

HW7 2150276 沈卓成

1 ContinueWith 方法改写该程序，使得输出结果一致

Code:


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

namespace ContinueWith
{
    class Program
    {
        static int WorkItem1(object o)
        {
            Console.WriteLine("Here1 {0}", o);
            return 1000;
        }

        static async Task WorkItemAsync()
        {
            Console.WriteLine("WorkItem2 Begin");
            Task<int> ta = new Task<int>(WorkItem1, 200);
            ta.Start();
            int result = await ta.ConfigureAwait(false);
            Console.WriteLine("Here2 {0}", result);
        }

        static void Main(string[] args)
        {
            Task t = Task.Run(() => WorkItemAsync());
            t.ContinueWith(_ => Console.WriteLine("End"));
            t.Wait();
        }
    }
}
```

Ans:



```
WorkItem2 Begin
Here1 200
Here2 1000
End
```

1. 在 `Main` 函数中，使用 `Task.Run` 创建并立即启动了一个异步任务，该任务的作用是运行 `WorkItemAsync` 函数。
2. `WorkItemAsync` 函数是一个异步方法，它创建并启动了一个新的任务 `ta`。这个任务是执行 `WorkItem1` 函数，并且把200传递给这个函数。
3. `WorkItem1` 函数接收一个参数，打印一条消息，并返回一个整数值1000。
4. `WorkItemAsync` 通过 `await` 关键字等待任务 `ta` 的完成，并获取其结果。

5. 完成 `WorkItemAsync` 之后，在 `Main` 函数中使用 `ContinueWith` 启动了一个连续任务，用于在任务结束后打印"End"。
6. `t.Wait()` 使主线程等待任务 `t` 的完成。如果没有这个语句，主线程可能会在任务 `t` 完成前结束，导致任务 `t` 得不到执行。

2 书写一个具有至少两个await的异步方法并分析之执行步骤

Code:

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

class Program
{
    static async Task Main(string[] args)
    {
        Console.WriteLine("Main method started.");
        await MethodWithTwoAwaits();
        Console.WriteLine("Main method finished.");
    }

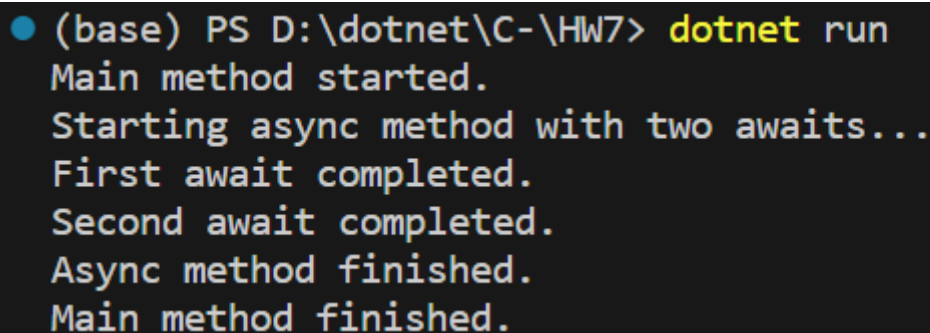
    static async Task MethodWithTwoAwaits()
    {
        Console.WriteLine("Starting async method with two awaits...");

        // First asynchronous call
        await Task.Delay(1000); // Simulates an async operation, e.g., fetching
data
        Console.WriteLine("First await completed.");

        // Second asynchronous call
        await Task.Delay(2000); // Simulates another async operation
        Console.WriteLine("Second await completed.");

        Console.WriteLine("Async method finished.");
    }
}
```

Ans:



```
● (base) PS D:\dotnet\C-\HW7> dotnet run
Main method started.
Starting async method with two awaits...
First await completed.
Second await completed.
Async method finished.
Main method finished.
```

1. 程序开始执行，调用 `Main` 方法，这是程序的入口点。
2. `Main` 方法向控制台输出 "Main method started."。
3. `Main` 方法中等待 `MethodWithTwoAwaits` 方法的执行。

4. 在 `MethodWithTwoAwaits` 方法内部，它输出 "Starting async method with two awaits..." 到控制台。
5. 遇到第一个 `await` 使用 `Task.Delay(1000)` 进行异步等待，等待时间为1秒。
6. 第一个等待完成后（延迟1秒），它输出 "First await completed."。
7. 然后继续执行下一个 `await` 使用 `Task.Delay(2000)` 进行异步等待，又等待了2秒钟。
8. 第二个等待完成后（延迟2秒），它输出 "Second await completed."。
9. `MethodWithTwoAwaits` 方法输出 "Async method finished." 表示它的执行结束。
10. 控制权返回到 `Main` 方法，接着输出 "Main method finished." 表明程序已完成所有任务的执行。

在异步等待过程中，运行 `Main` 方法的线程被释放，可以执行其他工作，直到等待的任务完成。这就是使用 `async` 和 `await` 的优势，因为它不会在等待异步操作完成时阻塞调用线程。

