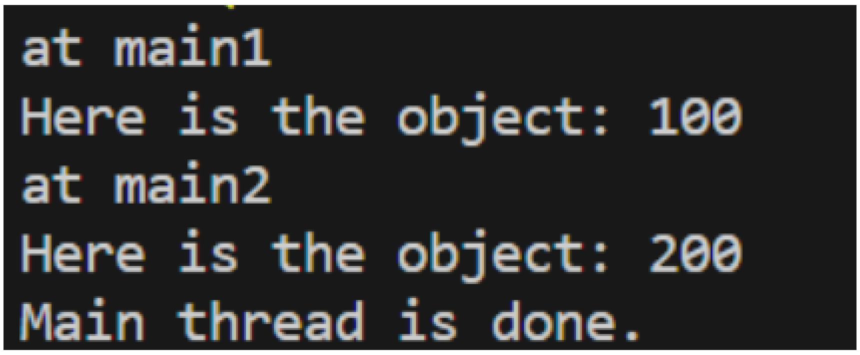
1 线程池 WorkItem 到 Task

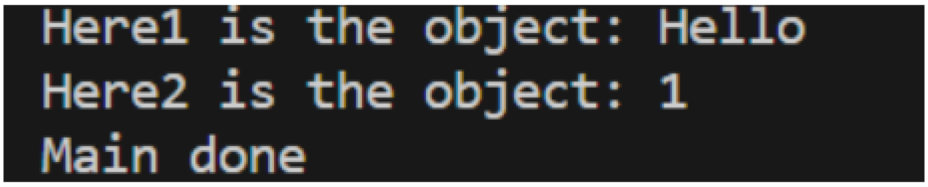
```
using System;

class Program
{
    static void WorkItem(object o)
    {
        Console.WriteLine("Here is the object: {0}", o);
    }
    static void Main(String[] args)
    {
        ThreadPool.QueueUserWorkItem(WorkItem, 100);
        Console.WriteLine("at main1");
        Thread.Sleep(1000);

        Task t = new Task(WorkItem, 200);
        t.Start();
        Console.WriteLine("at main2");
        t.Wait();
        Console.WriteLine("Main thread is done.");
    }
}
```



```
at main1
Here is the object: 100
at main2
Here is the object: 200
Main thread is done.
```

2 在任务完成时可以安排自动启动另一个新的任务

```
using System;

class Program
{
    static int WorkItem1(object o)
    {
        Console.WriteLine("Here1 is the object: {0}", o);
        return 1;
    }
    static void WorkItem2(Task<int> t)
    {   Console.WriteLine("Here2 is the object: {0}", t.Result);

    }
    static void Main(String[] args)
    {
        Task<int> ta = new Task<int>(WorkItem1, "Hello");
        Task tb = ta.ContinueWith(t=>WorkItem2(t));
        ta.Start();
        tb.Wait();
        Console.WriteLine("Main done");
    }
}
```

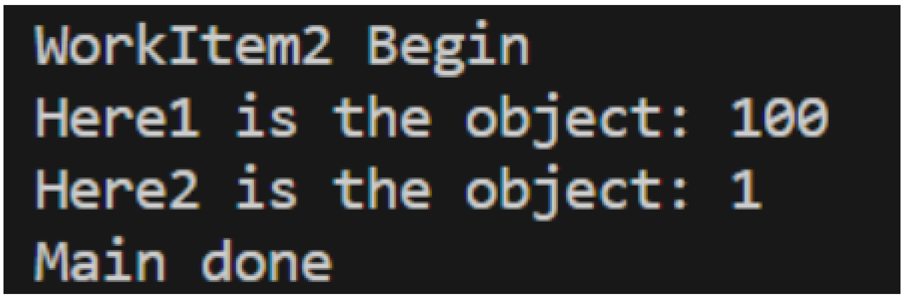


```
Here1 is the object: Hello
Here2 is the object: 1
Main done
```

使用 `Task.ContinueWith` 方法时，传递给它的 lambda 表达式的参数是一个 `Task` 类型的对象。这个 `Task` 对象代表了之前被 `ContinueWith` 方法链接的任务。`ContinueWith` 的参数类型仍然是 `Task<T>`，其中 `T` 是前一个任务的结果类型。

3 使用 async 和 await 进行异步编程

```
using System;

class Program
{
    static int WorkItem1(object o)
    {
        Console.WriteLine("Here1 is the object: {0}", o);
        return 1;
    }
    static async Task WorkItemAsync()
    {
        Console.WriteLine("WorkItem2 Begin");
        Task<int> ta = new Task<int>(WorkItem1, 100);
        ta.Start();
        var result = await ta;
        Console.WriteLine("Here2 is the object: {0}", result);
    }
    static void Main(String[] args)
    {
        Task t = WorkItemAsync();
        t.Wait();
        Console.WriteLine("Main done");
    }
}
```



```
WorkItem2 Begin
Here1 is the object: 100
Here2 is the object: 1
Main done
```

Q1 调用 async 修饰的异步方法，该方法在哪里执行?

