C#中实现并发的几种方法的性能测试

0x00 起因

去年写的一个程序因为需要在局域网发送消息支持一些命令和简单数据的传输，所以写了一个C/S的通信模块。当时的做法很简单，服务端等待链接，有用户接入后开启一个线程，在线程中运行一个while循环接收数据，接收到数据就处理。用户退出（收到QUIT命令）后线程结束。程序一直运行正常（当然还要处理“TCP粘包”、消息格式封装等问题，在此不作讨论），不过随着使用的人越来越多，而且考虑到线程开销比较大，如果有100个用户链接那么服务端就要多创建100个线程，500个用户就是500个线程，确实太夸张了。由于TCP通信并不是每时每刻都在进行着的，因此可以把所有客户端连接存储到一个列表中，通过轮询的方式依次开启一个线程进行数据接收，接收完毕后释放线程，这样可以充分利用线程池，避免大量线程消耗内存和CPU。

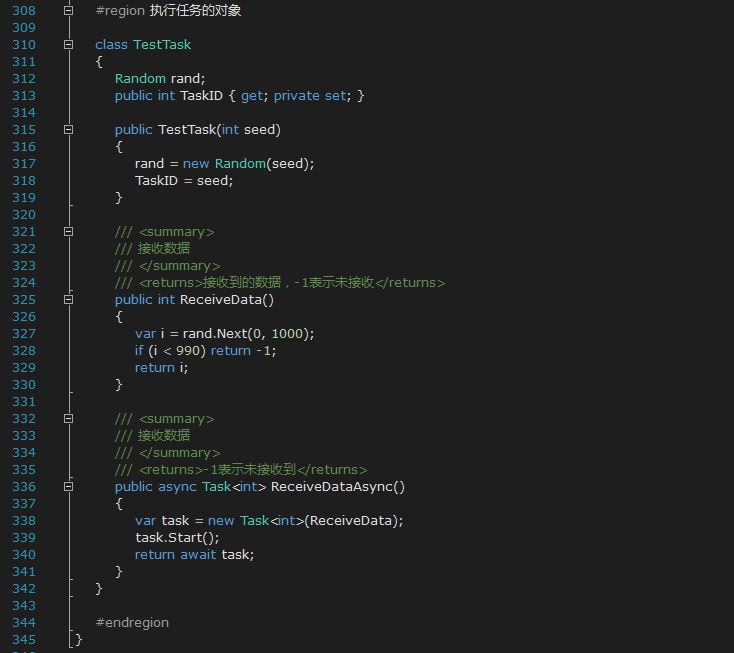
轮询的方式通过线程池实现了线程的复用，但轮询的方式在单位时间内的处理次数会不会比保持线程的方式少很多呢，本测试也将解决这个疑问。

0x01 实验方法

IDE：VS2015

.Net Framework 4.5

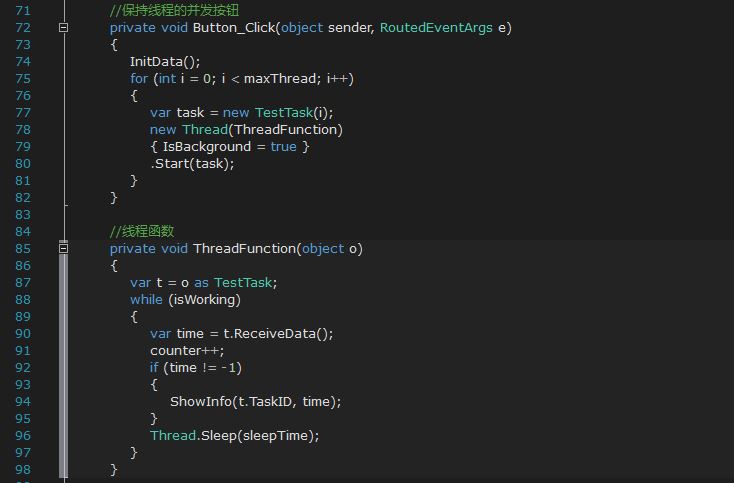
接收数据的对象如下所示



通过ReceiveData方法接收数据，每次接收只有1%的可能性收到数据，通过创建N个对象接收数据来模拟一个TCP服务端处理N个连接的情况。毕竟TCP通信不是随时进行的，当然这个百分比可以调整。程序输出的内容包括每秒执行了多少次接收操作，接收到数据的线程编号和接收到的内容等。

