任务并行库(Task Parallel Library，简称TPL)是线程池之上的一个抽象层，这个库对程序员隐藏了与线程池交互的底层代码，并提供了更方便的细粒度API。

图形用户界面, 表格

中度可信度描述已自动生成

图形用户界面, 应用程序

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

文本

中度可信度描述已自动生成

图片包含 文本

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成



图形用户界面, 应用程序

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面

低可信度描述已自动生成

图形用户界面, 应用程序, 表格

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

文档：[Task 类 (System.Threading.Tasks) | Microsoft Learn](https://learn.microsoft.com/zh-cn/dotnet/api/system.threading.tasks.task?view=net-8.0)

创建任务：

使用System.Threading.Tasks命名空间

例子：

图片包含 文本

描述已自动生成

使用Task传入一个委托创建的线程，需要调用Start方法才能开始运行；使用Task.Run和Task.Factory.StartNot传入委托创建的线程会直接运行，这两者中的后者的重载，第二个参数标记任务为长时间运行，因此不会使用线程池创建线程。

使用任务执行基本的操作：

电脑屏幕的截图

描述已自动生成

图形用户界面

描述已自动生成

使用RunSynchronously方法运行的Task任务线程不在线程池中。Task.Status属性返回任务状态，分别是Creatd，Running，RanToCompletion。Task.Result返回该任务的返回值

组合任务：

组合任务是相互依赖的任务，比如创建一个父任务和子任务，在父任务完成后自动运行子任务

Task<T>.ContinueWith方法用于创建一个新的任务，在当前任务完成后立即执行这个任务。

TaskContinueOptions：为通过使用 [ContinueWith(Action<Task>, CancellationToken, TaskContinuationOptions, TaskScheduler)](https://learn.microsoft.com/zh-cn/dotnet/api/system.threading.tasks.task.continuewith?view=net-7.0" \l "system-threading-tasks-task-continuewith(system-action((system-threading-tasks-task))-system-threading-cancellationtoken-system-threading-tasks-taskcontinuationoptions-system-threading-tasks-taskscheduler)) 或 [ContinueWith(Action<Task<TResult>>, TaskContinuationOptions)](https://learn.microsoft.com/zh-cn/dotnet/api/system.threading.tasks.task-1.continuewith?view=net-7.0" \l "system-threading-tasks-task-1-continuewith(system-action((system-threading-tasks-task((-0))))-system-threading-tasks-taskcontinuationoptions)) 方法创建的任务指定行为。此枚举支持其成员值的按位组合。

Ps：子任务必须在父任务运行时创建并正确地附加给父任务

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

文档：[TaskContinuationOptions 枚举 (System.Threading.Tasks) | Microsoft Learn](https://learn.microsoft.com/zh-cn/dotnet/api/system.threading.tasks.taskcontinuationoptions?view=net-7.0" \l "system-threading-tasks-taskcontinuationoptions-executesynchronously)

将APM模式转化为任务：

使用Task<T>.Factory.FromAsync方法，T为异步操作结果的类型。

方法的作用：创建一个 Task，表示符合异步编程模型模式的成对的开始和结束方法。

图形用户界面

中度可信度描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

文档：[TaskFactory.FromAsync 方法 (System.Threading.Tasks) | Microsoft Learn](https://learn.microsoft.com/zh-cn/dotnet/api/system.threading.tasks.taskfactory.fromasync?view=net-7.0#system-threading-tasks-taskfactory-fromasync(system-func((system-asynccallback-system-object-system-iasyncresult))-system-action((system-iasyncresult))-system-object))

将EAP模式转换为任务（基于事件的异步操作转换为任务）：

使用TaskCompletionSource<T>类型

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

文档：[TaskCompletionSource 类 (System.Threading.Tasks) | Microsoft Learn](https://learn.microsoft.com/zh-cn/dotnet/api/system.threading.tasks.taskcompletionsource?view=net-6.0)

给基于任务的异步操作实现取消流程：

书上的例子：

文本, 信件

描述已自动生成

文本

描述已自动生成

解释：

文本

描述已自动生成

Ps：若不传入标志直接调用Cancel方法可能导致一些问题，比如异步操作可能无法正确地响应取消请求，导致它继续执行而不进行必要的清理或提前结束。若仅传入标志而不调用Cancel方法则需要自定义取消逻辑，通过循环反复判断IsCancellationRequested属性是否为true来执行语句块内自定义的取消逻辑。