Ans:

通过使用`await`关键字，异步方法会等待`ta`任务的完成，并在任务完成后继续执行剩余的代码。一旦`ta`任务完成，`WorkItemAsync`方法会恢复执行。

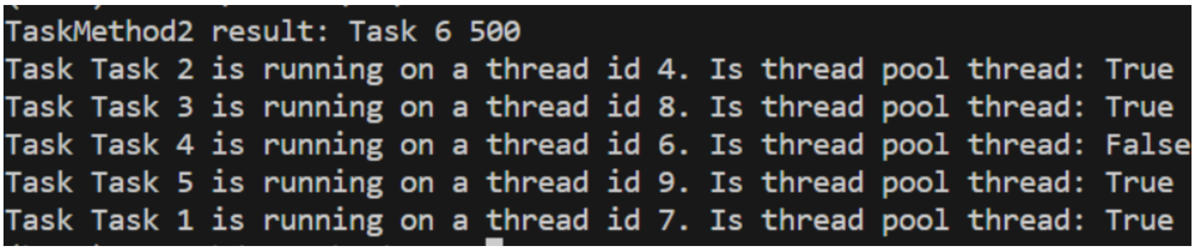
Q2 await 是否会阻塞当前线程?

Ans:

使用`await`关键字时，当前线程不会被阻塞，而是允许其他代码在当前线程上继续执行，直到异步操作完成后再继续执行`await`之后的代码。

4 创建任务

```
using System;
using System.Threading.Tasks;
using static System.Console;
using static System.Threading.Thread;

class Program
{
    static void TaskMethod(string name)
    {
        WriteLine($"Task {name} is running on a thread id " +
                  $"{CurrentThread.ManagedThreadId}. Is thread pool thread: " +
                  $"{CurrentThread.IsThreadPoolThread}");
    }

    static string TaskMethod2(string name)
    {
        return name + 500.ToString();
    }

    static void Main(string[] args)
    {
        //方式1: 创建任务对象并且启动,如果不调用Start，任务不会被执行
        var t1 = new Task(() => TaskMethod("Task 1"));
        var t2 = new Task(() => TaskMethod("Task 2"));
        t2.Start();
        t1.Start();

        //方式2: 创建任务并自动立即开始执行，可以提供一个参数
        Task.Factory.StartNew(() => TaskMethod("Task 3"));

        //标记为长时间运行的任务将不会使用线程池，而在单独的线程中运行
        Task.Factory.StartNew(() => TaskMethod("Task 4"),
TaskCreationOptions.LongRunning);
        //方式3: Task.Run是Task.Factory.StartNew的一个快捷方式
        Task.Run(() => TaskMethod("Task 5"));

        Task<string> t3 = Task.Run(() => TaskMethod2("Task 6 "));
        //t3.Wait();
        WriteLine($"TaskMethod2 result: {t3.Result}");

        Sleep(TimeSpan.FromSeconds(1));
    }
}
```

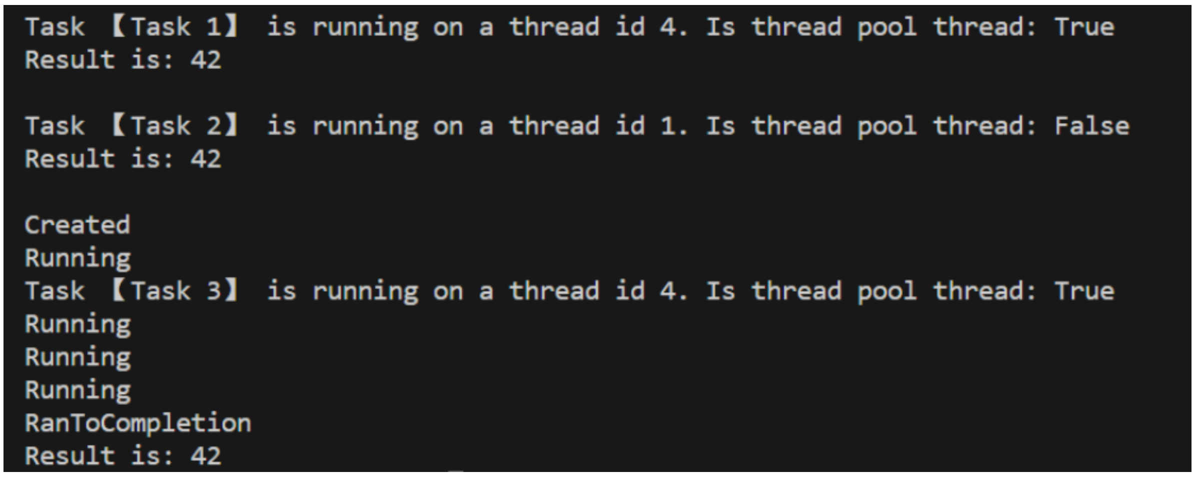


```
TaskMethod2 result: Task 6 500
Task Task 2 is running on a thread id 4. Is thread pool thread: True
Task Task 3 is running on a thread id 8. Is thread pool thread: True
Task Task 4 is running on a thread id 6. Is thread pool thread: False
Task Task 5 is running on a thread id 9. Is thread pool thread: True
Task Task 1 is running on a thread id 7. Is thread pool thread: True
```