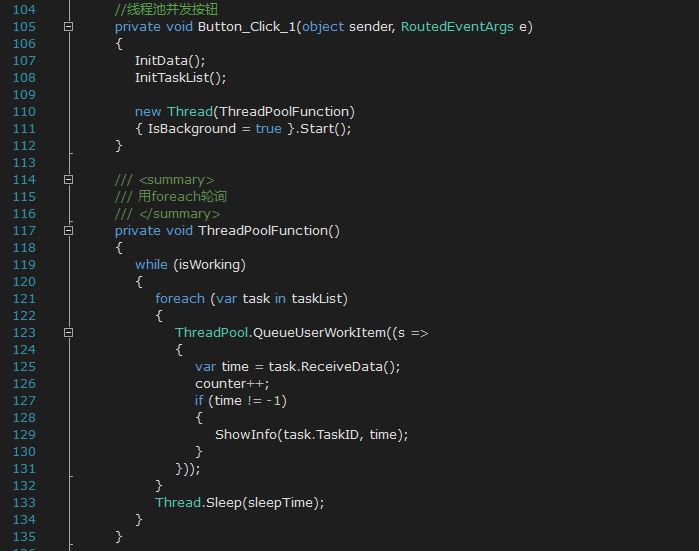
0x02 保持线程的并发

保持线程的并发非常直观，就是每建立一个对象就开一个新线程循环进行ReceiveData操作，当接收到数据就把相关信息输出到主界面上。代码如下所示：



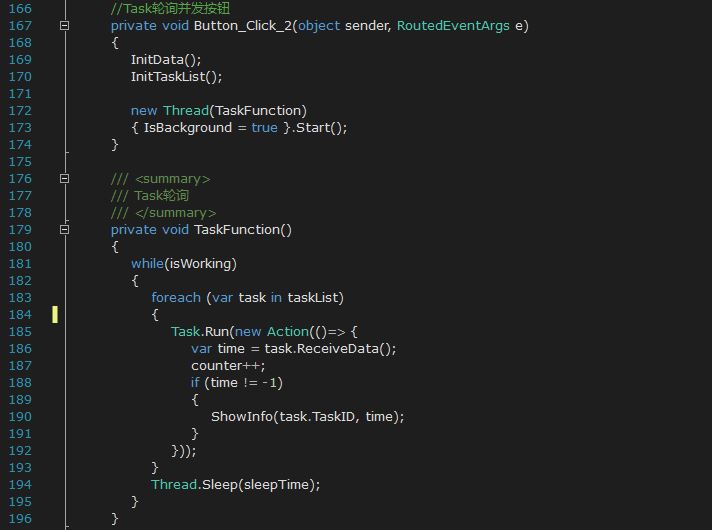
0x03 使用ThreadPool轮询并发

方法是使用一个List（或其他容器）把所有的对象放进去，创建一个线程（为了防止UI假死，由于这个线程创建后会一直执行切运算密集，所以使用TheadPool和Thread差别不大），在这个线程中使用foreach(或for)循环依次对每个对象执行ReceiveData方法，每次执行的时候创建一个线程池线程来执行。代码如下：



