

细说C#多线程那些事：线程基础 - 文章 - 伯乐在线



原文出处：love.net

我第一次接触“线程”的概念时，觉得它深奥难懂，看了好多本书，花了很长时间才领悟到它的真谛。现在我就以一个初学者的心态，把我所理解的“多线程”描述给大家。这一次是系列文章，比较完整的展示与线程相关的基本概念。希望对初学者有所帮助。

如果你是高手，请你别继续看，会浪费你宝贵的时间。

一、基本概念

什么是进程？

当一个程序开始运行时，它就是一个进程，进程包括运行中的程序和程序所使用到的内存和系统资源。而一个进程又是由多个线程所组成的。

什么是线程？

线程是程序中的一个执行流，每个线程都有自己的专有寄存器(栈指针、程序计数器等)，但代码区是共享的，即不同的线程可以执行同样的函数。

什么是多线程？

多线程是指程序中包含多个执行流，即在一个程序中可以同时运行多个不同的线程来执行不同的任务，也就是说允许单个程序创建多个并行执行的线程来完成各自的任务。

前台线程后台线程？

应用程序的主线程和通过构造一个Thread对象来显式创建的任何线程都默认是前台线程。相反线程池线程默认为后台线程。另外由进入托管执行环境的本机代码创建的任何线程都被标记为后台线程。

在线程的生命周期中，任何时候都可以从前台变为后台，或者从后台变为前台。

前台线程能阻止应用程序的终结。一直到所有的前台线程终止后，CLR才能关闭应用程序（即卸载承载的应用程序域）。

后台线程（有时也叫守护线程）被CLR认为是程序执行中可做出牺牲的途径，即在任何时候（即使这个线程此时正在执行某项工作）都可能被忽略。因此，如果所有的前台线程终止，当应用程序域卸载时，所有的后台线程也会被自动终止。

