Programação assíncrona em C#: Como e por que utilizar?

Nagib Sabbag Filho
FIAP (Faculty of Informatics and Administration Paulista) Avenida Paulista, 1106 - 7º andar - Bela Vista, São Paulo, Brazil.
e-mail: profnagib.filho@fiap.com.br
PermaLink: https://leaders.tec.br/article/programacao-assincrona-em-c-como-e-por-que-utilizar
jul 22 2024

Abstract:
A programação assíncrona em C# melhora a eficiência e escalabilidade de aplicações, permitindo a execução de múltiplas tarefas de forma concorrente sem bloquear o thread principal.

Key words:
programação assíncrona, C#, async, await, tarefas assíncronas, performance, paralelismo, processamento concorrente, awaiter

Introdução à programação assíncrona em C#

A programação assíncrona em C# é uma técnica relevante que permite que um programa execute várias tarefas de forma concorrente, sem bloquear o thread principal. Isso é extremamente útil em situações onde é necessário lidar com operações de entrada e saída (I/O) intensivas, como requisições de rede, leitura de arquivos, entre outras.

Por que utilizar programação assíncrona em C#?

A utilização de programação assíncrona em C# traz diversos benefícios, como a melhoria na responsividade de aplicações, a melhor utilização de recursos do sistema e a possibilidade de criar aplicações mais escaláveis e eficientes. Além disso, a programação assíncrona é fundamental em aplicações que precisam lidar com um grande volume de requisições simultâneas, como servidores web e sistemas de processamento em tempo real.

Como utilizar programação assíncrona em C#?

Existem várias opções para trabalharmos de forma assíncrona. No contexto desse artigo, vamos focar exclusivamente no C# e em palavras-chave específicas. Uma das formas de utilizar programação assíncrona em C# é a utilização das palavras-chave async e await em métodos que realizam operações assíncronas. Por exemplo, ao realizar uma requisição HTTP em um aplicativo C#, podemos utilizar a classe HttpClient juntamente com as palavras-chave async e await para tornar a operação assíncrona:

```
private async Task < string > GetHttpContentAsync(string url)
{
    using (HttpClient client = new HttpClient())
    {
        HttpResponseMessage response = await client.GetAsync(url);
        return await response.Content.ReadAsStringAsync();
    }
}
```

Dessa forma, o método GetHttpContentAsync irá realizar a requisição HTTP de forma assíncrona, sem bloquear o thread principal do programa.

Exemplo prático de programação assíncrona em C#

Vamos considerar um exemplo mais complexo, onde precisamos realizar múltiplas requisições HTTP de forma assíncrona e esperar que todas as requisições sejam completadas antes de continuar o processo. Neste caso, podemos utilizar o método Task.WhenAll para aguardar o término de todas as tarefas assíncronas:

```
private async Task < List < string > > GetMultipleHttpContentsAsync(List < string >
urls)
{
    List < Task < string > > tasks = new List < Task < string > > ();
    foreach (string url in urls)
    {
        tasks.Add(GetHttpContentAsync(url));
    }
    string[] results = await Task.WhenAll(tasks);
    return results.ToList();
}
```

Neste exemplo, o método GetMultipleHttpContentsAsync irá realizar múltiplas requisições HTTP de forma assíncrona e aguardar o término de todas elas antes de retornar o resultado final.

Outras formas de atuar no assíncrono com C#

Além do uso das palavras-chave async e await, existem outras abordagens e padrões para trabalhar com programação assíncrona em C#. Vamos explorar algumas dessas opções:

Uso de Task.Run para operações de CPU-bound

Para operações que demandam processamento intensivo de CPU, podemos utilizar o método Task.Run para executar essas tarefas em um thread separado, liberando o thread principal para continuar executando outras operações:

```
private async Task < int > PerformCpuBoundOperationAsync()
{
    return await Task.Run(() =>
    {
        // Simulação de uma operação que demanda processamento intensivo de CPU
        int result = 0;
        for (int i = 0; i < 1000000; i++)
        {
            result += i;
        }
        return result;
    });
}
```