使用ILSpy进行分析异步函数反汇编分析

核心部分如下(我用的是vscode的插件，和VS不太一样，不是图片)：

```
.method private hidebysig specialname static void"<Main>"(
string[] args
) cil managed
.custom instance void
[System,Runtime]system.Diagnostics.DebuggerStepThroughAttribute::.ctor()=(01 88
68 88
'/ Method begins at RVA 0x20bc
'/ Header size: 12
i/ code size:28(ex14)
.maxstack 1
.entrypoint
.locals init(
[0]valuetype [System.Runtime]system.Runtime.compilerservices.TaskAwaiter
IL e888: ldarg.8IL e801:call class [system,Runtime]System.Threading.Tasks.Task
MultiAwait,Program: :Main(string!))IL C006:callvirt instance valuetype
[System,Runtime]system.Runtime.compilerservices.TaskAwaiter [system..
IL 8886:stloc.
IL 880c:ldloc.s @call instance void
[System,Runtime]system.Runtime.compilerservices.TaskAwaiter::GetResult()IL
aade:IL 8a13:rot/'/ end of method Program::"<Main>"
```

3 TaskCompletionSource for I_O-Bound In .Net

Code:

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

namespace TaskCompletionSource
{
    public class AsyncWebClient
    {
        public static Task<byte[]> GetDataAsync(string uri)
        {
            // create completion source
            var tcs = new TaskCompletionSource<byte[]>();
            // create a web client for downloading the string
```

```

var wc = new WebClient();
// Subscribe to the completed event
wc.DownloadDataCompleted += (s, e) =>
{
    if (e.Error != null)
    {
        // If the download failed, set the error on the
taskcompletion source
        tcs.TrySetException(e.Error);
    }
    else if (e.Cancelled)
    {
        // If the download was cancelled, signal cancellation to the
task completion source
        tcs.TrySetCanceled();
    }
    else
    {
        // If the download was successful, set the result on the
task completion source
        tcs.TrySetResult(e.Result);
    }
    wc.Dispose();
};
// Start the asynchronous download
wc.DownloadDataAsync(new Uri(uri));
// Returns after the value has been added by trySetResult
return tcs.Task;
}
}
class Program
{
    static async Task Main(string[] args)
    {
        await AsyncWebClient.GetDataAsync("https://www.sina.com.cn");
        Console.WriteLine("Operation is completed");
    }
}
}

```

Ans:

```

● (base) PS D:\dotnet\C-\HW7> dotnet run
D:\dotnet\C-\HW7\3.cs(14,22): warning SYSLIB0014: "WebClient.WebClient()"已过时:"WebRequest, HttpWebRequest, and WebClient are obsolete. Use HttpClient instead." (https://aka.ms/dotnet-warnings/SYSLIB0014)
Operation is completed

```

这段代码定义了一个简单的异步网络客户端，用于异步地从指定的URI下载数据，并在下载完成后通过任务（Task）返回数据。代码结构包括两部分：AsyncWebClient 类和 Program 类的主入口。

AsyncWebClient 类中有一个静态方法 GetDataAsync，该方法接受一个 uri 字符串参数：

1. 首先创建一个 TaskCompletionSource 对象，它允许你手动控制一个 Task 的状态和结果。这个 Task 最终会用来返回下载的数据。
2. 接着创建一个 WebClient 实例，用于执行数据下载。
3. 为 WebClient 的 DownloadDataCompleted 事件订阅一个事件处理器。这个事件处理器会在数据下载完成时调用。

4. 事件处理器检查下载过程中是否发生了错误或者是否被取消：

- 如果有错误发生，它调用 `TaskCompletionSource` 的 `TrySetException` 方法，并将错误传递给任务。
- 如果下载被取消，它调用 `TaskCompletionSource` 的 `TrySetCanceled` 方法，以标记任务为取消状态。
- 如果下载成功完成，它调用 `TaskCompletionSource` 的 `TrySetResult` 方法，并将下载的数据作为结果传递给任务。

在这之后，`WebClient` 对象被释放（`Dispose` 调用）来清理资源。

5. `wc.DownloadDataAsync` 方法随即开始对指定 URI 的异步数据下载。

6. 方法返回一个 `Task`，它表示数据下载的异步操作。调用者可以等待这个任务完成，从而获得下载的数据。

在 `Program` 类的 `Main` 方法里（这是整个程序的入口点）：

1. 使用 `await` 操作符调用 `AsyncWebClient` 类的 `GetDataAsync` 方法，并传递 `https://www.sina.com.cn` 作为参数。这个调用将会暂停 `Main` 方法的执行，直到与 `GetDataAsync` 方法所返回的 `Task` 相关联的操作完成。
2. 一旦 `GetDataAsync` 方法完成，控制台会输出 `"Operation is completed"`，标明下载操作已经完成。

整个程序的目的是演示如何使用 `TaskCompletionSource` 来封装一个原本基于事件驱动的异步模型（`WebClient` 的 `DownloadDataAsync`）到一个基于任务的异步模型（返回 `Task`），从而使得可以使用 `async` 和 `await` 简化异步编程。