaixo:




A seguir, tentaremos realizar a mesma operação só que, desta vez, por meio de listas.

Resolveremos um problema, que é a conversão de uma lista de `string` para uma lista de `object`.

Esta lista conterá os meses do ano.

```
Console.WriteLine("List<string> para List<object>");

IList<string> listaMeses = new List<string>
{
        "Janeiro", "Fevereiro", "Março",
        "Abril", "Maio", "Junho",
        "Julho", "Agosto", "Setembro",
        "Outubro", "Novembro", "Dezembro"
};
IList<object> listaObj = listaMeses;
```

Do jeito que está, não é possível ainda executar a aplicação porque o Visual Studio indica que há um erro.

Isso porque a interface genérica `IList` não permite uma conversão de `IList<string>` para `IList<object>` .

Para isso, poderíamos tentar criar uma lista de `object` , e começar a popular a partir de uma lista de meses. Entretanto, isto não seria uma conversão, e sim uma cópia de outra lista existente.

Em conclusão, uma lista de `string` não pode ser convertida em uma lista de `object` .

Por isso, comentaremos esta última linha de código, e continuaremos abaixo.

Tentaremos converter um array de `string` para um array de `object` .

Declararemos a nova coleção: `string[] arrayMeses = new string[]` .

Dentro dela, incluiremos o conteúdo, que no caso, serão os meses do ano.

Em seguida, criaremos uma lista de objetos.

Ao fazermos a conversão implícita, percebemos que o Visual Studio não faz nenhuma ressalva.

Iremos, portanto, imprimir o conteúdo do `arrayObj` .

```
//IList<object> listaObj = listaMeses;
Console.WriteLine();

Console.WriteLine("strin[] para object[]?");
string[] arrayMeses = new string[]
{
        "Janeiro", "Fevereiro", "Março",
        "Abril", "Maio", "Junho",
        "Julho", "Agosto", "Setembro",
        "Outubro", "Novembro", "Dezembro"
};
object[] arrayObj = arrayMeses;
Console.WriteLine(arrayObj);''
```

Executaremos a aplicação, com o atalho "Ctrl + F5".

Como podemos observar, foi impresso o `System.String[]` , indicando que esse é o *namespace*, e o nome da classe que está sendo armazenada, que é `arrayObj` .

Com isso, foi possível visualizar o tipo.

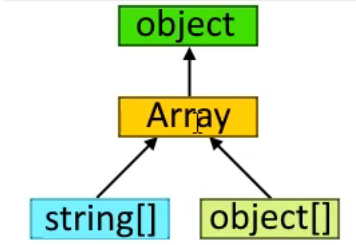
A seguir, imprimiremos o conteúdo de `arrayObj`, que foi convertido implicitamente.

Para isso, criaremos um laço `foreach`.

```
object[] arrayObj = arrayMeses;
Console.WriteLine(arrayObj);''
foreach (var item in arrayObj)
{
        Console.WriteLine(item);
}
```

Ao executarmos a aplicação, veremos que é possível visualizarmos no console, o nome de todos os meses do ano, ou seja, todos os elementos do array.

Esta conversão foi possível porque, como vemos no diagrama abaixo, há uma relação de hereditariedade entre os elementos:




Importante notarmos que a classe array de `string` não descende da de `object`. Elas estão em um mesmo nível.

A conversão implícita é possível por causa da característica chamada "covariância".

```
object[] arrayObj = arrayMeses; //COVARIÂNCIA!
Console.WriteLine(arrayObj);''
foreach (var item in arrayObj)
{
        Console.WriteLine(item);
}
```

Há um problema da covariância, em relação aos arrays. Isso porque, o array de `object` não precisa, necessariamente, armazenar somente strings.

Como teste, tentaremos trocar a instância de `arrayObj`. O primeiro elemento, índice "0", passará a ser "Primeiro Mês".

```
object[] arrayObj = arrayMeses; //COVARIÂNCIA!
Console.WriteLine(arrayObj);''
foreach (var item in arrayObj)
{
        Console.WriteLine(item);
}

arrayObj[0] = "PRIMEIRO MÊS";
```

```
Console.WriteLine(arrayObj[0]);
Console.WriteLine();
```

Executando o programa, e observando os elementos impressos, vimos que foi possível trocar, com sucesso, o conteúdo do primeiro elemento do `arrayObj`.

Em seguida, iremos trocar a string por um número inteiro.

```
arrayObj[0] = "PRIMEIRO MÊS";
Console.WriteLine(arrayObj[0]);
Console.WriteLine();

arrayObj[0] = 12345;
Console.WriteLine(arrayObj[0]);
Console.WriteLine();
```

Ao tentarmos executar a aplicação, surgirá uma caixa de diálogo, indicando que há uma "exceção sem tratamento". Uma tentativa de acesso a um elemento como um tipo incompatível com a matriz.

Isso acontece porque, apesar de nosso array ser `object`, ele está armazenando, na realidade, um array de strings.

Quando fizemos a conversão implícita, a covariância, isso resultou em um problema no qual um array de objetos não poderá mais armazenar qualquer tipo de objeto, como seria o esperado.

Este é um problema de covariância do array, que deve ser evitado, sempre. Apesar de ser possível, não é recomendável trocar uma string por outra, isto é uma funcionalidade que está quebrada no .NET Framework.

Comentaremos as linhas onde há o problema.

```
arrayObj[0] = "PRIMEIRO MÊS";
Console.WriteLine(arrayObj[0]);
Console.WriteLine();

//arrayObj[0] = 12345;
//Console.WriteLine(arrayObj[0]);
//Console.WriteLine();
```

Passaremos para a transformação de uma lista de strings em um `IEnumerable<object>`.

Declararemos uma nova variável.

```
arrayObj[0] = "PRIMEIRO MÊS";
Console.WriteLine(arrayObj[0]);
Console.WriteLine();

//arrayObj[0] = 12345;
//Console.WriteLine(arrayObj[0]);
//Console.WriteLine();

IEnumerable<object> enumObj = listaMeses; //COVARIÂNCIA
foreach (var item in enumObj)
```

```
{
        Console.WriteLine(item);
}
```

Neste caso, não houve nenhum erro indicado pelo Visual Studio.

A covariância é possível, não só no array, como no `IEnumerable`.

Executando a aplicação, vimos que foi possível listarmos todos os meses, de janeiro até dezembro.

Aqui, a covariância foi feita de forma segura, não há nenhum contra-indicação.

Clicando sobre `IEnumerable` e utilizando o atalho "Alt + F12", vemos sua interface pública, com um parâmetro que é um tipo, o `T`.

Sua definição é: o tipo do objeto para ser enumerado. Este parâmetro é covariante, ou seja, é possível utilizar tanto o tipo que está sendo especificado, que no caso é um `object` (poderia ser uma string), como qualquer tipo derivado.

O `IEnumerable<object>` pode ser utilizado para armazenar uma lista de objetos, de strings, ou de qualquer outro tipo. Isso porque ele é covariante.

Além disso, observamos que o parâmetro `T` está marcado como `out`, isso indica que ele não é modificado pela interface, ou seja, esta covariância é segura.

Se tentarmos quebrar o `enumObj` colocando, por exemplo, o número "12345", não será possível aplicar o indexador, porque nossa variável é uma que não possui indexador, e por isso mesmo, ela é segura.

Vimos neste vídeo, alguns problemas que podemos encontrar na conversão de coleções do .NET Framework.