03

(Opcional) Diferenças entre programação síncrona e assíncrona

Transcrição

Agora vamos falar de um assunto que está indiretamente ligado ao MongoDB, que é a diferença entre programação síncrona e assíncrona.

Na programação síncrona, sempre que estivermos trabalhando com um programa principal que possua uma função ou sub-rotina, quando for necessária sua utilização, elas serão executadas num mesmo processamento, ou *thread*, que o programa principal. Isso significa que ele ficará parado, esperando o término de determinada função.

Em contrapartida, na programação assíncrona o programa principal é executado num processamento independente da sub-rotina ou função, dessa forma ele pode continuar com seu processamento normal enquanto elas são executadas paralelamente.

Na prática, estes mecanismos são mais comumente utilizados quando temos funções que demandam muito tempo, como por exemplo um cálculo muito complicado, ou um acesso a um banco de dados. Assim o programa principal não depende do retorno deste processo para continuar.

Se por exemplo temos um front-end desktop, ele não fica congelado, esperando a função terminar, permitindo ao usuário que utilize outras funcionalidades enquanto o processamento é executado. Por exemplo um front-end em .ASP, ou desktop.

Isto é relevante no caso do MongoDB pois, na documentação dele, é feita uma recomendação expressa à utilização da forma assíncrona, apesar de ser possível utilizarmos o formato síncrono.

Neste próximo exemplo vamos trabalhar com as duas formas para que possamos observar em maior detalhe as diferenças entre elas.

Inicialmente iremos trabalhar com a forma síncrona.

Vamos primeiro criar uma rotina chamada `static void Main(string[] args)` e, em seguida, criar outra chamada `MainSync(args)`, onde vamos ter os mesmo argumentos recebidos na chamada principal. Notamos que ela está, inicialmente, apresentando um erro porque ainda não escrevemos esta função.

A seguir vamos incluir dois comandos `Console.WriteLine("Pressione ENTER")` e `Console.ReadLine;`. Este segundo comando fará com que a console espere até que alguma tecla no computador seja utilizada, mostrando o resultado do que foi escrito.

O próximo passo será criar a função `static void MainSync()`, que deverá receber os mesmos argumentos da primeira rotina que criamos.

Em seguida vamos criar o comando `Console.WriteLine("Esperando 10 segundos ....");`, e para que o computador espere este tempo, criaremos o comando `System.Threading.Thread.Sleep(10000);`. Uma vez que o parâmetro do comando `sleep` é em milisegundos, colocamos a numeração 10000 que equivale a 10 segundos.