1. 自动开始执行的任务：使用`Task.Factory.StartNew()`方法创建并自动开始执行任务。在代码中，`Task 3`是通过这种方式创建的任务。
2. 长时间运行的任务：通过在`Task.Factory.StartNew()`方法中使用`TaskCreationOptions.LongRunning`选项，可以将任务标记为长时间运行的任务。这些任务将在单独的线程中执行，而不是使用线程池线程。在代码中，`Task 4`是一个被标记为长时间运行的任务。
3. 使用`Task.Run()`方法：`Task.Run()`是`Task.Factory.StartNew()`方法的一个快捷方式，用于创建并开始执行任务。在代码中，`Task 5`是通过这种方式创建的任务。

5 运行任务并得到结果

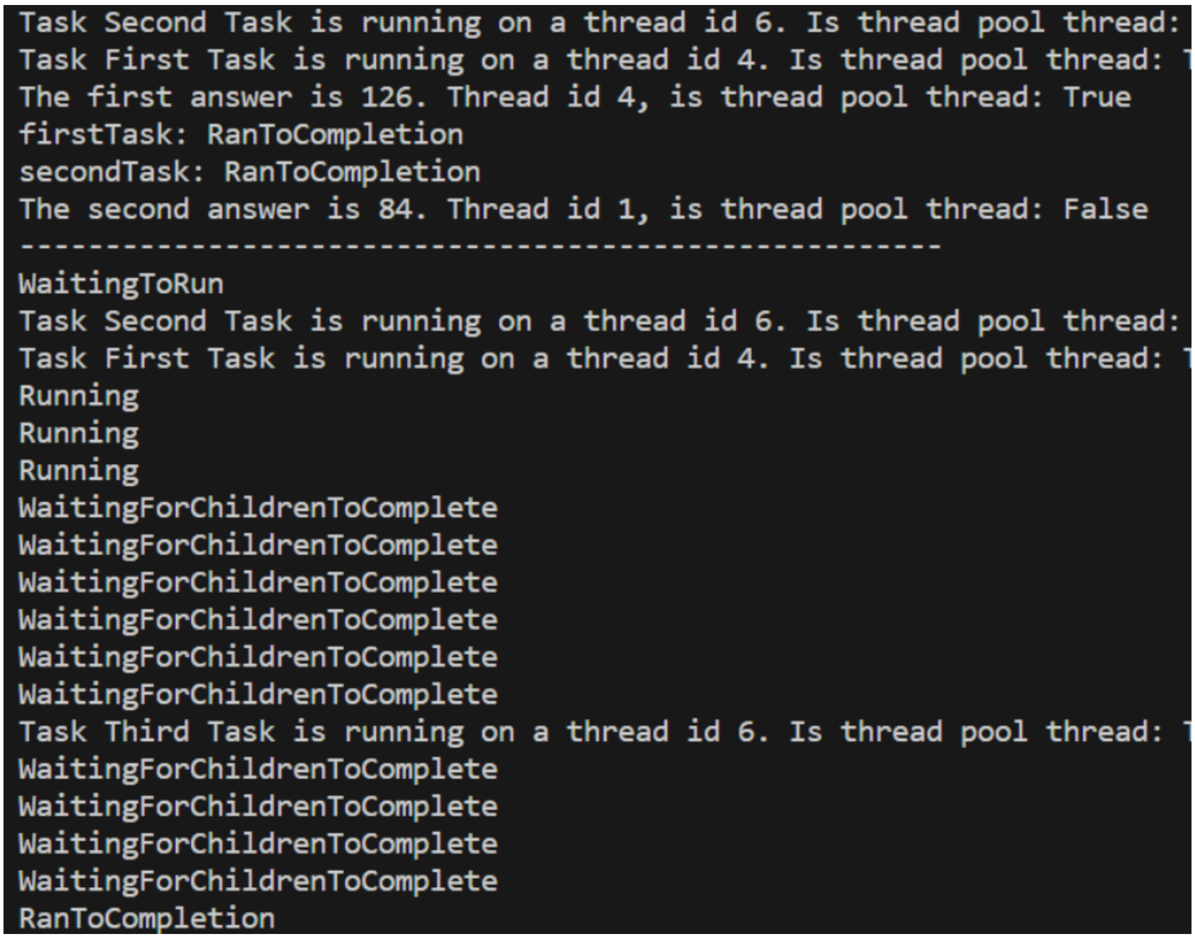
```
using System;
using System.Threading.Tasks;
using static System.Console;
using static System.Threading.Thread;

class Program
{
    static int TaskMethod(string name)
    {
        WriteLine($"Task 【{name}】 is running on a thread id " +
                  $"{CurrentThread.ManagedThreadId}. Is thread pool thread: " +
                  $"{CurrentThread.IsThreadPoolThread}");
        Sleep(TimeSpan.FromSeconds(2));
        return 42;
    }
    static Task<int> CreateTask(string name)
    {
        return new Task<int>(() => TaskMethod(name));
    }
    static void Main(string[] args)
    {         //(1)在主线程直接调用方法
        TaskMethod("Main Thread Task");
        WriteLine("");

        //(2)创建一个任务,然后正常执行，得到结果
        Task<int> task = CreateTask("Task 1");
        task.Start();
        int result = task.Result;
        WriteLine($"Result is: {result}\r\n");
        //return;

        //(3)创建任务,使用RunSynchronously进行执行,得到结果
        task = CreateTask("Task 2");
        task.RunSynchronously();
        result = task.Result;
        WriteLine($"Result is: {result}\r\n");

        //(4)观察任务的运行状态
        task = CreateTask("Task 3");
        WriteLine(task.Status);
        task.Start();
        while (!task.IsCompleted)
        {
            WriteLine(task.Status);
            Sleep(TimeSpan.FromSeconds(0.5));
        }
        WriteLine(task.Status);
        result = task.Result;
        WriteLine($"Result is: {result}");
    }
}
```



```
Task 【Task 1】 is running on a thread id 4. Is thread pool thread: True
Result is: 42

Task 【Task 2】 is running on a thread id 1. Is thread pool thread: False
Result is: 42

Created
Running
Task 【Task 3】 is running on a thread id 4. Is thread pool thread: True
Running
Running
Running
RanToCompletion
Result is: 42
```

