（这一整章都很模糊，没看懂，还需详细读一遍）

Windows如何执行IO同步操作的：

P643-646

C#的异步函数

一些文字和图片的手机截图

描述已自动生成

报纸上的文字

描述已自动生成

异步函数的限制：

报纸上的文字和图片的手机截图

描述已自动生成

如何将异步函数转换成状态机：

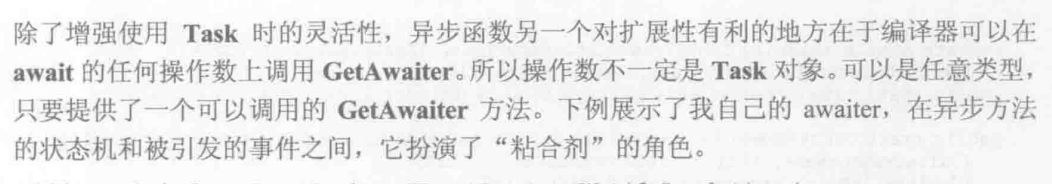
（没看太懂）

提炼一下：实际上使用了任务，功能相当于把await后面的代码ContinueWith。

异步函数的扩展性：

文本

描述已自动生成



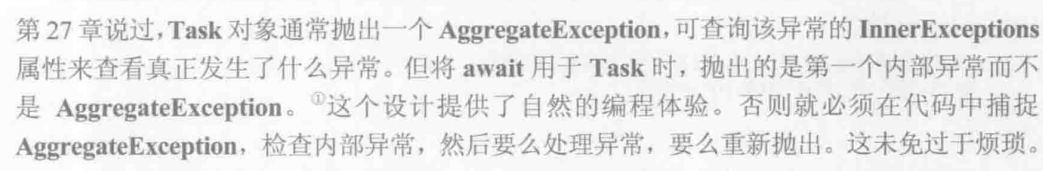
异步函数和事件处理程序：

异步函数的返回类型一般是Task或者Task<某类型>，它们代表函数的状态机完成。（不过也可以像我们前面的例子一样返回void）

FCL的异步函数：



异步函数和异常处理：



C#的异步lambda表达式：

图片包含 文本

描述已自动生成

文本, 信件

描述已自动生成

应用程序与线程处理模型：

.NET支持几种不同的应用程序模型，而每种模型可能引入了它自己的线程处理模型。

控制台应用程序和Windows服务（实际上也是控制台应用程序，只是看不到控制台）没有引入任何线程处理模型。

GUI应用程序引入了一个线程处理模型。在此模型中，UI元素只能由创建它的线程更新。

在GUI线程中，经常都需要生成一个异步操作，使GUI线程不至于阻塞并停止响应用户输入。但当异步操作完成时，是由一个线程池线程完成Task对象并恢复状态机。但是当这个线程池线程一旦更新UI元素就会抛出异常，所以线程池线程只能呢个以某种方式告诉GUI线程更新UI元素。然而FCL定义了一个SynchronizationContext类（同步上下文类）来解决这个问题，简单来说此类的对象将应用程序模型和线程处理模型连接起来。作为开发人员通常不需要了解这个类，等待一个Task时会获取调用线程的SynchronizationContext对象，线程池完成Task后，会使用该SynchronizationContext对象，确保为应用程序模型使用正确的线程处理模型。

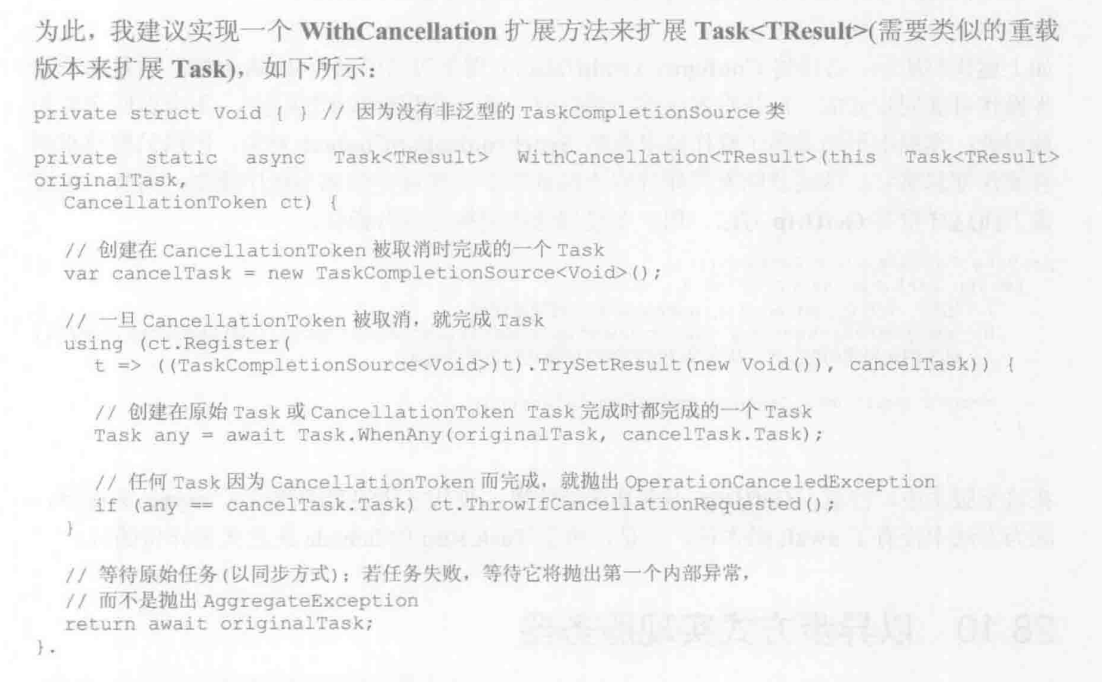
所以当GUI线程等待一个Task时，await操作符后面的代码保证在GUI线程上执行，使代码能正确执行。

Task提供了一个ConfigureAwait方法，向其传递true就相当于没有调用方法，传递false则await操作符就不查询调用线程的SynchronizationContext对象。当线程池结束Task时会直接完成，await操作符后面的代码通过线程池线程执行。

取消I/O操作：

Windows没有提供，需要自己实现

例子：



文本

描述已自动生成

文本

中度可信度描述已自动生成

I/O请求优先级

即在线程发出一个I/O请求时指定优先级

作者写书的时候FCL暂不支持，只能使用P/Invoke，使用Win32的实现（下面的代码可能是正确的）[.net - How can I/O priority of a process be increased? - Stack Overflow](https://stackoverflow.com/questions/301290/how-can-i-o-priority-of-a-process-be-increased)

