C#执行异步操作的几种方式总结比较

0x01 引言

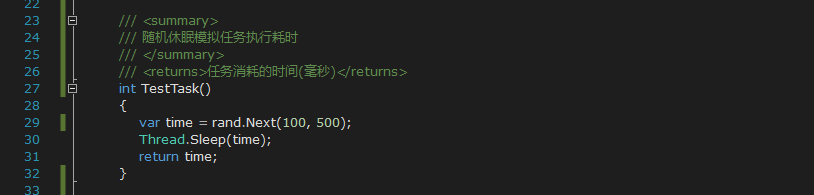
之前写程序的时候在遇到一些比较花时间的操作例如HTTP请求时，总是会new一个Thread处理。对于一些XxxxxAsync()方法也没去了解过，倒也没遇到什么大问题。最近因为需求要求用DevExpress写界面，跑起来后发现比Native控件效率差好多。这才想到之前看到的“金科玉律”:不要在UI线程上执行界面无关的操作，因此集中看了下C#的异步操作，分享一下自己的比较和总结。

0x01 测试方法

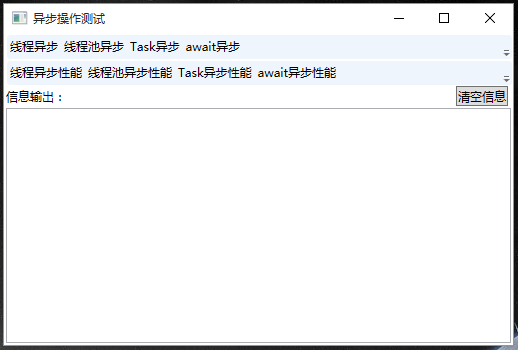
IDE：VS2015 Community

.NET版本：4.5

使用函数随机休眠100到500毫秒来模拟耗时任务，并返回任务花费的时间，在UI线程上调用这个方法会造成阻塞，导致UI假死，因此需要通过异步方式执行这个任务，并在信息输出区域显示花费的时间。

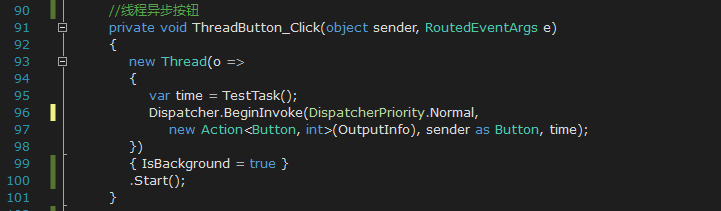


主界面中通过各种不同按钮测试不同的异步操作



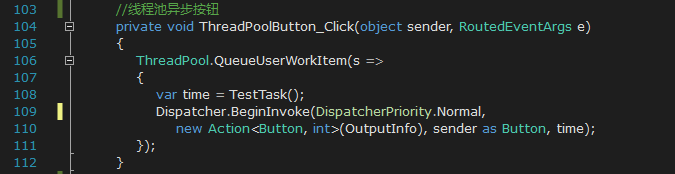
0x03 使用Thread进行异步操作

调用方法如下所示，需要注意的就是IsBackground默认为false，也就是该线程对调用它的线程不产生依赖，当调用线程退出时该线程也不会结束。因此需要将IsBackground设置为true以指明该线程是后台线程，这样当主线程退出时该线程也会结束。另外跨线程操作UI还是要借助Dispatcher.BeginInvoke()，如果又需要阻塞UI线程在经过用户某些操作后再继续需要使用Dispatcher.Invoke()。



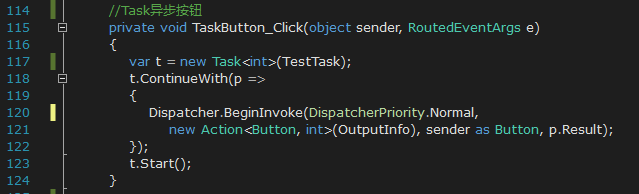
0x04 使用ThreadPool进行异步操作

为什么会出现ThreadPool（线程池）的概念呢，主要就是因为线程的创建是开销比较大的行为。为了达到较好的交互体验，开发中可能会大量使用异步操作，特别是需要频繁进行大量的短时间的异步操作时，频繁创建和销毁线程会使程序效率下降。线程池就是线程复用的一种方式，线程池中存在一些线程，当需要新建线程执行操作时就从线程池中取出一个已经存在的空闲线程使用，如果此时没有空闲线程，且线程池中的线程数未达到线程池上限，则新建一个线程，使用完成后再放回到线程池中。这样可以极大程度上省去线程创建的开销。线程池中线程的最小和最大数都可以指定，不过多数情况下无需指定，CLR有一套管理线程池的策略。ThreadPool的使用非常简单，代码如下所示。跨线程操作UI仍需借助Dispatcher。



