.NET Web应用中为什么要使用async/await异步编程

xhznl DotNetCore实战 2 days ago

阅读本文大概需要 3 分钟。

前言

1. 什么是async/await?
   await和async是.NET Framework4.5框架、C#5.0语法里面出现的技术，目的是用于简化异步编程模型。
2. async和await的关系?
   async和await是成对出现的。
   async出现在方法的声明里，用于批注一个异步方法。光有async是没有意义的。
   await出现在方法内部，Task前面。只能在使用async关键字批注的方法中使用await关键字。

```
private async Task DoSomething()
{
    await Task.Delay(TimeSpan.FromSeconds(10));
}
```

3. async/await会创建新的线程吗?
   不会。async/await关键字本身是不会创建新的线程的，但是被await的方法内部一般会创建新的线程。
4. asp.net mvc/webapi action中使用async/await会提高请求的响应速度吗?
   不会。

正题

我们都知道web应用不同于winform、wpf等客户端应用，客户端应用为了保证UI渲染的一致性往往都是采用单线程模式，这个UI线程称为主线程，如果在主线程做耗时操作就会导致程序界面假死，所以客户端开发中使用多线程异步编程非常必要。
可web应用本身就是多线程模式，服务器会为每个请求分配工作线程。
既然async/await不能创建新线程，又不能使提高请求的响应速度，那.NET Web应用中为什么要使用async/await异步编程呢?

> 在 web 服务器上，.NET Framework 维护用于处理 ASP.NET 请求的线程池。 当请求到达时，将调度池中的线程以处理该请求。 如果以同步方式处理请求，则处理请求的线程将在处理请求时处于繁忙状态，并且该线程无法处理其他请求。

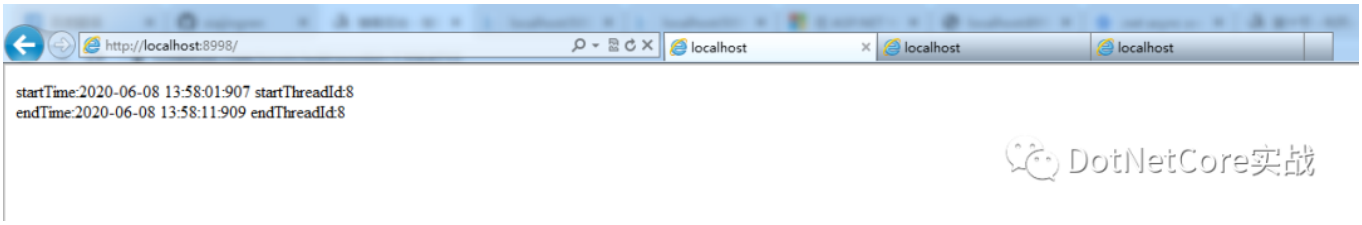
> 在启动时看到大量并发请求的 web 应用中，或具有突发负载（其中并发增长突然增加）时，使 web 服务调用异步会提高应用程序的响应能力。 异步请求与同步请求所需的处理时间相同。 如果请求发出需要两秒钟时间才能完成的 web 服务调用，则该请求将需要两秒钟，无论是同步执行还是异步执行。 但是，在异步调用期间，线程在等待第一个请求完成时不会被阻止响应其他请求。 因此，当有多个并发请求调用长时间运行的操作时，异步请求会阻止请求队列和线程池的增长。

