```
    task.ContinueWith (ant =>
    {
      if (ant.IsCanceled)
        killJoy.TrySetCanceled();
      else if (ant.IsFaulted)
        killJoy.TrySetException (ant.Exception.InnerException);
    });
  return await await Task.WhenAny (killJoy.Task, Task.WhenAll (tasks))
                         .ConfigureAwait (false);
}
```

本例首先创建了 TaskCompletionSource 对象，其唯一作用就是在任何一个任务出错时终止最终的任务。因此我们永远不会调用它的 SetResult 方法，只会调用它的 TrySetCanceled 和 TrySetException 方法。本例中，由于不需要访问任务的结果，也不需要在此时回弹到 UI 线程中，因此 ContinueWith 比 GetAwaiter().OnCompleted 更合适。

14.6.5 异步锁

22.4.1 节将介绍如何使用 SemaphoreSlim 来锁定或对并发的异步操作进行限制。

14.7 旧有的异步编程模式

在任务和异步函数出现之前，.NET 使用其他方式实现异步编程。由于基于任务的异步处理已经成为主流模式，这些模式现在已经很少使用了。

14.7.1 异步编程模型

异步编程模型（Asynchronous Programming Model，APM）是最古老的编程模式。它使用一对以 Begin 和 End 开头的方法，以及 IAsyncResult 接口实现异步执行。以下示例将调用 System.IO 命名空间中的 Stream 类的 Read 方法。其同步版本为：

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

可以推断，基于任务的异步版本的声明应当是：

```
public Task<int> ReadAsync (byte[] buffer, int offset, int size);
```

而 APM 版本的声明如下：

```
public IAsyncResult BeginRead (byte[] buffer, int offset, int size,
                               AsyncCallback callback, object state);
public int EndRead (IAsyncResult asyncResult);
```

调用 Begin* 方法将启动异步操作，并返回 IAsyncResult 对象，该对象是异步操作的令牌。当操作完成或出错时就会触发 AsyncCallback 委托：