线程是轻量级进程。一个使用线程的常见实例是现代操作系统中并行编程的实现。使用线程节省了CPU周期的浪费，同时提高了应用程序的效率。

二、线程的生命周期

线程生命周期开始于 `System.Threading.Thread` 类的对象被创建时，结束于线程被终止或完成执行时。

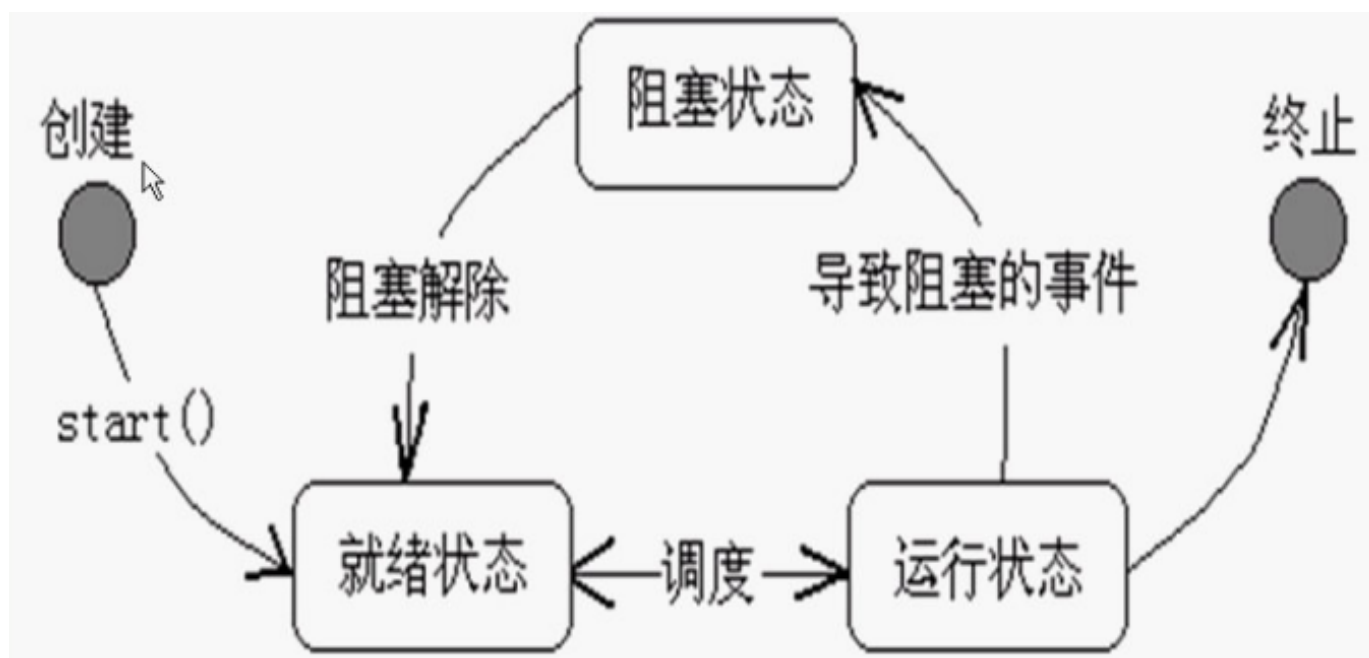
线程生命周期中的各种状态：

未启动状态：当线程实例被创建但 `Start` 方法未被调用时的状况（将该线程标记为可以运行的状态，但具体执行时间由cpu决定。）。

就绪状态：当线程准备好运行并等待 CPU 周期时的状况。

不可运行状态：下面的几种情况下载程是不可运行的：（已经调用 `Sleep` 方法，已经调用 `Wait` 方法，通过 I/O 操作阻塞）

死亡状态：当线程已完成执行或已中止时的状况。



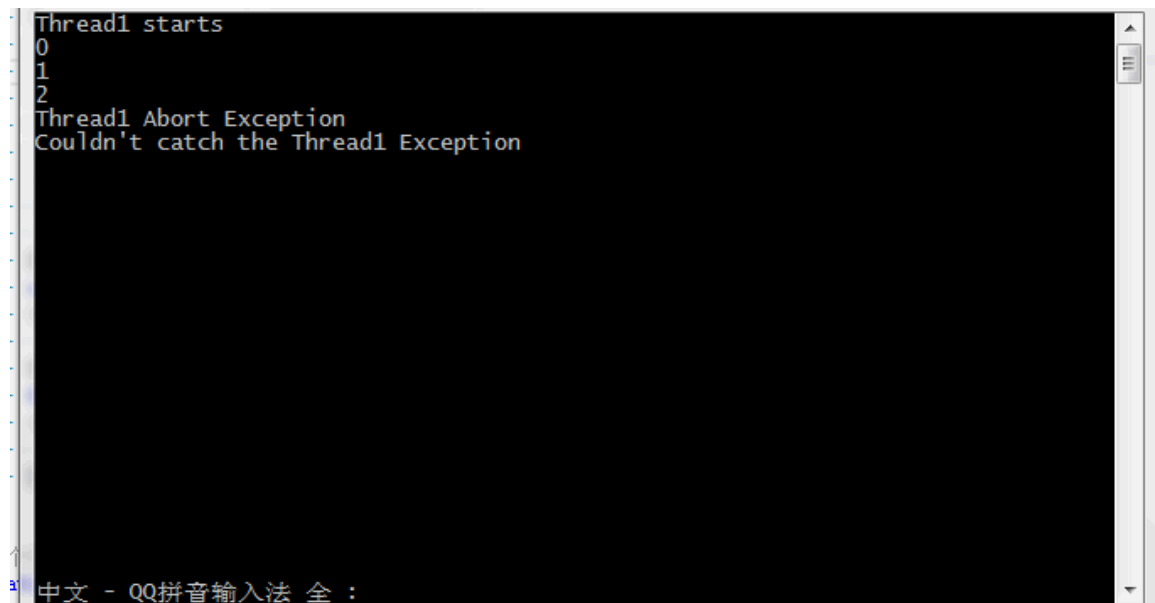
三、线程

1、主线程

进程中第一个被执行的线程称为主线程

C#

销毁代码执行结果：

A screenshot of a C# console application window. The console output shows: "Thread1 starts", "0", "1", "2", "Thread1 Abort Exception", and "Couldn't catch the Thread1 Exception". The window has a title bar and standard Windows controls. At the bottom, there is a text input field with the text "中文 - QQ拼音输入法 全 :".

```
Thread1 starts
0
1
2
Thread1 Abort Exception
Couldn't catch the Thread1 Exception
```