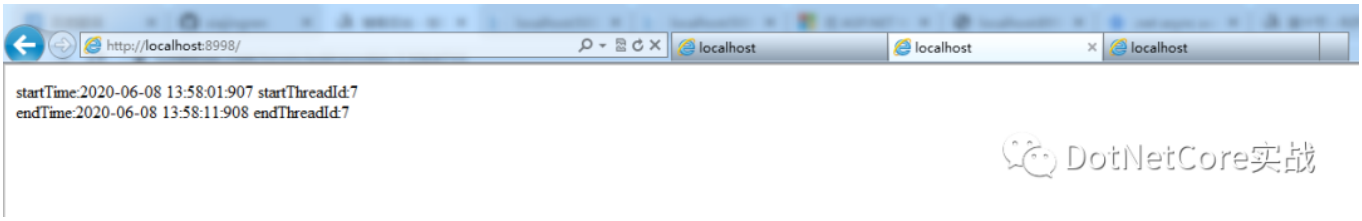
下面用代码来实际测试一下：

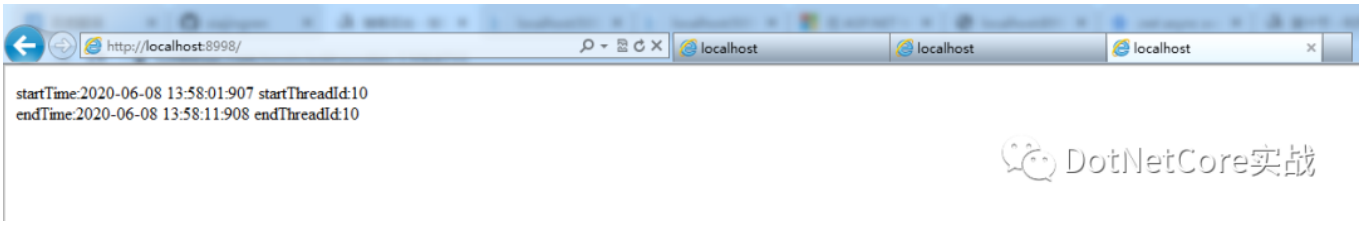
- 先是同步的方式，代码很简单，就是输出一下请求开始和结束的时间和线程ID：

```
        public ActionResult Index()
        {
            DateTime startTime = DateTime.Now;//进入DoSomething方法前的时间
            var startThreadId = Thread.CurrentThread.ManagedThreadId;//进入DoSomething方法前的线程ID

            DoSomething();//耗时操作

            DateTime endTime = DateTime.Now;//完成DoSomething方法的时间
            var endThreadId = Thread.CurrentThread.ManagedThreadId;//完成DoSomething方法后的线程ID
            return Content($"startTime:{ startTime.ToString("yyyy-MM-dd HH:mm:ss:fff") } startThreadId:{ startThreadId }<br/>endTime:{ endTime.ToString("yyyy-MM-dd HH:mm:ss:fff") } endThreadId:{ endThreadId }<br/><br/>");
        }

        /// <summary>
        /// 耗时操作
        /// </summary>
        /// <returns></returns>
        private void DoSomething()
        {
            Thread.Sleep(10000);
        }
```

使用浏览器开3个标签页进行测试（因为浏览器对同一域名下的连接数有限制，一般是6个左右，所以就弄3个吧）：





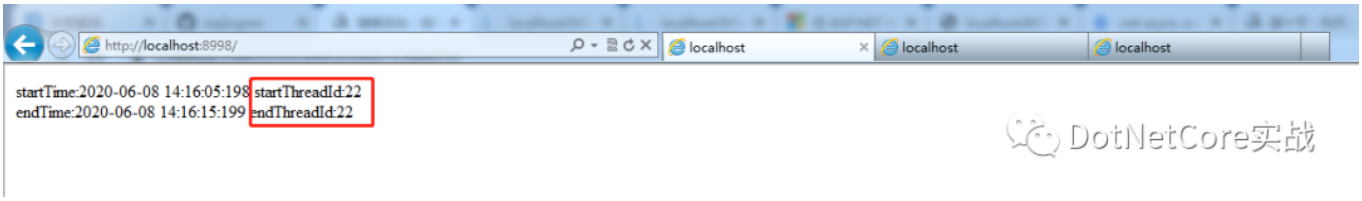


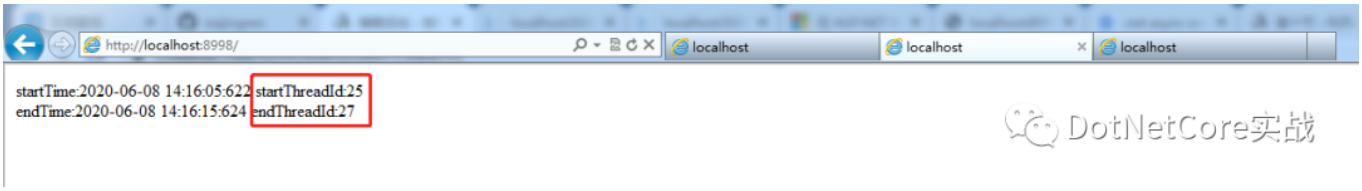
可以看到耗时都是10秒，开始和结束的线程ID一致。

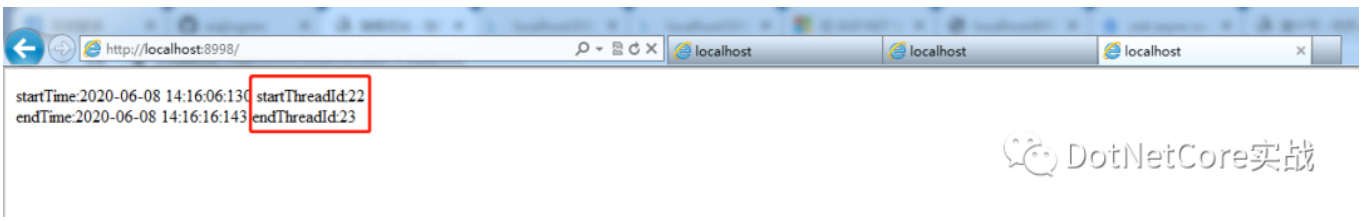
- 下面改造成异步的：

```
        public async Task<ActionResult> Index()
        {
            DateTime startTime = DateTime.Now;//进入DoSomething方法前的时间
            var startThreadId = Thread.CurrentThread.ManagedThreadId;//进入DoSomething方法前的线程ID

            await DoSomething();//耗时操作

            DateTime endTime = DateTime.Now;//完成DoSomething方法的时间
            var endThreadId = Thread.CurrentThread.ManagedThreadId;//完成DoSomething方法后的线程ID
            return Content($"startTime:{ startTime.ToString("yyyy-MM-dd HH:mm:ss:fff") } startThreadId:{ startThreadId }<br/>endTime:{ endTime.ToString("yyyy-MM-dd HH:mm:ss:fff") } endThreadId:{ endThreadId }<br/><br/>");
        }

        /// <summary>
        /// 耗时操作
        /// </summary>
        /// <returns></returns>
        private async Task DoSomething()
        {
            await Task.Run(() => Thread.Sleep(10000));
        }
```

