# Task-based Asynchronous Pattern (TAP)

**.NET Framework (current version)**

基于任务的异步模式 (TAP) 是基于 [System.Threading.Tasks](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks) 命名空间中的 [System.Threading.Tasks.Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task) 和[System.Threading.Tasks.Task<TResult>](https://msdn.microsoft.com/zh-cn/library/dd321424) 类型，这些类型用于表示任意异步操作。 TAP 是用于新开发的建议的异步设计模式。

## 命名、参数和返回类型

基于任务的异步模式 (TAP)使用单个方法表示异步操作的开始和完成。 这与异步编程模型（APM 或 **IAsyncResult**）模式相反（该模式要求 Begin 和 End 方法），并与基于事件的异步模式 (EAP) 相反（该模式要求具有 Async 后缀的方法，还要求一个或多个事件、事件处理程序委托类型和 **EventArg** 派生类型）。 TAP 中的异步方法在操作名称后包含 Async 后缀；例如，get 操作的 GetAsync。 如果你正在将 TAP 方法添加到已包含具有 Async 后缀的方法名称的类中，请改用后缀 TaskAsync。 例如，如果类具有 GetAsync 方法，请使用名称 GetTaskAsync。

TAP 方法返回 [System.Threading.Tasks.Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task) 或 [System.Threading.Tasks.Task<TResult>](https://msdn.microsoft.com/zh-cn/library/dd321424)，取决于相应的同步方法是返回 void 还是类型 **TResult**。

TAP 方法的参数应与其同步副本的参数匹配，并应以相同顺序提供。 但是，out 和 ref 参数不受此规则的限制，并应完全避免。 应该将通过 out 或 ref 参数返回的所有数据改为作为由 [Task<TResult>](https://msdn.microsoft.com/zh-cn/library/dd321424) 返回的 **TResult** 的一部分返回，且应使用元组或自定义数据结构来容纳多个值。 专用于创建、操作或组合任务的方法（其中，方法的名称或该方法所属的类型的名称已明确指示方法的异步用途）不需要遵循此命名模式；此类方法通常称为组合器。 组合器的示例包括 [WhenAll](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task.whenall) 和 [WhenAny](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task.whenany)，并在[Consuming the Task-based Asynchronous Pattern](https://msdn.microsoft.com/zh-cn/library/hh873173)一文的[“使用内置的基于任务的组合器”](https://msdn.microsoft.com/zh-cn/library/hh873173#combinators)部分进行了论述。

有关 TAP 语法与旧式异步编程模式（如异步编程模型 (APM) 和基于事件的异步模式 (EAP)）中使用的语法之间的区别的示例，请参阅[Asynchronous Programming Patterns](https://msdn.microsoft.com/zh-cn/library/jj152938)。

## 初始化异步操作

基于 TAP 的异步方法可以同步完成少量工作，如在返回结果任务之前，验证参数和启动异步操作。 应将同步工作保持最小，以便异步方法可以快速返回。 快速返回的原因包括以下几种：

* 可以从用户界面 (UI) 线程调用异步方法，因此，所有长期运行的同步工作可能会降低应用程序的响应能力。
* 可以同时启动多个异步方法。 因此，在异步方法的同步部分中的任何长时间运行的工作都可以延迟其他异步操作的启动，从而减少并发的优点。

在某些情况下，完成操作所需的工作量要比异步启动操作所需的工作量少。 读取流时，按照在内存中已缓冲好的数据来满足该读取，这就是此类情形的一个示例。 在这样的情况下，操作可能会同步完成，同时返回已完成的任务。

## 异常

异步方法应引发仅将引发异步方法调用的异常，以响应用法错误。 用法错误决不应该出现在成品代码中。 例如，如果将 null 引用（在 Visual Basic 中为 **Nothing**）作为某个方法的参数传递导致了错误状态（通常由 [ArgumentNullException](https://msdn.microsoft.com/zh-cn/library/system.argumentnullexception) 异常表示），则可以修改调用代码以确保绝对不传递 null 引用。 对于所有其他错误，在运行异步方法时发生的异常应分配给返回的任务，即使该异步方法碰巧在任务返回前同步完成。 通常，任务最多包含一个异常。 但是，如果任务表示多个操作（例如，[WhenAll](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task.whenall)），则多个异常可能与单个任务关联。