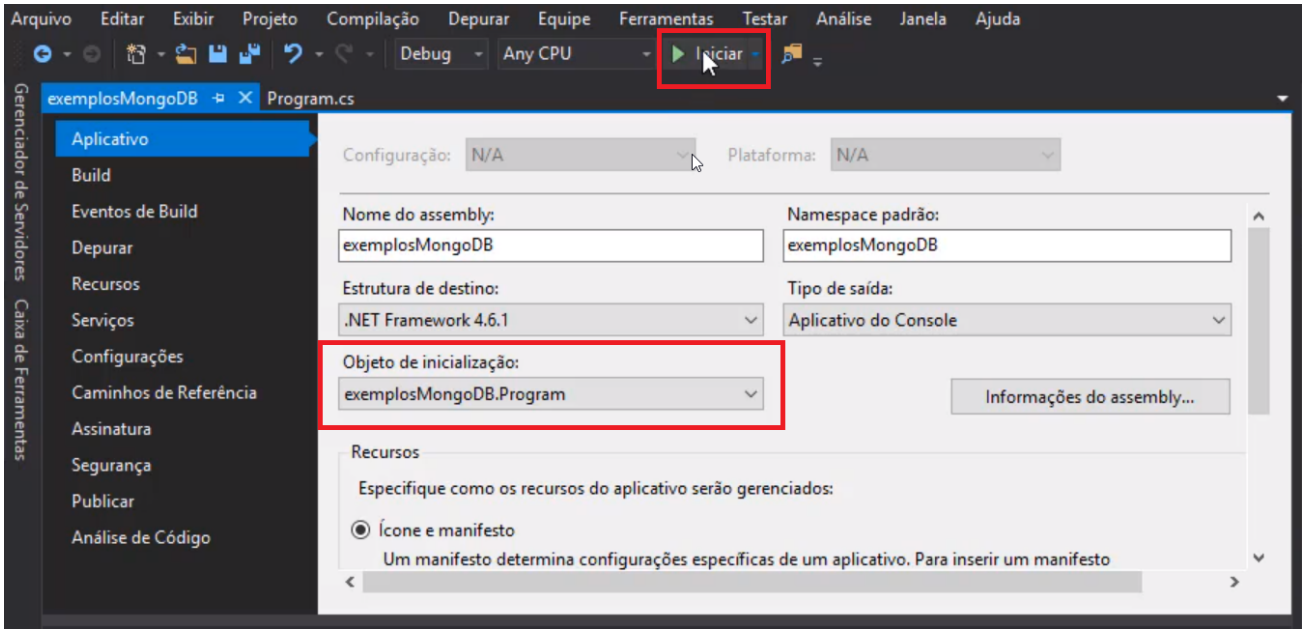
Por último vamos inserir o comando indicando o tempo esperado `Console.WriteLine("Esperei 10 segundos ....");`.

```
using System;
using System.Collections.Generic;
using System.Ling;
using System.Text;
using System.Threading.Tasks;

namespace exemploMongoDB
{
    class Program
    {
        static void Main(string[] args)
        {
            MainSync(args);
            Console.WriteLine("Pressione ENTER")
            Console.ReadLine();
        }

        static void MainSync(string[] args);
        {
            Console.WriteLine("Esperando 10 segundos ....");
            System.Threading.Thread.Sleep(10000);
            Console.WriteLine("Esperei 10 segundos ....");
        }
    }
}
```

O programa será iniciado, chamará a primeira função e ficará parado enquando a segunda é executada. Uma vez concluída, ele então executará os dois próximos comandos.

Para verificarmos o programa em execução, em "Gerenciador de Soluções" clicamos com o botão direito do mouse sobre o nome do nosso projeto e escolhemos a opção "Propriedades".

Em "Objeto de Inicialização" selecionamos o programa que foi construído e cliclamos em "Iniciar", no menu superior, para que ele seja executado.



Neste momento ele exibirá uma mensagem indicando que está aguardando por 10 segundos, período em que o processamento está suspenso, aguardando a execução da função.

Após os 10 segundos voltamos ao programa principal, a partir daí é exibida a mensagem "Pressione ENTER".

Por fim, se pressionarmos "ENTER", o programa é encerrado.

Agora, vamos repitir este programa só que de forma assíncrona.

Vamos clicar com o botão direito do mouse sobre o nome do nosso projeto e escolher "Adicionar > Novo Item". Vamos dar o nome de "programAssincrono.cs" e clicar em "Adicionar".

Vamos aproveitar o código já feito, basta copiá-lo para a nova janela do programa assíncrono.

Haverá algumas diferenças pontuais entre os dois códigos. Em primeiro lugar, a forma de declarar uma função não será a mesma, neste caso precisamos indicar explicitamente o processamento assíncrono, utilizando o comando `async`, resultando em `static async void MainSync(string[] args)`.

Além disso, não devemos tratar isso como uma rotina mas sim como função, que será uma variável do tipo *task* `static async Task MainSync(string[] args)`. Colocando o mouse sobre ela temos mais algumas informações e vemos que ela é um tipo de processamento do sistema operacional.

Para ela vamos dar a denominação de "MainASync", resultando numa função `static async Task MainASync(string[] args)`.

Assim como anteriormente, vamos aguardar 10 segundos. A diferença é que, aqui, o comando *sleep* será `await Task.Delay(10000);`.

