.NET 中有多少种定时器?

追逐时光者 2024年08月03日 08:00 广东

文章来源：博客园
文章作者：czwy
文章地址：https://www.cnblogs.com/czwy/p/17862702.html

前言

.NET中至少有6种定时器，每一种定时器都有它的用途和特点。根据定时器的应用场景，可以分为UI相关的定时器和UI无关的定时器。

本文将简单介绍这6种定时器的基本用法和特点。

UI定时器

.NET中的UI定时器主要是WinForm、WPF以及WebForm中的定时器。分别为：

- `System.Windows.Forms.Timer`
- `System.Windows.Threading.DispatcherTimer`
- `System.Web.UI.Timer`

通常情况下，WinForm、WPF中的定时器是在UI线程上执行回调函数，因此可以直接访问UI元素。由于WinForm、WPF支持单线程单元模型（Single-Thread Apartment，STA），定时器间隔事件是在UI线程上触发，因此，不用担心线程安全问题。 `System.Web.UI.Timer` 是通过Javascript定时器和服务端异步回调实现，也是单线程的。

请注意，这里说的是通常情况，后边介绍 `System.Windows.Threading.DispatcherTimer` 时会提到在非UI线程创建 `DispatcherTimer` 时也无法直接访问UI元素。

System.Windows.Forms.Timer

`System.Windows.Forms.Timer` 针对WinForm应用进行了优化，是只能在WinForm上使用的定时器。这个定时器是针对单线程环境设计的，是在UI线程上处理定时任务。它要求用户代码有可用的UI消息泵，定时任务须在UI线程上运行，或者跨线程通过 `Invoke` 或者 `BeginInvoke` 封送(marshal)到UI线程上运行。

其优点是使用简单，只需通过给 Interval 属性赋值来设置时间间隔，并注册 Tick 事件处理定时任务。其缺点是精度不高，精度为55毫秒，也就是 Interval 赋值小于55时，也是55毫秒触发一次定时任务。

```
public partial class TimerFrom : Form
{
    private System.Windows.Forms.Timer digitalClock;
    private void TimerFrom_Load(object sender, EventArgs e)
    {
        digitalClock = new System.Windows.Forms.Timer();//创建定时器
        digitalClock.Tick += new EventHandler(HandleTime);//注册定时任务事件
        digitalClock.Interval = 1000;//设置时间间隔
        digitalClock.Enabled = true;
        digitalClock.Start(); //开启定时器
    }
    public void HandleTime(Object myObject, EventArgs myEventArgs)
    {
        labelClock.Text = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
    }
    private void frmTimerDemo_FormClosed(object sender, FormClosedEventArgs e)
    {
        digitalClock.Stop();//停止定时器
        digitalClock.Dispose();
    }
}
```

System.Windows.Threading.DispatcherTimer

System.Windows.Threading.DispatcherTimer是WPF中的定时器，它是基于Dispatcher对象的(并不是基于UI线程的)。DispatcherTimer的定时任务是像其他操作一样放在Dispatcher队列上，其执行操作时间依赖于队列中其他任务及其优先级，因此，DispatcherTimer不保证在时间间隔发生时准确执行，只保证不会在时间间隔发生前执行。

> Dispatcher 为特定线程维护工作项（操作）的优先级队列，在线程上创建 Dispatcher 对象时，它成为唯一可以关联该线程的 Dispatcher 对象，WPF中， DispatcherObject 只能被与之关联的 Dispatcher 对象访问，也就是非UI线程中无法直接访问UI元素（WPF中的UI元素都是派生自 DispatcherObject ）

此外， `DispatcherTimer` 不像 `System.Windows.Forms.Timer` 那样只在UI线程上创建才能触发 `Tick` 事件，它在非UI线程下创建也可以触发 `Tick` 事件，此时访问UI元素也需要通过 `Invoke` 或者 `BeginInvoke` 封送(marshal)到UI线程上运行。其优点也是简单易用，适合在UI线程上执行任务或触发事件，缺点是精度不准确，可能存在延迟。

```
private void Dt_Tick(object sender, EventArgs e)
{
    Dispatcher.BeginInvoke((Action)delegate ()
    {
        text1.Text = DateTime.Now.ToString();
    });
    Console.WriteLine(DateTime.Now.ToString());
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    Task.Run(() =>{
        DispatcherTimer dt = new DispatcherTimer();
        dt.Tick += Dt_Tick;
        dt.Interval = TimeSpan.FromSeconds(1);
        dt.Start();
        Dispatcher.Run();
    });
}
```