## 目标环境

在实现 TAP 方法时，你可以确定异步执行发生的位置。 你可选择在线程池上执行工作负荷，可选择使用异步 I/O 实现它（不必绑定到大部分操作执行的线程）或可选择在特定线程（如 UI 线程）上运行它或使用任何数目的潜在上下文。 TAP 方法甚至可能没有要执行的操作，可能仅返回表示系统中其他位置的匹配项的 [Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task)（例如，表示到达某种排队数据结构的数据的任务）。TAP 方法的调用方可能会通过异步等待生成结果来阻止等待 TAP 方法完成，也可能在异步操作完成时运行其他（延续）节点。 延续代码的创建者可以控制该代码的执行位置。 你可以通过 [Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task) 类上的方法（例如，[ContinueWith](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task.continuewith)）显式创建延续代码，也可以使用基于延续（例如，C# 中的 **await**、Visual Basic 中的 **Await** 和 F# 中的 **AwaitValue**）构建的语言支持隐式创建延续代码。

## 任务状态

[Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task) 类提供了异步操作的生命周期，且该周期由 [TaskStatus](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 枚举表示。 为了支持派生自 [Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task) 和 [Task<TResult>](https://msdn.microsoft.com/zh-cn/library/dd321424) 的类型的个别案例，并支持调度时分离构造，[Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task) 类公开了 [Start](https://msdn.microsoft.com/zh-cn/library/dd270682) 方法。 公共 [Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task) 构造函数创建的任务称为冷任务，因为它们在非安排的 [Created](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态下开始其生命周期，并且仅在对这些实例调用 [Start](https://msdn.microsoft.com/zh-cn/library/dd270682) 时才进行安排。 所有其他任务在热状态下开始其生命周期，这意味着它们表示的异步操作已启动，并且其任务状态是 [TaskStatus.Created](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 以外的枚举值。 必须激活从 TAP 方法返回的所有任务。 如果 TAP 方法在内部使用任务的构造函数来实例化要返回的任务，则该 TAP 方法必须在返回之前对 [Start](https://msdn.microsoft.com/zh-cn/library/dd270682) 对象调用 [Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task)。 TAP 方法的使用者可以安全地假设返回的任务处于活动状态且不应尝试对从 TAP 方法返回的任何 [Start](https://msdn.microsoft.com/zh-cn/library/dd270682) 调用[Task](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task)。 对活动的任务调用 [Start](https://msdn.microsoft.com/zh-cn/library/dd270682) 将引发 [InvalidOperationException](https://msdn.microsoft.com/zh-cn/library/system.invalidoperationexception) 异常。

## 取消（可选）

在 TAP 中，取消是异步方法实现者和异步方法使用者的选项。 如果操作允许取消，则会公开接受取消标记（[CancellationToken](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken) 实例）的异步方法的重载。 按照约定，该参数命名为 cancellationToken。

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public Task ReadAsync(byte [] buffer, int offset, int count,

CancellationToken cancellationToken)

该异步操作监视取消请求的此标记。 如果它收到取消请求，则可以选择接受该请求并取消操作。 如果取消请求导致过早地结束工作，则 TAP 方法返回一个在 [Canceled](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态下结束的任务；没有可用结果且不引发异常。 [Canceled](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态被视为任务的最终（完成）状态，以及 [Faulted](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 和 [RanToCompletion](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态。 因此，如果一个任务处于 [Canceled](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态，则其 [IsCompleted](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task.iscompleted) 属性将返回 **true**。 在 [Canceled](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态下完成任务时，将计划或执行向任务注册的任何延续，除非延续选项（如[NotOnCanceled](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskcontinuationoptions)）特定于取消延续。 任何通过使用语言功能异步等待已取消的任务的代码都将继续运行，但不接收[OperationCanceledException](https://msdn.microsoft.com/zh-cn/library/system.operationcanceledexception) 或其中派生的异常。 通过诸如 [Wait](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task.wait) 的方法同步阻止的代码等待任务，并且 [WaitAll](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.task.waitall) 将继续运行但出现异常。

