

此站点使用 Cookie 进行分析、显示个性化内容和广告。继续浏览此站点，即表示您同意使用。 [了解详情](#)



Microsoft Logo



Gray Pipe

[Developer Network](#) [Developer Network](#) [Developer](#)

[MSDN 订阅](#)

[获取工具](#)

- [下载](#)
- [计划](#)
- [社区](#)
- [文档](#)



search clear

[Parallel Processing and Concurrency](#) [Asynchronous Programming Patterns](#) [Task-based Asynchronous Pattern \(TAP\)](#)

[Task-based Asynchronous Pattern \(TAP\)](#) [Interop with Other Asynchronous Patterns and Types](#)

[打印](#) [导出 \(0\)](#)

[Interop with Other Asynchronous Patterns and Types](#)

[打印](#) [导出 \(0\)](#)

[Implementing the Task-based Asynchronous Pattern](#)

[Consuming the Task-based Asynchronous Pattern](#)

[Interop with Other Asynchronous Patterns and Types](#)

[TOC](#)

要查看英语原文，请勾选“英语”复选框。也可将鼠标指针移到文本上，在弹出窗口中显示英语原文。	翻译英语
---	------

Interop with Other Asynchronous Patterns and Types

.NET Framework (current version)

.NET Framework 1.0 引进了 [IAsyncResult](#) 模式，也称为 [Asynchronous Programming Model \(APM\)](#) 或 **Begin/End** 模式。 .NET Framework 2.0 增加了 [Event-based Asynchronous Pattern \(EAP\)](#)。 从 .NET Framework 4 开始，[Task-based Asynchronous Pattern \(TAP\)](#) 取代了 APM 和 EAP，但能够轻松构建从早期模式中迁移的例程。

本主题内容：

- [任务和 APM](#) ([从 APM 到 TAP](#) 或 [从 TAP 到 APM](#))
- [任务和 EAP](#)
- [任务和等待句柄](#) ([从等待句柄到 TAP](#) 或 [从 TAP 到等待句柄](#))

任务和异步编程模型 (APM)

从 APM 到 TAP

因为 [Asynchronous Programming Model \(APM\)](#) 模式的结构非常合理，而且能够轻松生成包装，将 APM 实现公开为 TAP 实现。实际上，自 .NET Framework 4 之后的 .NET Framework 就包含采用 [FromAsync](#) 方法重载形式的帮助器例程来实现这种转换。

请考虑 [Stream](#) 类及其 [BeginRead](#) 和 [EndRead](#) 方法，它们代表与同步 [Read](#) 方法对应的 APM：

C#

[VB](#)

```
public int Read(byte[] buffer, int offset, int count)
```

C#

[VB](#)

```
public IAsyncResult BeginRead(byte[] buffer, int offset,  
                               int count, AsyncCallback callback,  
                               object state)
```

C#

VB

```
public int EndRead(IAsyncResult asyncResult)
```

可以使用 [TaskFactory<TResult>.FromAsync](#) 方法来实现此操作的 TAP 包装，如下所示：

C#

VB

```
public static Task<int> ReadAsync(this Stream stream,
                                   byte[] buffer, int offset,
                                   int count)
{
    if (stream == null)
        throw new ArgumentNullException("stream");

    return Task<int>.Factory.FromAsync(stream.BeginRead,
                                       stream.EndRead, buffer,
                                       offset, count, null);
}
```

此实现类似于以下内容：

C#

VB

```
public static Task<int> ReadAsync(this Stream stream,
                                   byte [] buffer, int offset,
                                   int count)
{
    if (stream == null)
        throw new ArgumentNullException("stream");

    var tcs = new TaskCompletionSource<int>();
    stream.BeginRead(buffer, offset, count, iar =>
    {
        try {
            tcs.TrySetResult(stream.EndRead(iar));
        }
        catch (OperationCanceledException) {
            tcs.TrySetCanceled();
        }
    });
}
```

```

    }
    catch(Exception exc) {
        tcs.TrySetException(exc);
    }
}, null);
return tcs.Task;
}

```

从 TAP 到 APM

如果现有的基础结构需要 APM 模式，则还需要采用 TAP 实现并在需要 APM 实现的地方使用它。由于任务可以组合，并且 [Task](#) 类实现 [IAsyncResult](#)，您可以使用一个简单的 helper 函数执行此操作。以下代码使用 [Task<TResult>](#) 类的扩展，但可以对非泛型任务使用几乎相同的函数。

C#

[VB](#)

```

public static IAsyncResult AsApm<T>(this Task<T> task,
                                     AsyncCallback callback,
                                     object state)
{
    if (task == null)
        throw new ArgumentNullException("task");

    var tcs = new TaskCompletionSource<T>(state);
    task.ContinueWith(t =>
    {
        if (t.IsFaulted)
            tcs.TrySetException(t.Exception.InnerExceptions);
        else if (t.IsCanceled)
            tcs.TrySetCanceled();
        else
            tcs.TrySetResult(t.Result);

        if (callback != null)
            callback(tcs.Task);
    }, TaskScheduler.Default);
}

```

```
    return tcs.Task;
}
```

现在，请考虑具有以下 TAP 实现的用例：

C#

[VB](#)

```
public static Task<String> DownloadStringAsync(Uri url)
```

并且想要提供此 APM 实现：

C#

[VB](#)

```
public IAsyncResult BeginDownloadString(Uri url,
                                         AsyncCallback callback,
                                         object state)
```