O comando `await` só funciona se declaramos que o processamento é assíncrono, através do comando `async`. Isso informa ao processador que será executado na *thread* nova que foi criada. Entretanto, na chamada da função também deverá ser criada uma variável do tipo *task*, chamando a nova função, ou seja `Task T = MainAsync(string[] args).

Dessa forma, o programa principal chamará a função de forma assíncrona e será executada na variável "T", do tipo *thread*.

A função será executada e, concomitantemente, o programa principal continuará exibindo os comandos, ao invés de ficar suspenso até a função terminar.

```
using System;
using System.Collections.Generic;
using System.Ling;
using System.Text;
using System.Threading.Tasks;

namespace exemploMongoDB
{
    class programAssincrono
    {
        static void Main(string[] args)
        {
            Task T = MainSync(args);
            Console.WriteLine("Pressione ENTER")
            Console.ReadLine();
        }

        static async Task MainSync(string[] args);
```

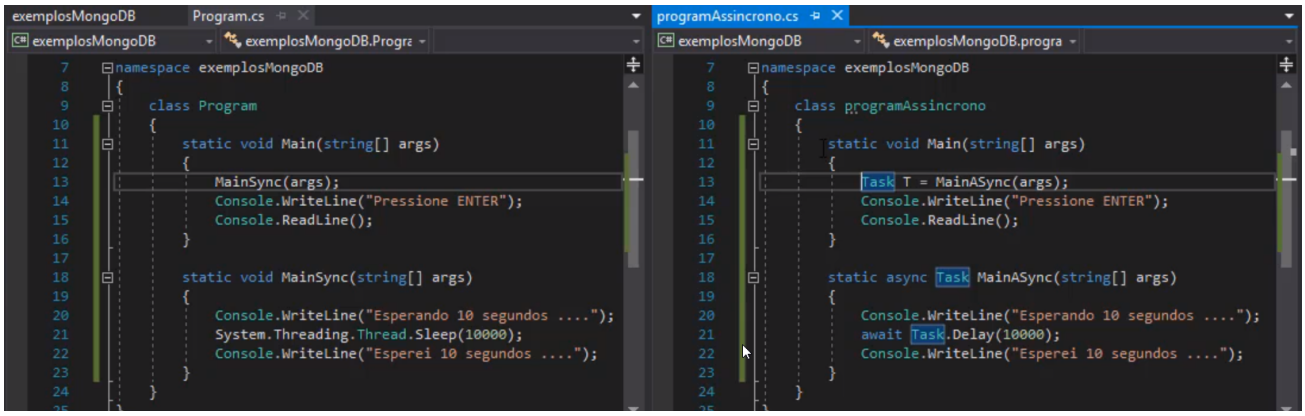
```
        {
            Console.WriteLine("Esperando 10 segundos ....");
            await Task.Delay(10000);
            Console.WriteLine("Esperei 10 segundos ....");
        }
    }
}
```

Novamente, indo ao "Gerenciador de Soluções" clicamos com o botão direito do mouse sobre "exemplosMongoDB", o nome do nosso projeto, e escolhemos a opção "Propriedades". Em "Objeto de Inicialização" selecionamos nosso programa "exemplosMongoDB.programAssincrono" e clicamos em "Iniciar", no menu superior, para que ele seja executado.

Vemos que, desta vez, a mensagem "Pressione ENTER" apareceu logo abaixo do texto "Esperando 10 segundos ....". Isso porque quando o programa chamou a função `MainSync(args)" ele automaticamente escreveu esta primeira frase, porém, como o sistema ficou suspenso por 10 segundos, paralelamente ele passou a executar os dois comandos seguintes em outra *thread*.

Ele escreveu "Pressione ENTER" antes de escrever "Esperei 10 segundos ....", ou seja, esta função foi executada paralelamente em uma outra *thread*, outro processamento, em relação ao programa principal. Isto é uma forma de programação assíncrona.

Comparando os dois programas, vemos como é possível a execução de um simples programa de maneira síncrona ou assíncrona.



Nos exemplos em que vamos trabalhar com o MongoDB será sempre utilizada a forma assíncrona.