6 组合任务

```
using System;
using System.Threading.Tasks;
using static System.Console;
using static System.Threading.Thread;

class Program
{
    static int TaskMethod(string name, int seconds)
    {
        WriteLine(
            $"Task {name} is running on a thread id " +
            $"{CurrentThread.ManagedThreadId}. Is thread pool thread: " +
            $"{CurrentThread.IsThreadPoolThread}");
        Sleep(TimeSpan.FromSeconds(seconds));

        //throw new Exception("err");

        return 42 * seconds;
    }

    static void Main(string[] args)
    {
        var firstTask  = new Task<int>(() => TaskMethod("First Task", 3));
        var secondTask = new Task<int>(() => TaskMethod("Second Task", 2));

        //使用ContinueWith组合任务
        firstTask.ContinueWith(
            t => WriteLine(
                $"The first answer is {t.Result}. Thread id " +
                $"{CurrentThread.ManagedThreadId}, is thread pool thread: " +
                $"{CurrentThread.IsThreadPoolThread}"),
            TaskContinuationOptions.OnlyOnRanToCompletion);
        // `TaskContinuationOptions.OnlyOnRanToCompletion` 的作用是确保延续操作只在前
一个任务成功完成时（即没有抛出异常）执行。

        firstTask.Start();
        secondTask.Start();

        Sleep(TimeSpan.FromSeconds(4));



        //firstTask: Faulted
        //secondTask: Faulted
        //firstTask: RanToCompletion
        //secondTask: RanToCompletion
        WriteLine($"firstTask: {firstTask.Status}");
        WriteLine($"secondTask: {secondTask.Status}");


        Task continuation = secondTask.ContinueWith(
            t => WriteLine(
                $"The second answer is {t.Result}. Thread id " +
                $"{CurrentThread.ManagedThreadId}, is thread pool thread: " +
                $"{CurrentThread.IsThreadPoolThread}"),
            TaskContinuationOptions.OnlyOnRanToCompletion
            | TaskContinuationOptions.ExecuteSynchronously);

        Sleep(TimeSpan.FromSeconds(2));
        WriteLine("------------------------------------------------------");


        //return;

        firstTask = new Task<int>(() =>
        {
            var innerTask = Task.Factory.StartNew(() => TaskMethod("Second
Task", 5),
                TaskCreationOptions.AttachedToParent);

            innerTask.ContinueWith(t => TaskMethod("Third Task", 2),
                TaskContinuationOptions.AttachedToParent);

            return TaskMethod("First Task", 2);
        });

        firstTask.Start();

        while (!firstTask.IsCompleted)
        {
            WriteLine(firstTask.Status);
            Sleep(TimeSpan.FromSeconds(0.5));
        }
        WriteLine(firstTask.Status);

        Sleep(TimeSpan.FromSeconds(10));
    }
}
```