Uso de IAsyncEnumerable para processamento assíncrono de streams

A interface IAsyncEnumerable permite a iteração assíncrona sobre coleções de dados, o que é útil para processamento de streams de dados que chegam de forma assíncrona:

```
private async IAsyncEnumerable GenerateSequenceAsync()
{
    for (int i = 0; i < 10; i++)
```

```
    {
        await Task.Delay(1000); // Simula uma operação assíncrona
        yield return i;
    }
}

public async Task ProcessSequenceAsync()
{
    await foreach (var number in GenerateSequenceAsync())
    {
        Console.WriteLine(number);
    }
}
```

Nesse exemplo, a sequência de números é gerada de forma assíncrona e processada à medida que os valores se tornam disponíveis, sem bloquear o thread principal.

Uso de Parallel.ForEachAsync para processamento paralelo

Em cenários onde precisamos processar um grande conjunto de dados de forma paralela, podemos utilizar o método Parallel.ForEachAsync para distribuir as tarefas entre múltiplos threads:

```
private async Task ProcessDataInParallelAsync(List data)
{
    await Parallel.ForEachAsync(data, async (item, cancellationToken) =>
    {
        await Task.Delay(100); // Simula uma operação assíncrona
        Console.WriteLine(item);
    });
}
```

Esse método distribui o processamento dos itens da lista data entre múltiplos threads, permitindo que as operações sejam executadas de forma paralela e assíncrona.

Considerações sobre desempenho e boas práticas

Ao trabalhar com programação assíncrona, é importante seguir algumas boas práticas para garantir um desempenho otimizado e evitar problemas comuns. Certifique-se de:

Evitar o uso excessivo de async e await em operações que não necessitam ser assíncronas.

Utilizar ConfigureAwait(false) em bibliotecas e componentes que não dependem do contexto de sincronização, para evitar deadlocks.

Monitorar e tratar exceções de forma adequada em métodos assíncronos.

Aproveitar a paralelização quando apropriado para melhorar o desempenho.

Conclusão

A programação assíncrona em C# é uma técnica poderosa que permite melhorar a eficiência e a escalabilidade de aplicações, especialmente em cenários onde é necessário lidar com operações de entrada e saída intensivas. Ao utilizar as palavras-chave async e await, bem como outras abordagens assíncronas, é possível tornar operações assíncronas de forma simples e intuitiva. Portanto, considere utilizar programação assíncrona em seus projetos C# para obter um desempenho otimizado e uma experiência do usuário mais fluida.

Referências

Para mais informações sobre programação assíncrona em C#, confira as seguintes referências oficiais:

MICROSOFT. Documentação oficial da Microsoft sobre programação assíncrona em C#. Disponível em: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/. Acesso em: 23 jul. 2024.
MICROSOFT. Documentação da classe HttpClient. Disponível em: https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient. Acesso em: 23 jul. 2024.
MICROSOFT. Padrões de programação assíncrona baseados em tarefas (TAP). Disponível em: https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap. Acesso em: 23 jul. 2024.
MICROSOFT. Documentação sobre Parallel.ForEachAsync. Disponível em: https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel.foreachasync. Acesso em: 23 jul. 2024.
MICROSOFT. Documentação sobre IAsyncEnumerable. Disponível em: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8#iasyncenumerable. Acesso em: 23 jul. 2024.

Nagib Filho é Professor Universitário e Tech Manager.
Possui uma trajetória de conquistas em certificações técnicas e ágeis,
incluindo MCSD, MCSA e PSM1.
PG em Gestão de TI pelo SENAC e MBA em Tecnologia de Software pela USP,
Nagib cursou programas de extensão do MIT e Universidade de Chicago.
Outras conquistas incluem a autoria de um artigo sobre chatbots, revisado por pares e apresentado na Universidade de Barcelona.