.net4.5 异步编程 Async 和 Await

先引入别人的一段话

从4.0版本开始.NET引入并行编程库，用户能够通过这个库快捷的开发并行计算和并行任务处理的程序。在4.5版本中.NET又引入了Async和Await两个新的关键字，在语言层面对并行编程给予进一步的支持，使得用户能以一种简洁直观的方式实现并行编程。因为在很多文档里针对Async和Await这两个关键字的使用都被称为异步编程，为了更符合大众的阅读习惯，我们使用异步编程这个叫法，意思上和并行编程完全一样。

嗯，《并行开发》，这并不是新技术，老技术新包装而已，本质还是多线程，我的理解。

而4.5的新关键字 Async 和 Await 只是让他实现了简洁直观的编程方式，已同步开发的编程方式，实现异步执行的效果，不必你再去写什么回调之类的东西。

事实上我对 使用Async和Await 关键字上，还不太适应，毕竟已经习惯了，异步回调的方式开发，我还在慢慢的适应当中。

       闲话少说，进入正题

       说起 这两个关键字，就不能不说起 .net的并行库。并行库，新名词，本质还是多线程，而线程大家应该已经不陌生了，4.0的并行库，提供了一种更简单的多线程使用方式（这是语法糖），如：创建线程，线程池，阻塞，信号量，等待，取消，回调，使得对线程的操作更加简单便捷，而异步带来的好处是很多的，最大的方面还是性能，最常见的应用场景应该就是winform程序了，大量耗时操作会阻塞UI的主线程，界面假死。

让我们来看一个简单的例子吧！

同步开发方式

static void Main(string[] args)

        {

            // 同步方式

            Console.WriteLine("同步方式测试开始！");

            SyncMethod(0);

            Console.WriteLine("同步方式结束！");

            Console.ReadKey();

        }

        // 同步操作

        private static void SyncMethod(int input)

        {

            Console.WriteLine("进入同步操作！");

            var result = SyancWork(input);

            Console.WriteLine("最终结果{0}", result);

            Console.WriteLine("退出同步操作！");

        }

        // 模拟耗时操作（同步方法）

        private static int SyancWork(int val)

        {

            for (int i = 0; i < 5; ++i)

            {

                Console.WriteLine("耗时操作{0}", i);

                Thread.Sleep(100);

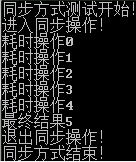
                val++;

            }

            return val;

        }

同步运行结果截图，



异步开发方式

static void Main(string[] args)

        {

            // 异步方式

            Console.WriteLine("\n异步方式测试开始！");

            AsyncMethod(0);

            Console.WriteLine("异步方式结束！");

            Console.ReadKey();

        }

        // 异步操作

        private static async void AsyncMethod(int input)

        {

            Console.WriteLine("进入异步操作！");

            var result = await AsyncWork(input);

            Console.WriteLine("最终结果{0}", result);

            Console.WriteLine("退出异步操作！");

        }

        // 模拟耗时操作（异步方法）

        private static async Task<int> AsyncWork(int val)

        {

            for (int i = 0; i < 5; ++i)

            {

                Console.WriteLine("耗时操作{0}", i);

                await Task.Delay(100);

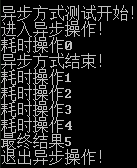
                val++;

            }

            return val;

        }

异步结果截图



异步执行的代码结构跟同步执行的代码结构一模一样的(除了新关键字修饰符)，这在代码可读性上面是十分有帮助的， 这就是我在上面说到的新关键字 带来的效果（事实上新关键字本身没有什么神秘的）。

目前net框架中，很多函数都已经提供了对await修饰符的调用方式，IO库就提供了，大家可以去看看 文件服务器的代码，这里已经实现了，对http请求的异步执行（节约http链接资源，提高IIS的吞吐量），与文件读取写入的异步执行（提高IO性能，不阻塞主线程代码执行）。

如何知道 .net提供的内API支持 await修饰符，一般情况下，API名字后面带有Async字样的都是支持的（命名规范）。

细心的同学应该明白上一句的伏笔了，“如何知道.net提供的API支持修饰符“

换句话说，并不是所有的都支持。

那么如何让它们支持，或让自己写的函数 API 支持新关键的使用？

答案是使用并行库的Task任务类，

你的API 返回值是 Task或者 Task<T> 就行了

例如：

public Task<string> MyAsync()

        {

            var t = new Task<string>((str) =>

                {

                    var dt = DateTime.Now;

                    Thread.Sleep(4000);

                    return String.Format("({0}){1} - {2}",

                        Thread.CurrentThread.ManagedThreadId, dt, DateTime.Now);

                }, null);

            t.Start();

            return t;

        }

那么这个函数 就支持 await修饰符了，

嗯·还有await修饰符 是成双成对出现的 你API 里使用await修饰符，那你的这个API的 返回值声明就必须带上 async 修饰符。

下次在分享并行开发。