上述代码中， `DispatcherTimer` 是非UI线程中创建，定时任务中访问UI元素text1，需要通过 `Invoke` 或者 `BeginInvoke` 封送(marshal)到UI线程上运行，而 `Console.WriteLine` 则可以直接运行。

System.Web.UI.Timer

`System.Web.UI.Timer` 是仅适用于 `.NET Framework` 的 `ASP.NET` 组件。通过Javascript定时器和服务端异步回调实现。每次触发定时器时，只能执行一个异步回调方法，而其他的异步回调方法需要等待前一个异步回调方法执行完毕后才能执行。这样可以保证在任意时刻只有一个异步回调方法在执行，避免了多线程并发执行的问题。

UI无关定时器

从 .NET 6开始，UI无关定时器有三个：

- `System.Threading.Timer`
- `System.Timers.Timer`
- `System.Threading.PeriodicTimer` （.NET 6+）

System.Threading.Timer

`System.Threading.Timer` 是最基础轻量的定时器，它将定期在线程池线程上执行单个回调方法。在创建定时器对象时必须指定回调方法，并且后续不能修改，同时也可以指定定时器回调开始执行的时间以及时间间隔。定时器创建后可以通过 `Change` 方法修改回调开始执行的时间以及时间间隔。该定时器的优点是轻量，精度相对较高，与Windows操作系统时钟精度一致，大约15毫秒。但因为是基于线程池的，所以在任务执行时间较长或者线程池过载时，会出现延迟。其缺点是使用不太方便，定时器创建后无法修改回调方法。

```
var stateTimer = new
var autoEvent = new AutoResetEvent(false);
Timer(CheckStatus, autoEvent, 1000,250);

private int invokeCount=0;

public void CheckStatus(Object stateInfo)
{
    AutoResetEvent autoEvent = (AutoResetEvent)stateInfo;
    Console.WriteLine("{0} Checking status {1,2}.",DateTime.Now.ToString("h:mm:ss.fff"),(++ir

    if(invokeCount == 10)
    {
        invokeCount = 0;
        autoEvent.Set();
    }
}
```

System.Timers.Timer

System.Timers.Timer在内部使用System.Threading.Timer，并公开了更多的属性，如AutoReset, Enabled或SynchronizingObject，这些属性允许配置回调的执行方式。此外，Tick事件允许注册多个处理程序。因此，一个定时器可以触发多个处理程序。还可以在计时器启动后更改处理程序。

与System.Threading.Timer相似，其优点也是精度相对较高，与Windows操作系统时钟精度一致，大约15毫秒。因为默认（或者SynchronizingObject=null时）是基于线程池的，所以在任务执行时间较长或者线程池过载时，会出现延迟。但使用要更简便一些。

```
public partial class TimerFrom : Form
{
```

```
    private System.Timers.Timer timer;
    private void TimerFrom_Load(object sender, EventArgs e)
    {
        // 支持注册多个处理程序
        timer.Elapsed += (sender, e) => { label1.Text = DateTime.Now.ToLongTimeString(); };
        timer.Elapsed += (sender, e) => { Console.WriteLine(DateTime.Now.ToLongTimeString());
        //自定义回调执行的方式（指定对象所在的线程），SynchronizingObject=null时在线程池上执行
        timer.SynchronizingObject = this;
        timer.AutoReset = true;
        timer.Start();
    }
}
```

本例中将SynchronizingObject属性设置为Form对象，因此Elapsed的处理程序在UI线程上执行，可以直接修改label1.Text，如果SynchronizingObject属性为null，处理程序则是在线程池线程上执行，修改label1.Text时需要通过Invoke或者BeginInvoke封送(marshal)到UI线程上运行。