C#

[VB](#)

```
public string EndDownloadString(IAsyncResult asyncResult)
```

以下示例演示了一种向 APM 迁移的方法：

C#

[VB](#)

```
public IAsyncResult BeginDownloadString(Uri url,
                                         AsyncCallback callback,
                                         object state)
{
    return DownloadStringAsync(url).AsApm(callback, state);
}

public string EndDownloadString(IAsyncResult asyncResult)
{
    return ((Task<string>)asyncResult).Result;
}
```

任务和基于事件的异步模式 (EAP)

包装 [Event-based Asynchronous Pattern \(EAP\)](#) 实现比包装 APM 模式更为复杂，因为与 APM 模式相比，EAP 模式的变体更多，结构更少。为了演示，以下代码包装了 DownloadStringAsync 方法。DownloadStringAsync 接受 URI，在下载时引发

DownloadProgressChanged 事件，以报告进度的多个统计信息，并在完成时引发 DownloadStringCompleted 事件。最终在指定 URI 中返回一个字符串，其中包含页面内容。

C#

[VB](#)

```
public static Task<string> DownloadStringAsync(Uri url)
{
    var tcs = new TaskCompletionSource<string>();
    var wc = new WebClient();
    wc.DownloadStringCompleted += (s,e) =>
    {
        if (e.Error != null)
            tcs.TrySetException(e.Error);
        else if (e.Cancelled)
            tcs.TrySetCanceled();
        else
            tcs.TrySetResult(e.Result);
    };
    wc.DownloadStringAsync(url);
    return tcs.Task;
}
```

任务和等待句柄

从等待句柄到 TAP

虽然等待句柄不能实现异步模式，但高级开发人员可以在设置等待句柄时使用 [WaitHandle](#) 类和 [ThreadPool.RegisterWaitForSingleObject](#) 方法实现异步通知。可以包装 [RegisterWaitForSingleObject](#) 方法以在等待句柄中启用针对任何同步等待的基于任务的替代方法：

C#

[VB](#)

```
public static Task WaitOneAsync(this WaitHandle waitHandle)
{
    if (waitHandle == null)
        throw new ArgumentNullException("waitHandle");
}
```

```

var tcs = new TaskCompletionSource<bool>();
var rwh = ThreadPool.RegisterWaitForSingleObject(waitHandle,
    delegate { tcs.TrySetResult(true); }, null, -1, true);
var t = tcs.Task;
t.ContinueWith( (antecedent) => rwh.Unregister(null));
return t;
}

```

使用此方法，可以在异步方法中使用现有 [WaitHandle](#) 实现。例如，如果想要限制在任何特定时间执行的异步操作的数目，则可以利用一个信号量（[System.Threading.SemaphoreSlim](#) 对象）。可以将并发运行的操作数目限制到 N ，方法为：初始化到 N 的信号量的数目、在想要执行操作时等待信号量，并在完成操作时释放信号量：

C#

[VB](#)

```
static int N = 3;
```

```
static SemaphoreSlim m_throttle = new SemaphoreSlim(N, N);
```

```

static async Task DoOperation()
{
    await m_throttle.WaitAsync();
    // do work
    m_throttle.Release();
}

```

还可以构建不依赖等待句柄就完全可以处理任务的异步信号量。若要执行此操作，可以使用 [Consuming the Task-based Asynchronous Pattern](#) 中所述的用于在 [Task](#) 上构建数据结构的技术。

从 TAP 到等待句柄

正如前面所述，[Task](#) 类实现 [IAsyncResult](#)，且该实现公开 [IAsyncResult.AsyncWaitHandle](#) 属性，该属性会返回在 [Task](#) 完成时设置的等待句柄。可以获得 [Task](#) 的 [WaitHandle](#)，如下所示：

C#

[VB](#)

WaitHandle wh = ((IAsyncResult)task).AsyncWaitHandle;

请参阅

[Task-based Asynchronous Pattern \(TAP\)](#)

[Implementing the Task-based Asynchronous Pattern](#)

[Consuming the Task-based Asynchronous Pattern](#)

显示: 继承 保护

[打印共享](#)

本文内容

- [任务和异步编程模型 \(APM\)](#)
- [任务和基于事件的异步模式 \(EAP\)](#)
- [任务和等待句柄](#)
- [请参阅](#)

此页面有帮助吗？

开发人员中心

- [Windows](#)
- [Office](#)
- [Visual Studio](#)
- [Microsoft Azure](#)
- [更多...](#)

学习资源

- [Microsoft 虚拟学院](#)
- [第 9 频道](#)
- [MSDN 杂志](#)

社区

- [论坛](#)
- [博客](#)
- [Codeplex](#)

支持

- [自助支持](#)

计划

- [BizSpark \(针对新创企业 \)](#)
- [Microsoft Imagine \(for students\)](#)

[中国 \(简体中文\)](#)

- [新闻稿](#)
- [隐私& Cookie](#)
- [使用条款](#)
- [商标](#)

 Microsoft 

logo

© 2017 Microsoft

© 2017 Microsoft

The .NET Framework 1.0 introduced the IAsyncResult pattern, otherwise known as the Asynchronous Programming Model (APM), or the **Begin/End** pattern.

[建议更好的翻译](#)