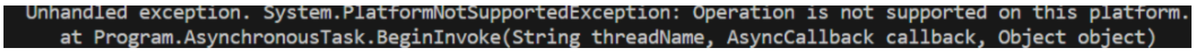


```
Task Second Task is running on a thread id 6. Is thread pool thread:
Task First Task is running on a thread id 4. Is thread pool thread: T
The first answer is 126. Thread id 4, is thread pool thread: True
firstTask: RanToCompletion
secondTask: RanToCompletion
The second answer is 84. Thread id 1, is thread pool thread: False
--------------------------------------------------------
WaitingToRun
Task Second Task is running on a thread id 6. Is thread pool thread:
Task First Task is running on a thread id 4. Is thread pool thread: T
Running
Running
Running
WaitingForChildrenToComplete
WaitingForChildrenToComplete
WaitingForChildrenToComplete
WaitingForChildrenToComplete
WaitingForChildrenToComplete
WaitingForChildrenToComplete
Task Third Task is running on a thread id 6. Is thread pool thread: T
WaitingForChildrenToComplete
WaitingForChildrenToComplete
WaitingForChildrenToComplete
WaitingForChildrenToComplete
RanToCompletion
```

1. ContinueWith方法：使用`ContinueWith`方法来创建任务之间的延续关系。
2. 通过`TaskCreationOptions.AttachedToParent`和`TaskContinuationOptions.AttachedToParent`来创建父子任务关系。通过`Task.Factory.StartNew`方法创建子任务，并与父任务关联。

7 APM To Task

```
using System;
using System.Threading.Tasks;
using static System.Console;
using static System.Threading.Thread;

class Program
{
    delegate string AsynchronousTask(string threadName);
    delegate string IncompatibleAsynchronousTask(out int threadId);

    static void Callback(IAsyncResult ar)
    {
        WriteLine("Starting a callback...");
        WriteLine($"State passed to a callbak: {ar.AsyncState}");
        WriteLine($"Is thread pool thread: {CurrentThread.IsThreadPoolThread}");
        WriteLine($"Thread pool worker thread id:
{CurrentThread.ManagedThreadId}");
    }

    static string Test(string threadName)
    {
        WriteLine("Starting...");
        WriteLine($"Is thread pool thread: {CurrentThread.IsThreadPoolThread}");
        Sleep(TimeSpan.FromSeconds(2));
        CurrentThread.Name = threadName;
        return $"Thread name: {CurrentThread.Name}";
    }

    static string Test(out int threadId)
    {
        WriteLine("Starting...");
        WriteLine($"Is thread pool thread: {CurrentThread.IsThreadPoolThread}");
        Sleep(TimeSpan.FromSeconds(2));
        threadId = CurrentThread.ManagedThreadId;
        return $"Thread pool worker thread id was: {threadId}";
    }

    static void Main(string[] args)
    {
        int threadId;
        AsynchronousTask d             = Test;
        IncompatibleAsynchronousTask e = Test;

        WriteLine("Option 1");

        //FromAsync(IAsyncResult, Action<IAsyncResult>)
        //创建一个 Task，它在指定的 IAsyncResult 完成时执行一个结束方法操作
        Task<string> task = Task<string>.Factory.FromAsync(
            d.BeginInvoke("AsyncTaskThread", Callback,
            "a delegate asynchronous call"), d.EndInvoke);

        task.ContinueWith(t => WriteLine(
            $"Callback is finished, now running a continuation! Result:
{t.Result}"));

        while (!task.IsCompleted)
        {
            WriteLine(task.Status);
            Sleep(TimeSpan.FromSeconds(0.5));
        }
        WriteLine(task.Status);
        Sleep(TimeSpan.FromSeconds(1));


        WriteLine("----------------------------------------------");
        WriteLine();
        WriteLine("Option 2");

        task = Task<string>.Factory.FromAsync(
            d.BeginInvoke, d.EndInvoke, "AsyncTaskThread", "a delegate
asynchronous call");

        task.ContinueWith(t => WriteLine(
            $"Task is completed, now running a continuation! Result:
{t.Result}"));
        while (!task.IsCompleted)
        {
            WriteLine(task.Status);
            Sleep(TimeSpan.FromSeconds(0.5));
        }
        WriteLine(task.Status);
        Sleep(TimeSpan.FromSeconds(1));

        WriteLine("----------------------------------------------");
        WriteLine();
        WriteLine("Option 3");


        IAsyncResult ar = e.BeginInvoke(out threadId, Callback, "a delegate
asynchronous call");
        task = Task<string>.Factory.FromAsync(ar, _ => e.EndInvoke(out threadId,
ar));

        task.ContinueWith(t =>
            WriteLine(
                $"Task is completed, now running a continuation! " +
                $"Result: {t.Result}, ThreadId: {threadId}"));

        while (!task.IsCompleted)
        {
            WriteLine(task.Status);
            Sleep(TimeSpan.FromSeconds(0.5));
        }
        WriteLine(task.Status);

        Sleep(TimeSpan.FromSeconds(1));
    }
}
```