四、线程池

在多线程程序中，线程把大部分的时间花费在等待状态，等待某个事件发生，然后才能给予响应我们一般用ThreadPool（线程池）来解决；线程平时都处于休眠状态，只是周期性地被唤醒我们使用使用Timer（定时器）来解决。

由于线程的创建和销毁需要耗费一定的开销，过多的使用线程会造成内存资源的浪费，出于对性能的考虑，于是引入了线程池的概念。线程池维护一个请求队列，线程池的代码从队列提取任务，然后委派给线程池的一个线程执行，线程执行完不会被立即销毁，这样既可以在后台执行任务，又可以减少线程创建和销毁所带来的开销。线程池线程默认为后台线程。

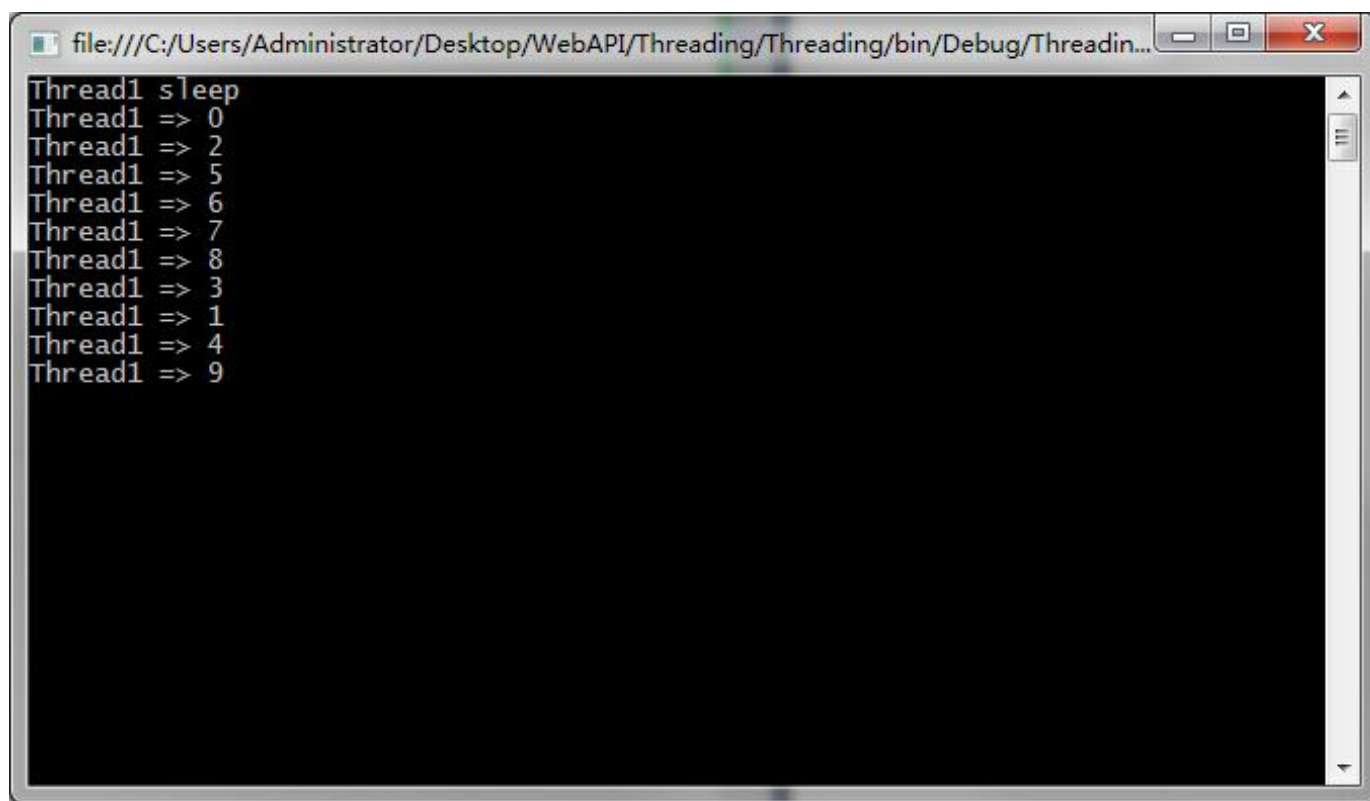
线程池自动管理线程线程的创建和销毁。

代码展示：

```
using System;
using System.Threading;
namespace Threading
{
    class Program
    {
        public static void Thread1(object data)
        {
            Console.WriteLine("Thread1 => {0}", data.ToString());
        }
        static void Main(string[] args)
        {
            //控制线程数大小
            //第一个参数是：线程池中辅助线程的最大数目
            //第二个参数是：线程池中异步 I/O 线程的最大数目
```

```
ThreadPool.SetMaxThreads(3, 3);  
for (int i = 0; i < 10; i++)  
{  
    //ThreadPool是静态类无需实例化,  
    //ThreadPool.QueueUserWorkItem(new WaitCallback(Thread1),  
    i);  
  
    ThreadPool.QueueUserWorkItem(Thread1, i);  
}  
Console.WriteLine("Thread1 sleep");  
Thread.Sleep(100000);  
Console.WriteLine("Thread1 end");  
Console.ReadKey();  
}  
}
```

运行结果:



```
file:///C:/Users/Administrator/Desktop/WebAPI/Threading/Threading/bin/Debug/Threadin...  
Thread1 sleep  
Thread1 => 0  
Thread1 => 2  
Thread1 => 5  
Thread1 => 6  
Thread1 => 7  
Thread1 => 8  
Thread1 => 3  
Thread1 => 1  
Thread1 => 4  
Thread1 => 9
```

但是为什么最开始输出Thread1 sleep? 有时候也会在中途随机输出呢?

其实, 线程池的启动和终止不是我们程序所能控制的, 线程池中的线程执行完之后是没有返回值的, 我们可以用ManualResetEvent通知一个或多个正在等待的线程已发生事件