如果取消标记请求在接受调用标记的 TAP 方法之前取消，TAP 方法应返回 [Canceled](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 任务。 但是，如果在运行异步操作时请求取消，则异步操作不需要接受该取消请求。 仅当该操作如取消请求的结果那样结束时，返回的任务才应以 [Canceled](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态结束。 如果已请求取消，但仍然生成了结果或异常，则任务应在 [RanToCompletion](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 或 [Faulted](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskstatus) 状态下结束。 对于希望首先进行取消的开发人员使用的异步方法，你不需要提供不接受取消标记的重载。 对于无法取消的方法，不提供接受取消标记的重载；这有助于向调用方指示目标方法是否真正可取消。 不要求取消的使用者代码可以调用接受 [CancellationToken](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken) 的方法，并提供 [None](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken.none) 作为参数值。 [None](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken.none) 在功能上等效于默认 [CancellationToken](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken)。

## 进度报告（可选）

某些异步操作受益于提供进度通知；这些通常用于使用有关该异步操作的进度的信息更新用户界面。 在 TAP 中，通过[IProgress<T>](https://msdn.microsoft.com/zh-cn/library/hh138298) 接口处理进度，此接口作为通常名为 progress 的参数传递给异步方法。 调用异步方法时提供进度接口有助于消除不正确使用导致的争用情况（也就是说，操作启动后未正确注册的事件处理程序可能缺少更新）。 更重要的是，根据所使用的代码，进度接口将支持不同的进度实现。 例如，使用代码可能只关心最新的进度更新，可能需要缓冲所有更新，可能需要为各个更新调用操作，也可能需要控制是否需将该调用封送到特定线程。 通过使用接口的不同实现，所有这些选项都可以满足特定的使用者的需要。 与取消一样，仅在 API 支持进度通知时，TAP 实现才应提供 [IProgress<T>](https://msdn.microsoft.com/zh-cn/library/hh138298) 参数。 例如，如果本文前面所述的 ReadAsync 方法可以以到目前为止读取的字节数的形式报告中间进度，则进度回调可能为 [IProgress<T>](https://msdn.microsoft.com/zh-cn/library/hh138298) 接口：

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-2)

public Task ReadAsync(byte[] buffer, int offset, int count,

IProgress<long> progress)

如果 FindFilesAsync 方法返回满足特定搜索模式的所有文件的列表，则进度回调可以对已完成工作的百分比和当前部分结果集进行估计。 可使用元组执行此操作：

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-3)

public Task<ReadOnlyCollection<FileInfo>> FindFilesAsync(

string pattern,

IProgress<Tuple<double,

ReadOnlyCollection<List<FileInfo>>>> progress)

也可以使用特定于 API 的数据类型执行此操作：

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-4)

public Task<ReadOnlyCollection<FileInfo>> FindFilesAsync(

string pattern,

IProgress<FindFilesProgressInfo> progress)

在后一种情况下，特殊数据类型应加上后缀 ProgressInfo。

如果 TAP 实现提供接受 progress 参数的重载，则必须允许该参数成为 **null**，在这种情况下，不会报告任何进度。 TAP 实现应该同步将进度报告到 [Progress<T>](https://msdn.microsoft.com/zh-cn/library/hh193692) 对象，使异步方法能够快速提供进度，并允许进度的使用者确定处理信息的最佳方式和位置。 例如，进度实例可以选择将回调封送，并引发有关捕获到的同步上下文的事件。

## IProgress<T> 实现

.NET Framework 4.5 提供单个 [IProgress<T>](https://msdn.microsoft.com/zh-cn/library/hh138298) 实现：[Progress<T>](https://msdn.microsoft.com/zh-cn/library/hh193692)。 [Progress<T>](https://msdn.microsoft.com/zh-cn/library/hh193692) 类的声明方式如下：

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-5)

public class Progress<T> : IProgress<T>

{

public Progress();

public Progress(Action<T> handler);

protected virtual void OnReport(T value);

public event EventHandler<T> ProgressChanged;