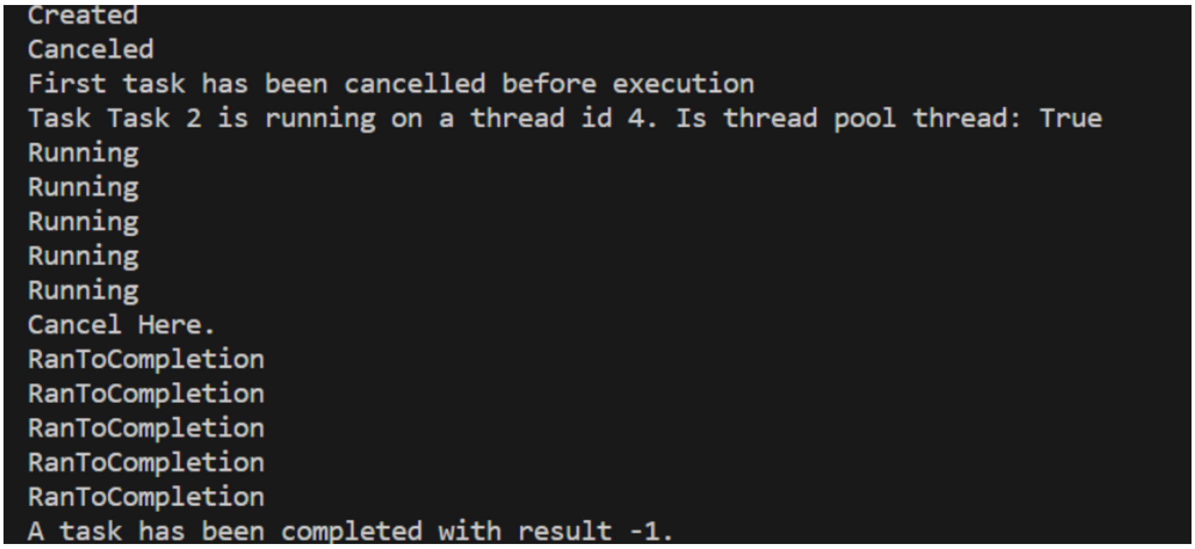


```
Unhandled exception. System.PlatformNotSupportedException: Operation is not supported on this platform.
   at Program.AsynchronousTask.BeginInvoke(String threadName, AsyncCallback callback, Object object)
```

APM在.NET 8已经不支持相关函数

8 取消任务执行

```
using System;
using System.Threading;
using System.Threading.Tasks;
using static System.Console;
using static System.Threading.Thread;

class Program
{
    static int TaskMethod(string name, int seconds, CancellationToken token)
    {
        WriteLine(
            $"Task {name} is running on a thread id " +
            $"{CurrentThread.ManagedThreadId}. Is thread pool thread: " +
            $"{CurrentThread.IsThreadPoolThread}");
        for (int i = 0; i < seconds; i ++)
        {
            Sleep(TimeSpan.FromSeconds(1));
            if (token.IsCancellationRequested)
            {
                WriteLine("Cancel Here.");
                return -1;
            }
        }
        return 42*seconds;
    }

    static void Main(string[] args)
    {
        //参数cts.Token传递两次的原因:如果任务没有开始就被取消，需要有TPL基础设施处理取消操
作

        var cts = new CancellationTokenSource();
        var longTask = new Task<int>(() => TaskMethod("Task 1", 10, cts.Token),
cts.Token);
        WriteLine(longTask.Status);
        cts.Cancel();
        WriteLine(longTask.Status);
        WriteLine("First task has been cancelled before execution");

        cts = new CancellationTokenSource();
        longTask = new Task<int>(() => TaskMethod("Task 2", 10, cts.Token),
cts.Token);
        longTask.Start();
        for (int i = 0; i < 5; i++ )
        {
            Sleep(TimeSpan.FromSeconds(0.5));
            WriteLine(longTask.Status);
        }
        cts.Cancel();
        for (int i = 0; i < 5; i++)
        {
            Sleep(TimeSpan.FromSeconds(0.5));
            WriteLine(longTask.Status);
        }
        WriteLine($"A task has been completed with result {longTask.Result}.");
    }
}
```