处理任务中的异常：

1.使用try-catch语句块，使用同步的方式捕获并处理异常

2.使用GetAwaiter和GetResult捕获异常，但是缺点是一次只能捕获一个原始异常，是第一个异常。

3.使用书上例子中的第三段代码的方式可以同时获取多个异常

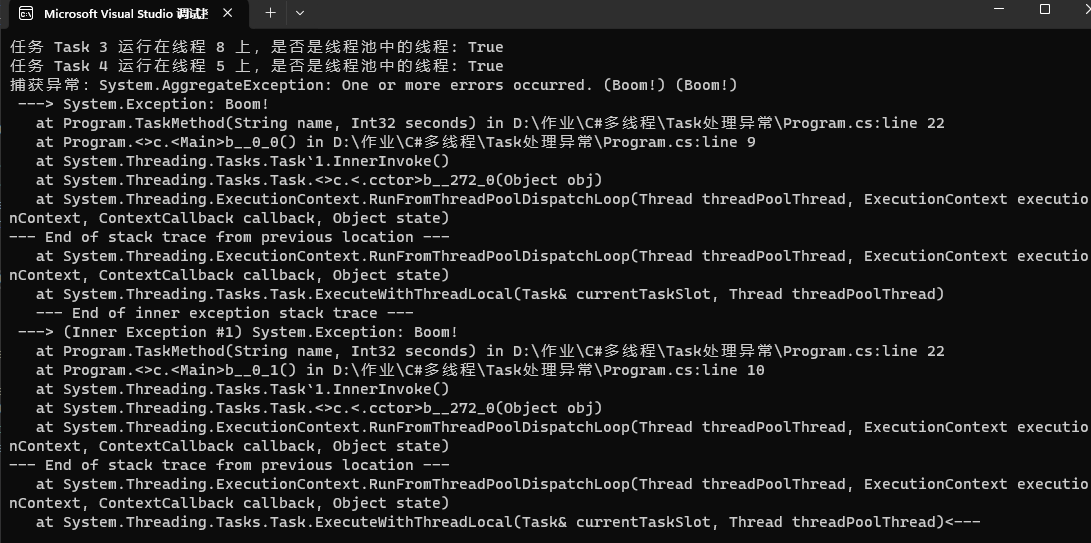
文本

描述已自动生成

文本, 信件

描述已自动生成

上面这段代码展示了两个任务都抛出异常的情况，只有在前面操作有异常时ContinueWith子任务才会执行，结果会打印出一个AggregateException异常，这个异常内部封装了两个异常。



Ps：WhenAll和WaitAll的区别，前者等待指定的所有任务完成并返回一个新的任务，后者则不会返回一个任务

并行运行任务（同时运行多个异步任务）：

使用WhenAny方法和WhenAll方法来并行执行多个任务，书上的例子简单易懂。使用WhenAll方法在指定的并行的任务都完成时返回一个新的任务，通过这个新的任务可以使用ContinueWith完成同步；使用WhenAny方法，在指定的并行的任务中的任意一个完成时返回一个新的任务，这个任务可以有以下作用，比如例子中的从任务列表中移除该任务，也可以用这个任务来记录是否超时，该任务先完成时取消掉剩下的未完成的任务，下面是书上的例子：

文本, 信件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

使用TaskScheduler配置任务的运行：

TaskScheduler 的一些主要作用：

1.任务的调度： TaskScheduler 允许你指定任务在何时执行。它提供了一个抽象的接口，让你能够自定义任务的执行策略。例如，你可以将任务调度到线程池线程、UI 线程、特定的同步上下文中等。

2.并行任务执行： TaskScheduler 用于支持并行任务的执行。当你有多个任务时，可以使用合适的任务调度器来确保这些任务能够并行执行，提高程序性能。

3.同步上下文： 在 UI 编程中，需要确保任务在正确的同步上下文中执行，以避免 UI 操作中的线程安全问题。TaskScheduler 允许你为任务指定合适的同步上下文。

4.任务取消： TaskScheduler 提供了支持任务取消的功能。通过 CancellationToken 和 TaskCreationOptions，你可以在任务执行时实现取消操作。

5.定制任务执行策略： 如果默认的任务调度策略不符合你的需求，你可以通过继承 TaskScheduler 并实现自定义的调度逻辑来创建自定义任务调度器。

在 .NET 中，有几个内置的任务调度器，其中一些是：

TaskScheduler.Default： 使用线程池线程执行任务。

TaskScheduler.Current： 获取当前线程的任务调度器。

TaskScheduler.FromCurrentSynchronizationContext： 在 UI 线程中执行任务的调度器。

TaskSechduler类：

表格

低可信度描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

文本

中度可信度描述已自动生成

文档：[TaskScheduler 类 (System.Threading.Tasks) | Microsoft Learn](https://learn.microsoft.com/zh-cn/dotnet/api/system.threading.tasks.taskscheduler?view=net-8.0)

Ps：TaskScheduler的所有线程都是安全的

[【C#】异步编程自定义任务调度器TaskScheduler介绍\_c# taskscheduler-CSDN博客](https://blog.csdn.net/m0_38045338/article/details/109225049)

[C# TaskScheduler-CSDN博客](https://blog.csdn.net/sudazf/article/details/107864243)

[精：C#TaskScheduler任务调度器的原理 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/632765795)