}

[Progress<T>](https://msdn.microsoft.com/zh-cn/library/hh193692) 的实例公开 [ProgressChanged](https://msdn.microsoft.com/zh-cn/library/hh137516) 事件，此事件在异步操作每次报告进度更新时引发。 实例化 [ProgressChanged](https://msdn.microsoft.com/zh-cn/library/hh137516)实例后，会在捕获到的 [SynchronizationContext](https://msdn.microsoft.com/zh-cn/library/system.threading.synchronizationcontext) 对象上引发 [Progress<T>](https://msdn.microsoft.com/zh-cn/library/hh193692) 事件。 如果没有可用的同步上下文，则使用针对线程池的默认上下文。 可以向此事件注册处理程序。 为了方便起见，也可将单个处理程序提供给 [Progress<T>](https://msdn.microsoft.com/zh-cn/library/hh193692) 构造函数，并且行为与 [ProgressChanged](https://msdn.microsoft.com/zh-cn/library/hh137516) 事件的事件处理程序一样。 异步引发进度更新以避免延迟异步操作，同时执行事件处理程序。 另一个 [IProgress<T>](https://msdn.microsoft.com/zh-cn/library/hh138298) 实现可以选择应用不同的语义。

## 选择要提供的重载

如果 TAP 实现使用可选的 [CancellationToken](https://msdn.microsoft.com/zh-cn/library/system.threading.tasks.taskfactory.cancellationtoken) 和可选的 [IProgress<T>](https://msdn.microsoft.com/zh-cn/library/hh138298) 参数，则可能需要多达四次的重载：

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-6)

public Task MethodNameAsync(…);

public Task MethodNameAsync(…, CancellationToken cancellationToken);

public Task MethodNameAsync(…, IProgress<T> progress);

public Task MethodNameAsync(…,

CancellationToken cancellationToken, IProgress<T> progress);

但是，许多 TAP 实现没有提供取消或进度功能，因此它们需要一个方法：

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-7)

public Task MethodNameAsync(…);

如果 TAP 实现支持取消或进度但不同时支持二者，则 TAP 实现可能提供以下两种重载：

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-8)

public Task MethodNameAsync(…);

public Task MethodNameAsync(…, CancellationToken cancellationToken);

// … or …

public Task MethodNameAsync(…);

public Task MethodNameAsync(…, IProgress<T> progress);

如果 TAP 实现同时支持取消和进度，则会公开所有四个重载。 但是，它只提供以下两个:

C#

[**VB**](https://msdn.microsoft.com/library/hh873175?cs-save-lang=1&cs-lang=vb#code-snippet-9)

public Task MethodNameAsync(…);

public Task MethodNameAsync(…,

CancellationToken cancellationToken, IProgress<T> progress);

若要弥补缺少的两个中间组合，开发人员可以传递 cancellationToken 参数的 [None](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken.none) 或默认的 [CancellationToken](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken) 和progress 参数的 **null**。

如果需要 TAP 方法的每种用法支持取消或进度，则可以忽略不接受相关参数的重载。

如果决定公开多个重载以使取消或进度可选，则不支持取消或进度的重载的行为方式就像其已将取消的 [None](https://msdn.microsoft.com/zh-cn/library/system.threading.cancellationtoken.none) 或进度的 **null** 传递给确实支持它们的重载。

## 相关主题

|  |  |
| --- | --- |
| **标题** | **描述** |
| [Asynchronous Programming Patterns](https://msdn.microsoft.com/zh-cn/library/jj152938) | 介绍执行异步操作的三种模式：基于任务的异步模式 (TAP)、异步编程模型 (APM) 和基于事件的异步模式 (EAP)。 |
| [Implementing the Task-based Asynchronous Pattern](https://msdn.microsoft.com/zh-cn/library/hh873177) | 描述如何使用以下三种方式实现基于任务的异步模式 (TAP)：手动使用 Visual Studio 中的 C# 和 Visual Basic 编译器，或通过编译器和手动方法的组合。 |
| [Consuming the Task-based Asynchronous Pattern](https://msdn.microsoft.com/zh-cn/library/hh873173) | 描述你可以如何使用任务和回调实现等待，而无需阻止。 |
| [Interop with Other Asynchronous Patterns and Types](https://msdn.microsoft.com/zh-cn/library/hh873178) | 描述如何使用基于任务的异步模式 (TAP) 实现异步编程模型 (APM) 和基于事件的异步模式 (EAP)。 |