.NET4.5新特性之异步编程（Async和Await）的使用 - 文章 - 伯乐在线

一、简介

首先来看看.net的发展中的各个阶段的特性：NET 与C# 的每个版本发布都是有一个“主题”。即：C#1.0托管代码→C#2.0泛型→C#3.0LINQ→C#4.0动态语言→C#4.5异步编程

下面我来简单的介绍一下异步编程：异步编程，在 .NET Framework 4.5 和 Windows 运行时利用异步支持。 编译器可执行开发人员曾进行的高难度工作，且应用程序保留了一个类似于同步代码的逻辑结构。 因此，你只需做一小部分工作就可以获得异步编程的所有好处。
（https://msdn.microsoft.com/zh-cn/library/hh191443.aspx）

所谓的异步编程是利用CPU空闲时间和多核的特性，它所返回的Task或Task是对await的一个承诺，当任务执行完毕后返回一个结果给接收者。这里看到这个可能各位不太明白，不要紧，下面会有讲解。

二、使用说明

- 方法签名包含一个 Async 或 async 修饰符。
- 按照约定，异步方法的名称以“Async”后缀结尾。
- 返回类型为下列类型之一：
  - 如果你的方法有操作数为 TResult 类型的返回语句，则为 Task。
  - 如果你的方法没有返回语句或具有没有操作数的返回语句，则为 Task。
  - Sub in Visual Basic) if you’re writing an async event handler.”>如果你编写的是异步事件处理程序，则为 Void（Visual Basic 中为 Sub）。

  有关详细信息，请参见本主题后面的“返回类型和参数”。

- 方法通常包含至少一个 await 表达式，该表达式标记一个点，在该点上，直到等待的异步操作完成方法才能继续。 同时，将方法挂起，并且控件返回到方法的调用方。（这里所谓的挂起就是上文所提到的承诺，异步方法承诺会给调用方一个结果）

三、示例

实践才是检验真知的最佳途径。

C#

```
using System;
using System.Diagnostics;
```

```
using System.Net.Http;
using System.Threading.Tasks;
namespace 异步递归
{
        class Program
        {
                static void Main(string[] args)
                {
                        Stopwatch stopwatch = new Stopwatch();
                        stopwatch.Start();
                        ConsoleAsync1();
                        stopwatch.Stop();
                        Console.WriteLine("同步方法用时：" +
stopwatch.ElapsedMilliseconds);
                        stopwatch.Reset();
                        stopwatch.Start();
                        ConsoleAsync();
                        stopwatch.Stop();
                        Console.WriteLine("异步方法用时："+ stopwatch.ElapsedMilliseconds);
                        Console.Read();
                }
                private static async void ConsoleAsync()
                {
                        Console.WriteLine("异步方法开始");
                        Console.WriteLine("Result:" + await SumAsync(10));
                        Console.WriteLine("异步方法结束");
                }
                private static async Taskint> SumAsync(int part)
                {
                        if ((part += 10) >= 100)
                        {
                                return 100;
                        }
                        HttpClient client = new HttpClient();
                        Taskstring> getStringTask =
client.GetStringAsync("http://msdn.microsoft.com");
                        Console.WriteLine(DateTime.Now.Millisecond + " 异步 " + (await
getStringTask).Length);
                        return await SumAsync(part);
                }
                private static void ConsoleAsync1()
                {
```

```
                Console.WriteLine("同步方法开始");
                Console.WriteLine("Result:" + SumAsync1(10));
                Console.WriteLine("同步方法结束");
            }
            private static int SumAsync1(int part)
            {
                if ((part += 10) >= 100)
                {
                    return 100;
                }
                HttpClient client = new HttpClient();
                Taskstring> getStringTask =
client.GetStringAsync("http://msdn.microsoft.com");
                Console.WriteLine(DateTime.Now.Millisecond + " 同步 " +
getStringTask.Result.Length);
                return SumAsync1(part);
            }
        }
}
```

