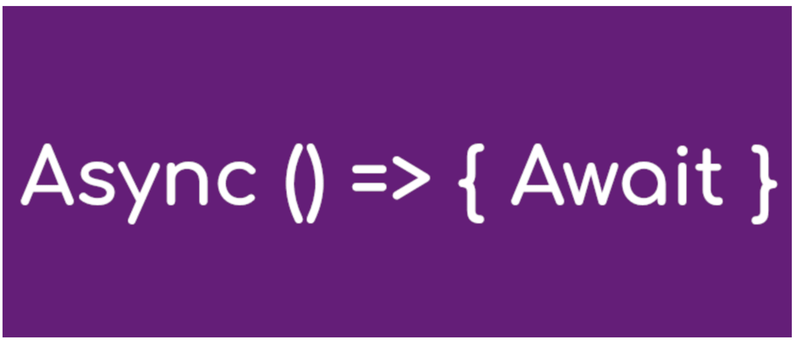
C#中await /async 的使用场景，优点，使用方法你真的知道吗?

作者：微信公众号：【架构师老卢】 11-15 16:8



概述：async/await 是 C# 中异步编程的关键特性，它使得异步代码编写更为简单和直观。下面深入详细描述了 async/await 的使用场景、优点以及一些高级使用方法，并提供了相应的实例源代码。

`async/await` 是 C# 中异步编程的关键特性，它使得异步代码编写更为简单和直观。下面深入详细描述了 `async/await` 的使用场景、优点以及一些高级使用方法，并提供了相应的实例源代码。

使用场景：

1. **I/O 操作：** 异步编程特别适用于涉及 I/O 操作（如文件读写、网络请求等）的场景。在等待 I/O 操作完成的过程中，CPU 可以继续执行其他任务，提高程序的并发性。
2. **GUI 应用程序：** 在 GUI 应用程序中，避免阻塞主线程是至关重要的。使用 `async/await` 可以确保在进行长时间运行的任务时，GUI 界面保持响应。
3. **并行编程：** 在涉及多个任务的并行编程中，`async/await` 可以简化代码的编写，提高代码的可读性和维护性。
4. **Web 服务：** 在处理 Web 请求时，异步操作可以确保服务器资源的高效利用，提高系统的吞吐量。

优点：

1. **简化异步编程：** `async/await` 使得异步编程更加直观和易于理解。代码看起来像是同步的，但实际上是异步执行的。
2. **避免阻塞：** 使用 `async/await` 可以避免在等待 I/O 操作完成时阻塞线程，提高程序的并发性。
3. **提高性能：** 在异步操作中，CPU 可以在等待的过程中执行其他任务，提高系统的整体性能。
4. **简化错误处理：** 使用 `try/catch` 结构可以捕获异步操作中的异常，使错误处理更加简便。

使用方法：

基本使用：

```
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        Console.WriteLine("Start");
        await DoAsyncTask();
        Console.WriteLine("End");
    }

    static async Task DoAsyncTask()
    {
        Console.WriteLine("Async Task Start");
        await Task.Delay(2000); // 模拟异步操作
        Console.WriteLine("Async Task End");
    }
}
```

高级使用方法：

- **并发执行多个异步任务：**

```
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        Console.WriteLine("Start");

        Task task1 = DoAsyncTask("Task 1", 2000);
        Task task2 = DoAsyncTask("Task 2", 1000);

        await Task.WhenAll(task1, task2);

        Console.WriteLine("End");
    }

    static async Task DoAsyncTask(string taskName, int delay)
    {
        Console.WriteLine($"{taskName} Start");
        await Task.Delay(delay);
        Console.WriteLine($"{taskName} End");
    }
}
```

- **取消异步操作：**

```
using System;
using System.Threading;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        Console.WriteLine("Start");

        CancellationTokenSource cts = new CancellationTokenSource();
        Task task = DoAsyncTask(cts.Token);

        // 模拟一段时间后取消任务
        await Task.Delay(1000);
        cts.Cancel();

        try
        {
            await task;
        }
        catch (TaskCanceledException)
        {
            Console.WriteLine("Task canceled");
        }

        Console.WriteLine("End");
    }

    static async Task DoAsyncTask(CancellationToken cancellationToken)
    {
        Console.WriteLine("Async Task Start");
        await Task.Delay(5000, cancellationToken); // 模拟异步操作
        Console.WriteLine("Async Task End");
    }
}
```

这些示例展示了 `async/await` 在不同场景下的使用方法，包括基本使用、并发执行多个异步任务以及取消异步操作。希望这些例子对你理解 `async/await` 的使用有所帮助。