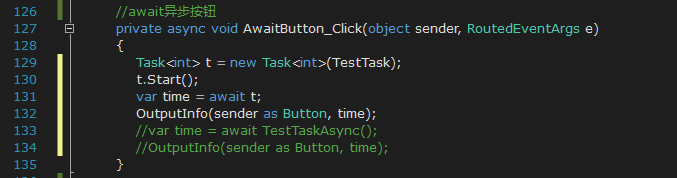
0x05 使用Task进行异步操作

Task进行异步操作时也是从线程池中取线程进行操作，不过支持的操作更加丰富一些。而且Task<T>可以支持异步返回值，通过Task的ContinueWith()可以在Task执行结束后将返回值传入以进行操作，但在ContinueWith中跨线程操作UI仍需借助Dispatcher。另外Task也可以直接使用静态方法Task.Run<T>()执行异步操作。

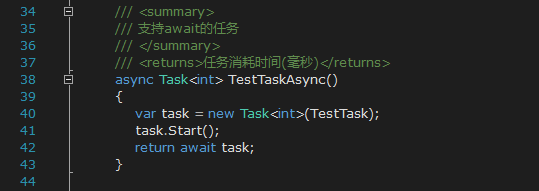


0x06 使用async/await进行异步操作

这个是C#5中的新特性，当遇到await时，会从线程池中取出一个线程异步执行await等待的操作，方法立即返回。等异步操作结束后返回await所在的地方接着往后执行。await需要等待async Task<T>类型的函数。详细的使用方法可参考相关资料，测试代码如下所示。异步结束后的会返回到调用线程，所以修改UI不需要Dispatcher。



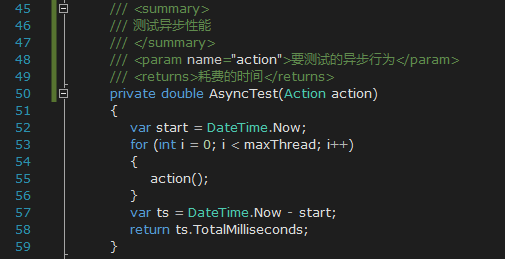
此外也可以把TestTask包装成async方法，这样就可以使用上图中注释掉的两行代码进行处理。包装后的异步方法如下所示：



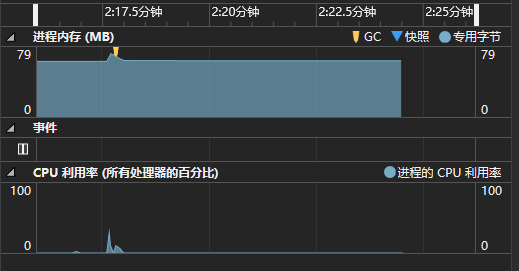
async/await也是从线程池中取线程，可实现线程复用，而且代码简洁容易阅读，异步操作返回后会自动返回调用线程，是执行异步操作的首选方式。而且虽然是C#5的新特性，但C#4可以通过下载升级包来支持async/await。

0x07 关于效率

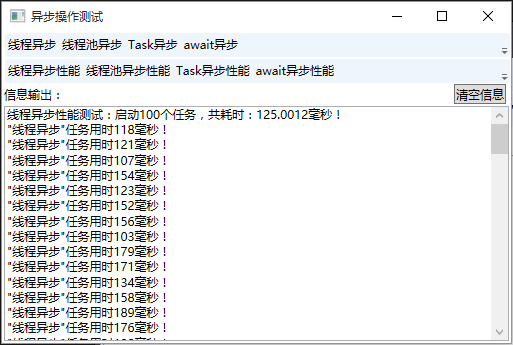
以上尝试的方法除了直接使用Thread之外，其他几种都是直接或间接使用线程池来获取线程的。从理论上来分析，创建线程时要给线程分配栈空间，线程销毁时需要回收内存，创建线程也会增加CPU的工作。因此可以连续创建线程并记录消耗的时间来测试性能。测试代码如下所示：



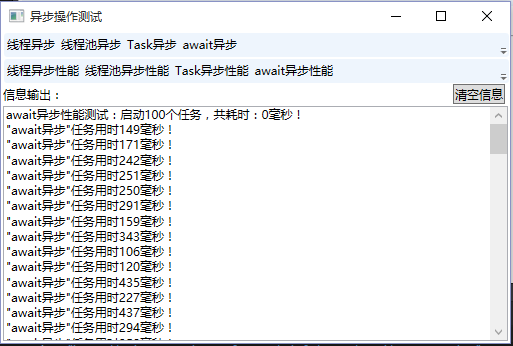
当测试Thread时每次测试在连续创建线程时内存和CPU都会有个小突起，不过在线程结束后很快就会降下去，在我的电脑上连续创建100个线程大概花费120-130毫秒。如图所示：



测试结果：



使用基于线程池的方法创建线程时，有时第一次会稍慢一些，应该是线程池内线程不足，时间开销在0-15毫秒，第一次创建内存也会上升。后面再测试时时间开销为0毫秒，内存表现也很平稳，CPU开销分布比较平均。测试结果如图所示：



0x08 结论

在执行异步操作时应使用基于线程池的操作，从代码的简洁程度和可读性上优先使用async/await方式。对于较老的.NET版本可以使用Task或ThreadPool。符合以下情况的可以使用Thread：

1、线程创建后需要持续工作到主线程退出的。这种情况下就算使用线程池线程也不会归还，实现不了复用，可以使用Thread。

2、线程在主线程退出后仍需要执行的，这种情况使用线程池线程无法满足需求，需要使用Thread并制定IsBackground为false（默认）。