结果：

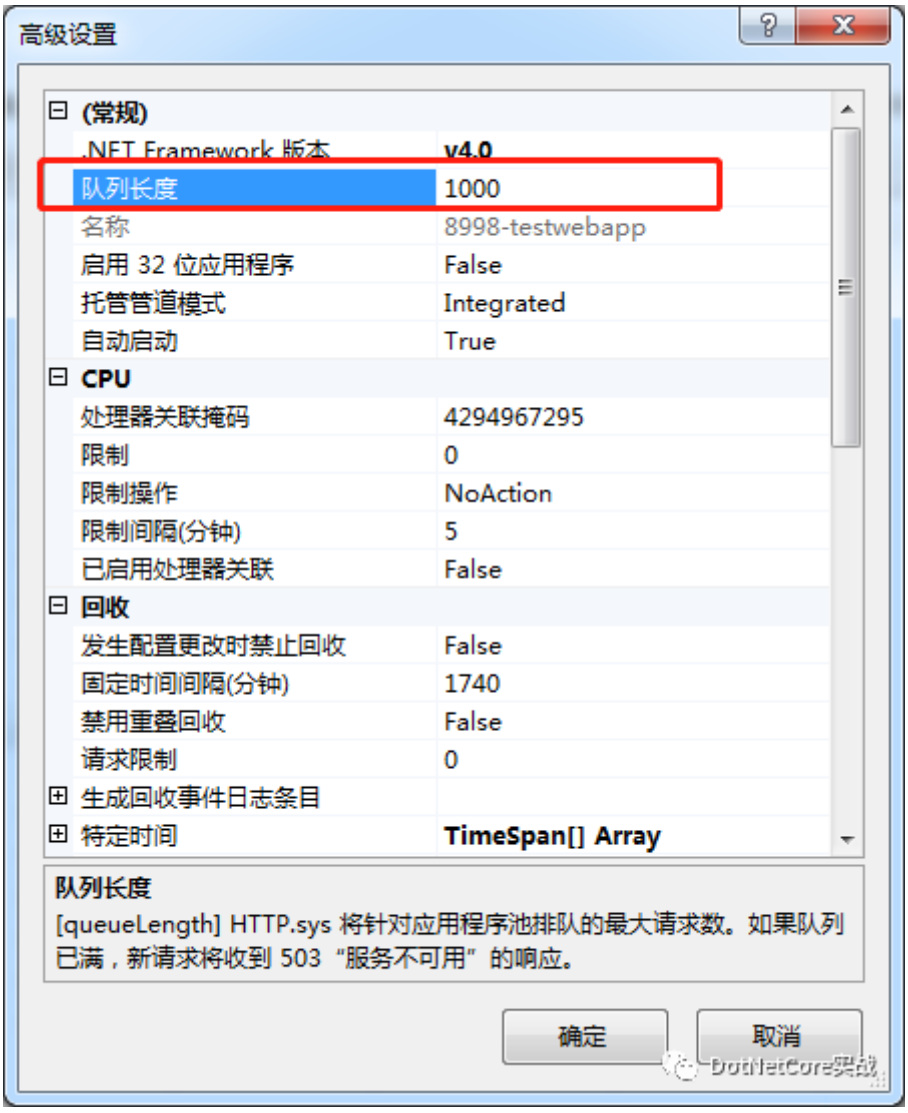






可以看到3次请求中，虽然耗时都是10秒，但是出现了开始和结束的线程ID不一致的情况，ID为22的这个线程工作了多次，这意味着**使用异步方式在同一时间可以处理更多的请求**！

IIS默认队列长度：



> await关键字不会阻塞线程直到任务完成。 它将方法的其余部分注册为任务的回调，并立即返回。 当await的任务最终完成时，它将调用该回调，并因此在其中断时继续执行方法。

简单来说：就是使用同步方法时，线程会被耗时操作一直占有，直到耗时操作完成。而使用异步方法，程序走到await关键字时会立即return，释放线程，余下的代码会放进一个回调中（Task.GetAwaiter()的UnsafeOnCompleted(Action)回调），耗时操作完成时才会回调执行，所以async/await是语法糖，其本质是一个状态机。

那是不是所有的action都要用async/await呢?
不是。一般的磁盘IO或者网络请求等耗时操作才考虑使用异步，不要为了异步而异步，异步也是需要消耗性能的，使用不合理会适得其反。

结论

async/await异步编程不能提升响应速度，但是可以提升响应能力（吞吐量）。异步和同步各有优劣，要合理选择，不要为了异步而异步。

往期精彩回顾

Read more