延迟等待的艺术：C#中Task.Delay与Thread.Sleep的对比与实战

作者：微信公众号：【架构师老卢】 12-19 18:34

概述：在C#中，`Task.Delay`和`Thread.Sleep`都用于引入延迟，但`Task.Delay`适用于异步环境，不阻塞线程；而`Thread.Sleep`适用于同步环境，直接阻塞当前线程。实例源代码清晰演示了两者的用法和场景选择，帮助开发者根据需求做出明智的选择。

在C#中，`Task.Delay` 和 `Thread.Sleep` 都用于在代码中引入延迟，但它们的使用场景和效果略有不同。

1. `Task.Delay`

`Task.Delay` 是异步等待的一部分，用于在异步代码中引入延迟，而不会阻塞线程。

```
async Task SomeAsyncMethod()
{
    Console.WriteLine("Start");
    await Task.Delay(1000); // 等待1秒，不会阻塞线程
    Console.WriteLine("End");
}
```

2. `Thread.Sleep`

`Thread.Sleep` 是同步方法，会直接导致当前线程阻塞。

```
void SomeMethod()
{
    Console.WriteLine("Start");
    Thread.Sleep(1000); // 阻塞线程1秒
    Console.WriteLine("End");
}
```

3. 使用场景对比

- 使用 `Task.Delay` 适合异步环境，例如在异步方法中实现延迟。
- 使用 `Thread.Sleep` 适合在同步环境下，如控制台应用程序或单线程应用中引入延迟。

4. 实例源代码

4.1 使用 `Task.Delay`

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

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

    static async Task SomeAsyncMethod()
    {
        await Task.Delay(1000); // 异步等待1秒，不会阻塞线程
    }
}
```

4.2 使用 `Thread.Sleep`

```
using System;
using System.Threading;

class Program
{
    static void Main()
    {
        Console.WriteLine("Start");
        SomeMethod();
        Console.WriteLine("End");
    }

    static void SomeMethod()
    {
        Thread.Sleep(1000); // 同步阻塞线程1秒
    }
```

```
}
```

以上实例展示了在异步和同步环境中使用 `Task.Delay` 和 `Thread.Sleep` 的不同方式。选择使用哪一个取决于你的代码执行上下文和需求。