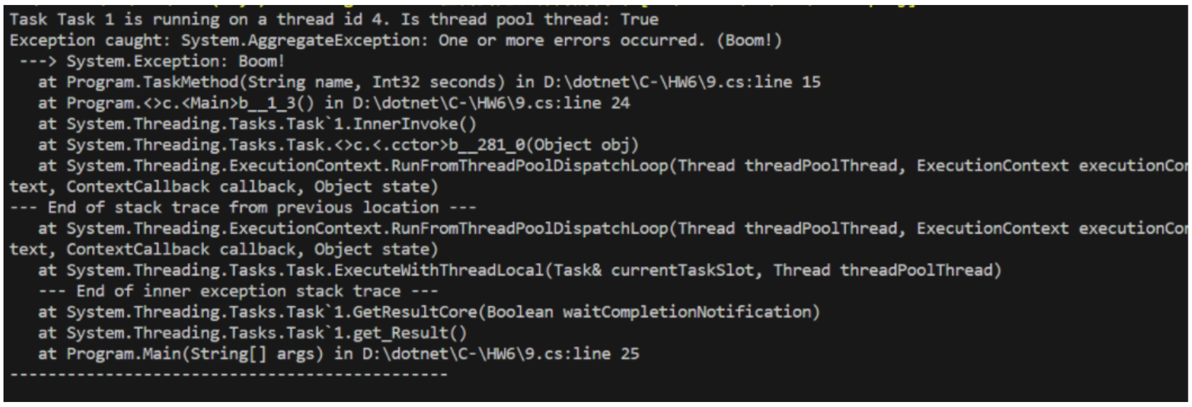
```
Created
Canceled
First task has been cancelled before execution
Task Task 2 is running on a thread id 4. Is thread pool thread: True
Running
Running
Running
Running
Running
Cancel Here.
RanToCompletion
RanToCompletion
RanToCompletion
RanToCompletion
RanToCompletion
A task has been completed with result -1.
```

1. `CancellationTokenSource`用于创建取消标记，`CancellationToken`用于将取消标记传递给任务方法。
2. `token.IsCancellationRequested`检查是否请求取消任务。如果请求取消，则可以通过返回特定值或抛出异常来取消任务的执行。
3. 创建`CancellationTokenSource`对象来创建取消标记源，并将取消标记传递给任务方法。
4. 在任务未开始执行之前，可以调用`cts.Cancel()`方法来取消任务。

9 任务异常处理

```
using System;
using System.Threading.Tasks;
using static System.Console;
using static System.Threading.Thread;

class Program
{
    static int TaskMethod(string name, int seconds)
    {        WriteLine(
            $"Task {name} is running on a thread id " +
            $"{CurrentThread.ManagedThreadId}. Is thread pool thread: " +
            $"{CurrentThread.IsThreadPoolThread}");

        Sleep(TimeSpan.FromSeconds(seconds));
        throw new Exception("Boom!");
        return 42 * seconds;
    }

    static void Main(string[] args)
    {        //这个这样捕捉到任务产生的异常
        Task<int> task;
        try
        {
            task = Task.Run(() => TaskMethod("Task 1", 2));
            int result = task.Result;
            WriteLine($"Result: {result}");
        }        catch (Exception ex)
        {            WriteLine($"Exception caught: {ex}");
        }        WriteLine("----------------------------------------------");
        WriteLine();

       var t1 = new Task<int>(() => TaskMethod("Task 3", 3));
```