System.Threading.PeriodicTimer

System.Threading.PeriodicTimer是 .NET 6中引入的定时器。它能方便地使用异步方式，它没有Tick事件，而是提供WaitForNextTickAsync方法处理定时任务。通常是使用While循环结合CancellationToken一起使用。

和CancellationToken一起用的时候需要注意，如果CancellationToken被取消的时候会抛出一个OperationCanceledException需要考虑自己处理异常。相比之前的定时器来说，有下面几个特点：

1、没有callback 来绑定事件;

2、不会发生重入，只允许有一个消费者，不允许同一个PeriodicTimer在不同的地方同时WaitForNextTickAsync，不需要自己做排他锁来实现不能重入；

3、异步化。之前的 timer 的 callback 都是同步的，使用新 timer 可以使用异步方法，避免了编写 Sync over Async 代码；

4、Dispose 之后，实例就无法使用，并且 WaitForNextTickAsync 始终返回 false。

```
var cts = new CancellationTokenSource(TimeSpan.FromSeconds(15));
using (var timer = new PeriodicTimer(TimeSpan.FromSeconds(1)))
{
    try
    {
        while (await timer.WaitForNextTickAsync(cts.Token))
        {
            await Task.Delay(3000);
            Console.WriteLine($"ThreadId is {Thread.CurrentThread.ManagedThreadId} --- Time i
        }
    }
    catch (OperationCanceledException)
    {
        Console.WriteLine("Operation cancelled");
    }
}
```

小结

我们在开发过程中遇到的坑往往不是技术本身的坑，而是我们滥用没有掌握的技术导致的，在有多种技术方案可选的时候，通常只关注技术的优点，忽略了技术适用场景及其局限性。

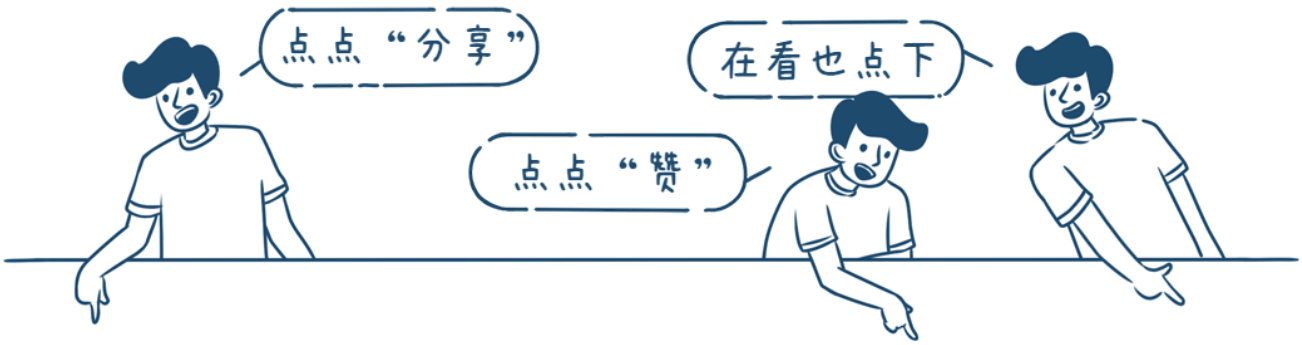
.NET中几种定时器各自都有其适用场景和不足，但都不支持高精度计时。了解这些有助于我们在开发过程中选择合适定时器，避免遇到问题后被动地替换解决方案。

追逐时光者
DotNetGuide官方公众号，微软MVP，专注于C#/.NET/.NET Core学习、工作、面试干...
515篇原创内容

公众号

学习是一个永无止境的过程，你知道的越多，你不知道的也会越多，在有限的时间内坚持每天多学一点，你一定能成为你想要成为的那个人。不积跬步无以至千里，不积小流无以成江河！！！

See you next good day



.NET 343　C# 274　拾遗补漏 38　定时器 1

.NET · 目录

上一篇
提升生产力：8个.NET开源且功能强大的快速开发框架

下一篇
C#/.NET/.NET Core推荐学习书籍（24年8月更新）