修改后的代码:

```
using System;  
using System.Threading;  
namespace Threading
```

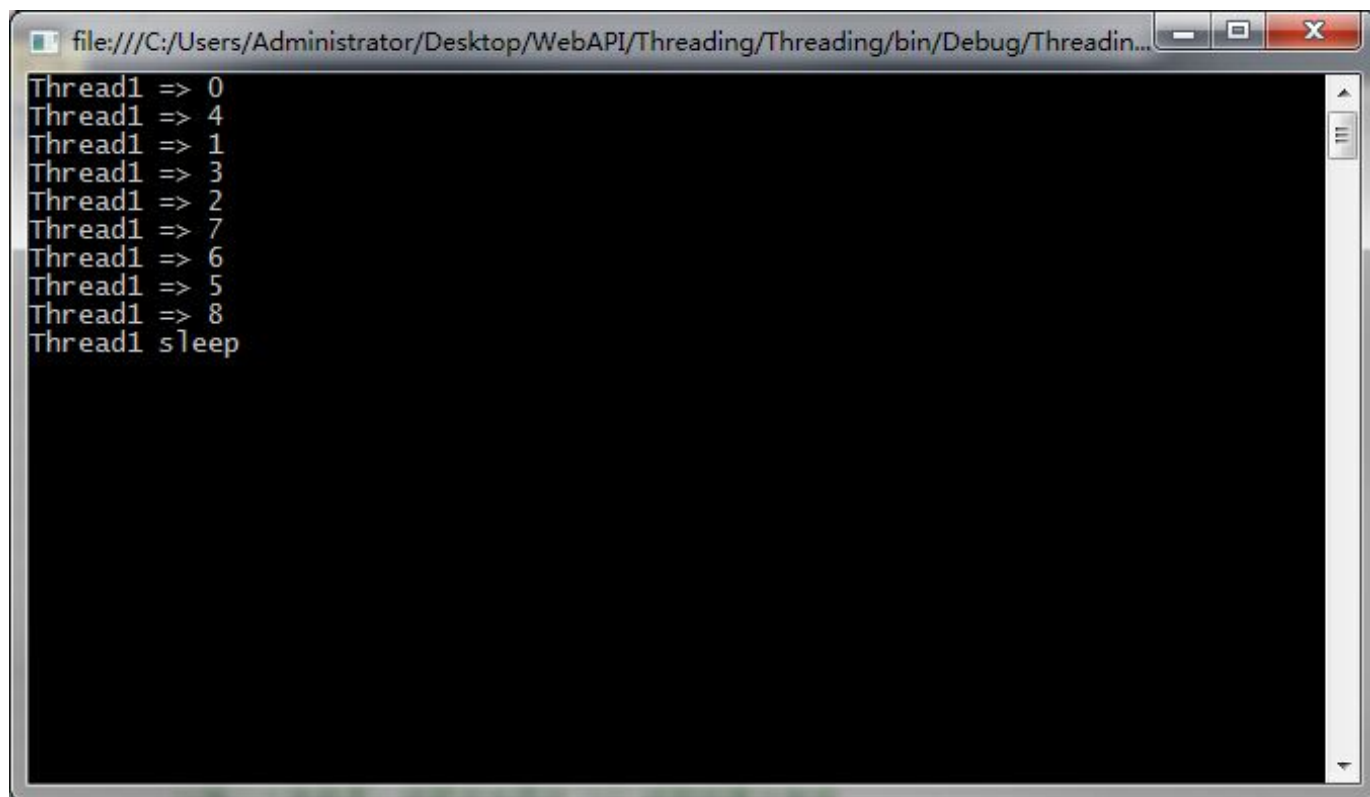
```

{
    class Program
    {
        //新建ManualResetEvent对象并且初始化为无信号状态
        private static ManualResetEvent mre = new ManualResetEvent(false);
        public static void Thread1(object data)
        {
            Console.WriteLine("Thread1 => {0}", data.ToString());
            if (Convert.ToInt32(data) == 9)
            {
                mre.Set();
            }
        }
        static void Main(string[] args)
        {
            //控制线程数大小
            //第一个参数是：线程池中辅助线程的最大数目
            //第二个参数是：线程池中异步 I/O 线程的最大数目
            ThreadPool.SetMaxThreads(3, 3);
            for (int i = 0; i < 10; i++)
            {
                //ThreadPool是静态类无需实例化,
                //ThreadPool.QueueUserWorkItem(new WaitCallback(Thread1),
                i);

                ThreadPool.QueueUserWorkItem(Thread1, i);
            }
            //阻止当前线程，直到当前 WaitHandle 收到信号为止。
            mre.WaitOne(Timeout.Infinite, true);
            Console.WriteLine("Thread1 sleep");
            Thread.Sleep(100000);
            Console.WriteLine("Thread1 end");
            Console.ReadKey();
        }
    }
}

```

输入结果：



```
file:///C:/Users/Administrator/Desktop/WebAPI/Threading/Threading/bin/Debug/Threadin...
Thread1 => 0
Thread1 => 4
Thread1 => 1
Thread1 => 3
Thread1 => 2
Thread1 => 7
Thread1 => 6
Thread1 => 5
Thread1 => 8
Thread1 sleep
```

ok, 搞定。

参考资料：

ThreadPool: <https://msdn.microsoft.com/zh-cn/library/system.threading.threadpool.aspx#Y0>

ManualResetEvent: <https://msdn.microsoft.com/zh-cn/library/system.threading.manualresetevent.aspx>

五、总结

多线程的好处：

可以提高CPU的利用率。在多线程程序中，一个线程必须等待的时候，CPU可以运行其它的线程而不是等待，这样就大大提高了程序的效率。

多线程的不利方面：

线程也是程序，所以线程需要占用内存，线程越多占用内存也越多；

多线程需要协调和管理，所以需要CPU时间跟踪线程；

线程之间对共享资源的访问会相互影响，必须解决竞用共享资源的问题；

线程太多会导致控制太复杂，最终可能造成很多Bug；

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](#)》

2 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