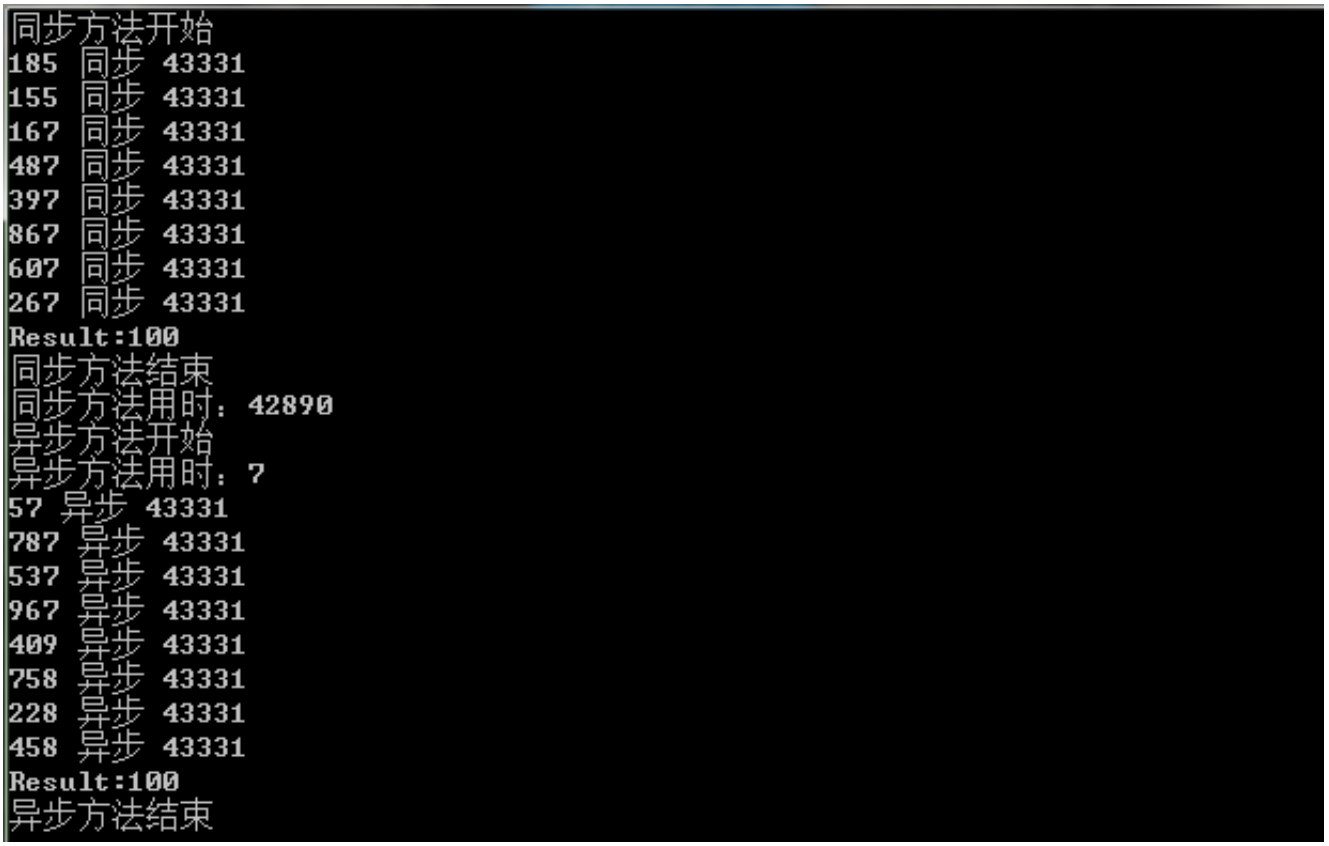
示例介绍：

1、这个例子中有两种实现方式：（1）利用异步编程的方式实现（2）利用普通同步方式实现

2、同时这个例子中实现了递归，这个可以不用考虑，博主只是想验证一下在异步的情况下，递归是否有效而已，实验结果为有效。

3、这段代码中的GetStringAsync（）方法是获取远程界面内容用的，主要目的是延长响应时间。

程序结果如下：



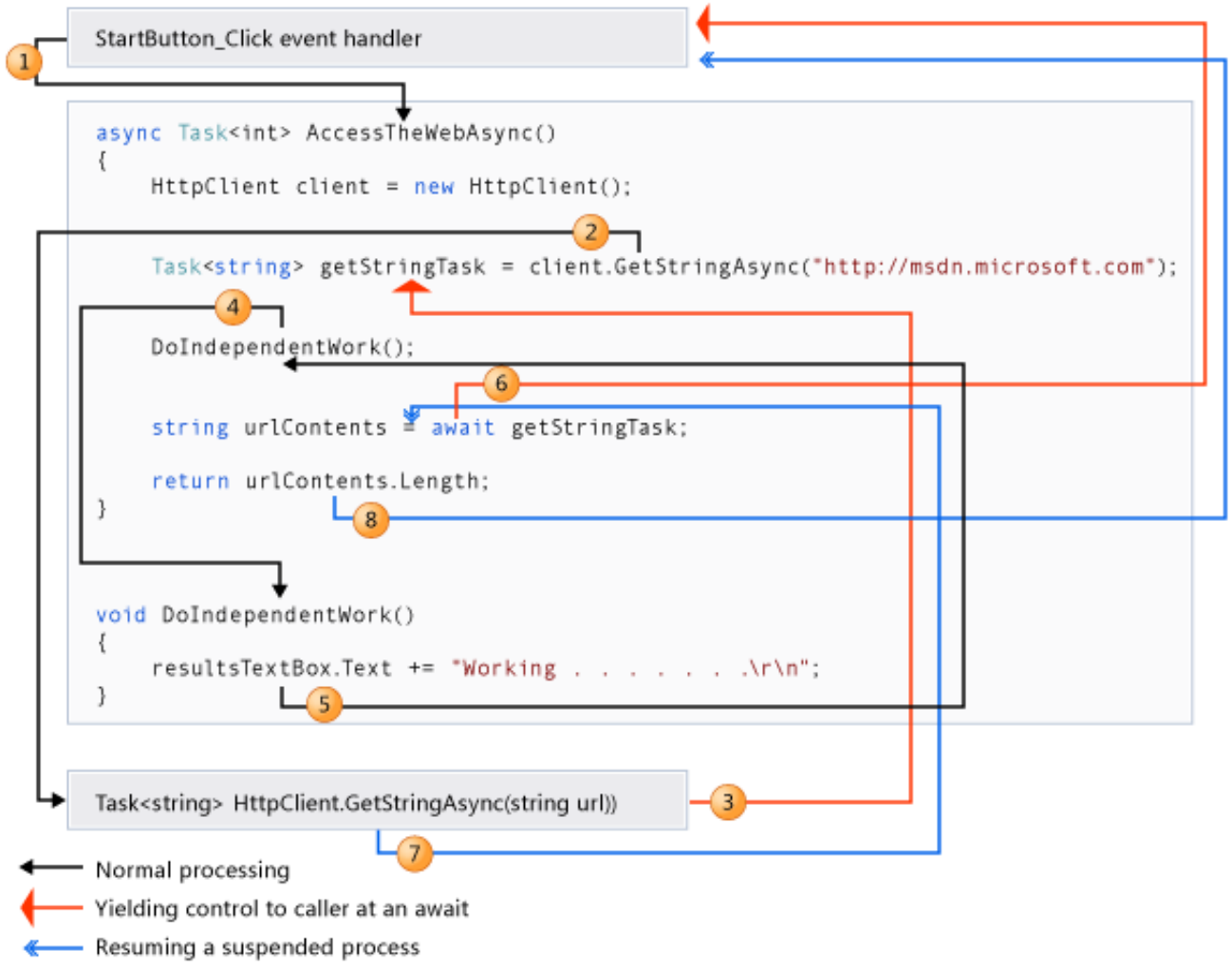
结果说明：

1、同步方法按规矩进行，有条不紊。

2、异步方法直接执行完毕，用时7毫秒。执行过程异步于主线程。

四、尾语

微软的官方文档很值得学习，大家感兴趣的可以看看去。这里引一个流程原理图，所对应的示例到文档中去看，链接上方已给。



（源程序感兴趣的可以留言，我随时提供）

加入伯乐在线专栏作者。扩大知名度，还能得赞赏！详见《招募专栏作者》

1 赞 1 收藏 1 评论

合作联系

Email：bd@Jobbole.com

QQ： 2302462408 （加好友请注明来意）

更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子

头条 - 分享和发现有价值的内容与观点

相亲 - 为IT单身男女服务的征婚传播平台

资源 - 优秀的工具资源导航

翻译 - 翻译传播优秀的外文文章

文章 - 国内外的精选文章

- 设计 - UI,网页，交互和用户体验
- iOS - 专注iOS技术分享
- 安卓 - 专注Android技术分享
- 前端 - JavaScript, HTML5, CSS
- Java - 专注Java技术分享
- Python - 专注Python技术分享