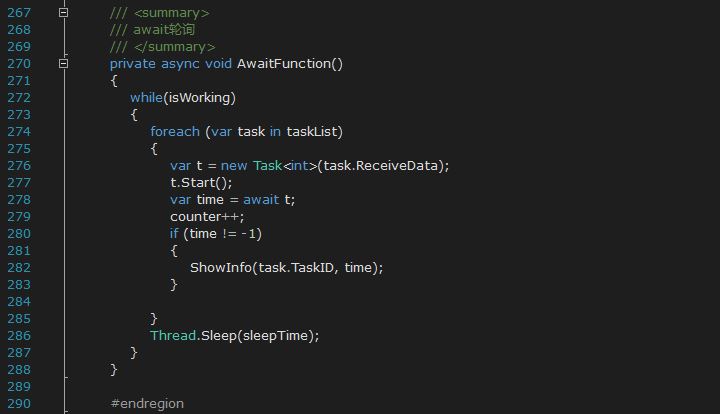
0x04使用Task轮询并发

方法与ThreadPool类似，只是每次创建线程池线程执行ReceiveData方法时是通过Task创建的线程。代码如下所示：



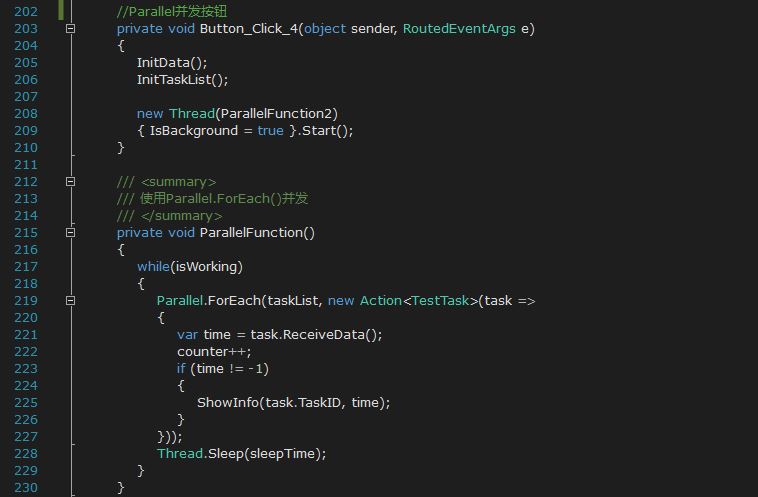
0x05 使用await轮询并发

方法与ThreadPool类似，只是每次创建线程池线程执行ReceiveData方法时是通过await等待操作。代码如下：



0x06 使用Parallel并发

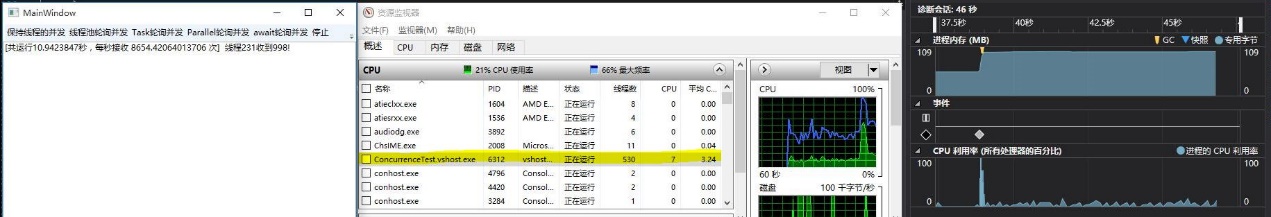
这是FCL提供的一种方法，Parallel.ForEach中每次方法都是异步执行，执行采用的是线程池线程。代码如下所示：



0x07 测试结果

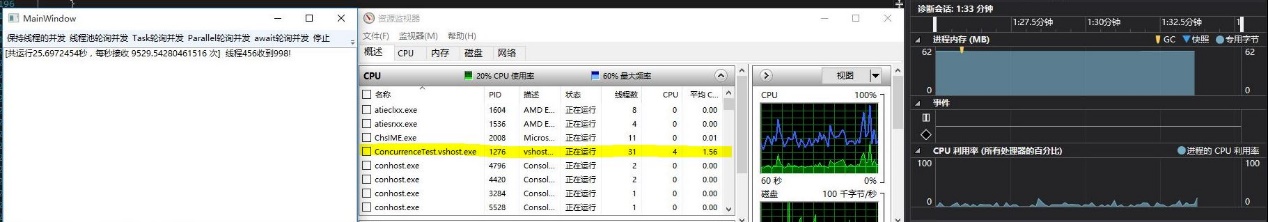
创建500个对象来模拟500个连接的情况。其中测试结果中的每秒接收次数会有个波动范围，主要参照百位以上。使用线程池线程的几个方法（ThreadPool、Task、await、Parallel）中程序的线程数略有差别，可能跟执行环境有关，难以表明实质性差异。其中await因为线程切换导致线程执行时间略长，使得线程池需要多创建一些线程。

**1、保持线程的并发**



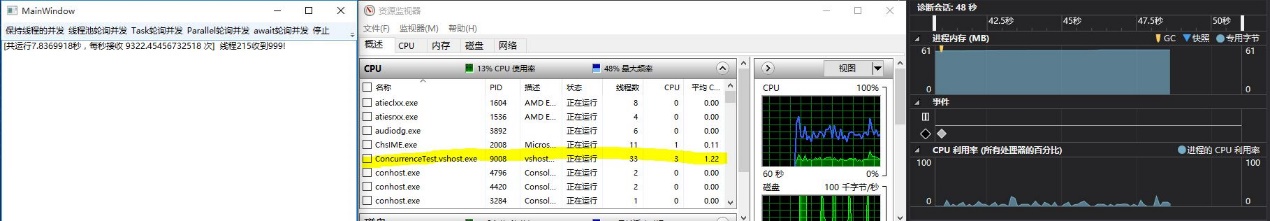
平均每秒接收8654次数据。在任务开始后会创建500个线程，由于每个线程都需要单独的栈空间来执行，内存消耗较大。频繁切换线程也会加重CPU的负担。