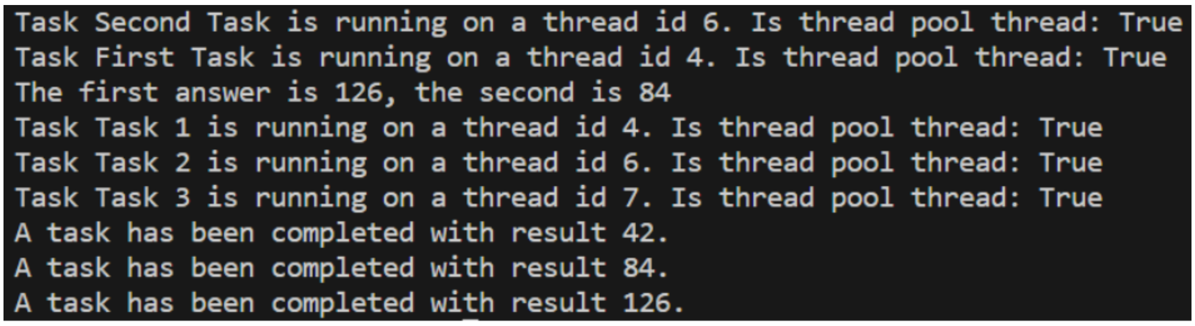
```
        var t2 = new Task<int>(() => TaskMethod("Task 4", 2));
        var complexTask = Task.WhenAll(t1, t2); // Task.WhenAll 方法会等待所有的任务
都完成后才会继续执行
        var exceptionHandler = complexTask.ContinueWith(t =>
                WriteLine($"Exception caught: {t.Exception}"),
TaskContinuationOptions.OnlyOnFaulted // exceptionHandler 任务会在 complexTask 发生
异常时执行
        );
        t1.Start();
        t2.Start();

        Sleep(TimeSpan.FromSeconds(5));
    }}
```



```
Task Task 1 is running on a thread id 4. Is thread pool thread: True
Exception caught: System.AggregateException: One or more errors occurred. (Boom!)
 ---> System.Exception: Boom!
   at Program.TaskMethod(String name, Int32 seconds) in D:\dotnet\C-\HW6\9.cs:line 15
   at Program.<>c.<Main>b__1_3() in D:\dotnet\C-\HW6\9.cs:line 24
   at System.Threading.Tasks.Task`1.InnerInvoke()
   at System.Threading.Tasks.Task.<>c.<.cctor>b__281_0(Object obj)
   at System.Threading.ExecutionContext.RunFromThreadPoolDispatchLoop(Thread threadPoolThread, ExecutionContext executionCon
text, ContextCallback callback, Object state)
--- End of stack trace from previous location ---
   at System.Threading.ExecutionContext.RunFromThreadPoolDispatchLoop(Thread threadPoolThread, ExecutionContext executionCon
text, ContextCallback callback, Object state)
   at System.Threading.Tasks.Task.ExecuteWithThreadLocal(Task& currentTaskSlot, Thread threadPoolThread)
   --- End of inner exception stack trace ---
   at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotification)
   at System.Threading.Tasks.Task`1.get_Result()
   at Program.Main(String[] args) in D:\dotnet\C-\HW6\9.cs:line 25
----------------------------------------------
```

10 等待任务完成

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using static System.Console;
using static System.Threading.Thread;

class Program
{
    static int TaskMethod(string name, int seconds)
    {
        WriteLine(
            $"Task {name} is running on a thread id " +
            $"{CurrentThread.ManagedThreadId}. Is thread pool thread: " +
            $"{CurrentThread.IsThreadPoolThread}");

        Sleep(TimeSpan.FromSeconds(seconds));
        return 42 * seconds;
    }

    static void Main(string[] args)
    {
        var firstTask = new Task<int>(() => TaskMethod("First Task", 3));
        var secondTask = new Task<int>(() => TaskMethod("Second Task", 2));

        //(1)使用WhenAll等待所有任务完成

        var whenAllTask = Task.WhenAll(firstTask, secondTask);

        whenAllTask.ContinueWith(t =>
            WriteLine($"The first answer is {t.Result[0]}, the second is
{t.Result[1]}"),
            TaskContinuationOptions.OnlyOnRanToCompletion);

        firstTask.Start();
        secondTask.Start();

        Sleep(TimeSpan.FromSeconds(4));

        //(2)使用WhenAll等待所有任务完成
        var tasks = new List<Task<int>>();
        for (int i = 1; i < 4; i++)
        {
            int counter = i;
            var task = new Task<int>(() => TaskMethod($"Task {counter}",
counter));
            tasks.Add(task);
            task.Start();
        }

        while (tasks.Count > 0)
        {
            var completedTask = Task.WhenAny(tasks).Result;
            tasks.Remove(completedTask);
            WriteLine($"A task has been completed with result
{completedTask.Result}.");
        }

        Sleep(TimeSpan.FromSeconds(1));
    }
}
```



```
Task Second Task is running on a thread id 6. Is thread pool thread: True
Task First Task is running on a thread id 4. Is thread pool thread: True
The first answer is 126, the second is 84
Task Task 1 is running on a thread id 4. Is thread pool thread: True
Task Task 2 is running on a thread id 6. Is thread pool thread: True
Task Task 3 is running on a thread id 7. Is thread pool thread: True
A task has been completed with result 42.
A task has been completed with result 84.
A task has been completed with result 126.
```

1. 使用了`Task.WhenAll`方法来创建一个新的任务，这个新任务会在firstTask 和 secondTask这两个任务都完成的时候完成，并在任务完成后输出它们的执行结果。
2. 调用这两个任务的`Start`方法来启动它们。
3. 然后又创建了三个任务并启动，使用`Task.WhenAny`方法来等待任何一个任务完成，并输出完成任务的执行结果，这个过程会持续到所有任务都完成。