**2、ThreadPool轮询并发**



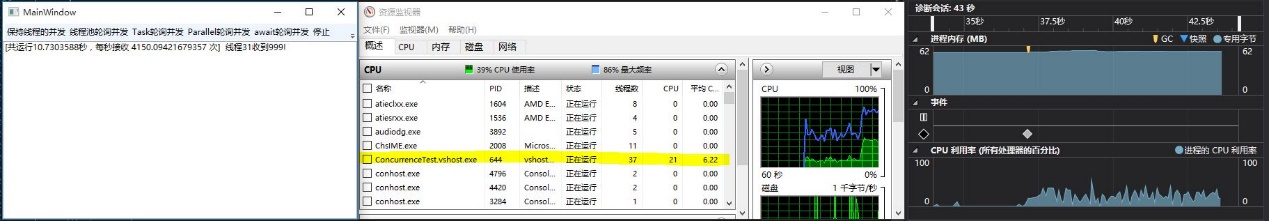
平均每秒接受9529次数据。由于实现了线程池线程的复用，无需创建太多线程，内存没有出现波动，CPU消耗也比较均匀。

**3、Task轮询并发**



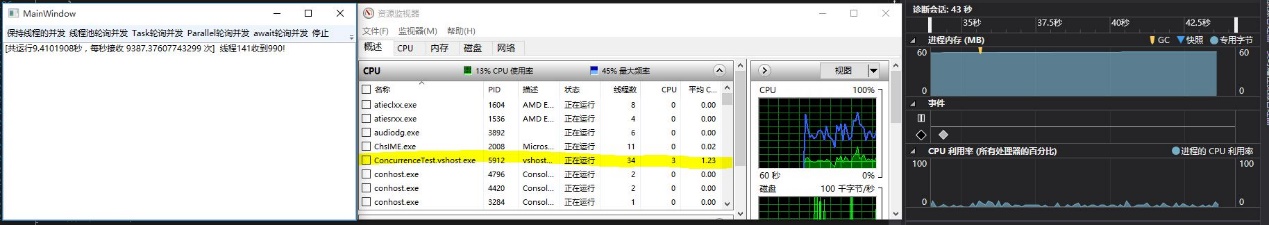
平均每秒接收9322次数据，由于Task也是基于线程池的封装，因此与ThreadPool结果差别不大。

**4、await轮询并发**



平均每秒接收4150次。await也是使用线程池线程，所以在内存开销和线程数上与其他使用线程池线程的方法没有太大差别。但await在等待完毕后会将执行上下文从线程池线程切换回调用线程，因此CPU开销较大。

**5、Parallel并发**



看名字就知道这个设计出来就是应用于这种使用环境的，平均每秒接收9387次数据，也是使用线程池线程，所以内存和CPU消耗与ThreadPool和Task差不多。但不需要自己写foreach(for)循环，只要写循环体即可。

0x08 结论

首先明显能看出来的是使用轮询的方式比保持线程能节省很多资源，特别是内存。而且在处理效率上轮询的方式（每秒接收9300-9500次）比保持线程还要高（每秒8600+）。因此在这种并发模型下应该使用轮询的方式以节省资源并提高并发效率。

实际上硬拿await来比较是不太公平的，await被设计出来就不是应用于这种场景的。不管是之前关于异步的测试还是并发的测试，基于线程池的方案相差都不大。因此思路对了的情况下使用ThreadPool总是没错的。但有些类型把ThreadPool包装了以更好适应某些特殊场景，因此有了Task、await、Parallel等。而在这次的测试条件下显然Parallel是最合适的，与直接使用ThreadPool相比资源开销和执行效率一样，但代码更少。

实际上根据并发线程的数量和线程执行时间的不同，轮询和保持县城的性能差异也在发生变化，感兴趣的可下载代码自行测试。

0x09 相关下载

测试代码下载链接：https://github.com/durow/TestArea/tree/master/AsyncTest/ConcurrenceTest