

Thread 类

.NET Framework 4

创建并控制线程，设置其优先级并获取其状态。

继承层次结构

System.Object
System.Runtime.ConstrainedExecution.CriticalFinalizerObject
System.Threading.Thread

命名空间：[System.Threading](#)
程序集：[mscorlib](#) (在 [mscorlib.dll](#) 中)


语法

C#


```
[ClassInterfaceAttribute(ClassInterfaceType.None)]  
[ComVisibleAttribute(true)]  
public sealed class Thread : CriticalFinalizerObject,  
    _Thread
```

Thread 类型公开以下成员。

构造函数

显示: ☒ 继承 ☒ 保护 


	名称	说明
	Thread(ParameterizedThreadStart)	初始化 Thread 类的新实例，指定允许对象在线程启动时传递给线程的委托。
	Thread(ThreadStart)	初始化 Thread 类的新实例。
	Thread(ParameterizedThreadStart, Int32)	初始化 Thread 类的新实例，指定允许对象在线程启动时传递给线程的委托，并指定线程的最大堆栈大小。

	<code>Thread(ThreadStart, Int32)</code>	初始化 <code>Thread</code> 类的新实例，指定线程的最大堆栈大小。
---	---	--

[页首](#)


属性

显示: ☒ 继承 ☒ 保护 

	名称	说明
	<code>ApartmentState</code>	已过时。 获取或设置此线程的单元状态。
	<code>CurrentContext</code>	获取线程正在其中执行的当前上下文。
	<code>CurrentCulture</code>	获取或设置当前线程的区域性。
	<code>CurrentPrincipal</code>	获取或设置线程的当前负责人（对基于角色的安全性而言）。
	<code>CurrentThread</code>	获取当前正在运行的线程。
	<code>CurrentUICulture</code>	获取或设置资源管理器使用的当前区域性以便在运行时查找区域性特定的资源。
	<code>ExecutionContext</code>	获取一个 <code>ExecutionContext</code> 对象，该对象包含有关当前线程的各种上下文的信息。
	<code>IsAlive</code>	获取一个值，该值指示当前线程的执行状态。
	<code>IsBackground</code>	获取或设置一个值，该值指示某个线程是否为后台线程。
	<code>IsThreadPoolThread</code>	获取一个值，该值指示线程是否属于托管线程池。
	<code>ManagedThreadId</code>	获取当前托管线程的唯一标识符。
	<code>Name</code>	获取或设置线程的名称。
	<code>Priority</code>	获取或设置一个值，该值指示线程的调度优先级。
	<code>ThreadState</code>	获取一个值，该值包含当前线程的状态。



[页首](#)









方法

显示: ☒ 继承 ☒ 保护 

	名称	说明
	<code>Abort()</code>	在调用此方法的线程上引发 <code>ThreadAbortException</code> ，以开始终止此线程的过程。调用此方法通常会终止线程。
	<code>Abort(Object)</code>	在调用此方法的线程上引发 <code>ThreadAbortException</code> ，以开始终止此线程并提供有关线程终止的异常信息的过程。调用此方法通常会终止线程。
	<code>AllocateDataSlot</code>	在所有的线程上分配未命名的数据槽。为了获得更好的性能，请改用以 <code>ThreadStaticAttribute</code> 特性标记的字段。
	<code>AllocateNamedDataSlot</code>	在所有线程上分配已命名的数据槽。为了获得更好的性能，请改用以 <code>ThreadStaticAttribute</code> 特性标记的字段。
	<code>BeginCriticalRegion</code>	通知宿主执行将要进入一个代码区域，在该代码区域内线程中止或未经处理的异常的影响可能会危害应用程序域中的其他任务。
	<code>BeginThreadAffinity</code>	通知宿主托管代码将要执行依赖于当前物理操作系统线程的标识的指令。
	<code>DisableComObjectEagerCleanup</code>	对于当前线程关闭运行时可调用包装 (RCW) 的自动清理。
	<code>EndCriticalRegion</code>	通知宿主执行将要进入一个代码区域，在该代码区域内线程中止或未经处理的异常仅影响当前任务。
	<code>EndThreadAffinity</code>	通知宿主托管代码已执行完依赖于当前物理操作系统线程的标识的指令。
	<code>Equals(Object)</code>	确定指定的 <code>Object</code> 是否等于当前的 <code>Object</code> 。（继承自 <code>Object</code> 。）
	<code>Finalize</code>	<p>释放由 <code>CriticalFinalizerObject</code> 类使用的所有资源。（继承自 <code>CriticalFinalizerObject</code>。）</p> <p>在 XNA Framework 3.0 <code>Finalize()</code>。</p> <p>在 <code>Finalize()</code>。</p>
	<code>FreeNamedDataSlot</code>	为进程中的所有线程消除名称与槽之间的关联。为了获得更好的性能，请改用以 <code>ThreadStaticAttribute</code> 特性标记的字段。
	<code>GetApartmentState</code>	返回一个 <code>ApartmentState</code> 值，该值指示单元状态。
	<code>GetCompressedStack</code>	已过时。 返回一个 <code>CompressedStack</code> 对象，该对象可用于捕获当前线程的堆栈。

	GetData	在当前线程的当前域中从当前线程上指定的槽中检索值。 为了获得更好的性能，请改用以 ThreadStaticAttribute 特性标记的字段。
	GetDomain	返回当前线程正在其中运行的当前域。
	GetDomainID	返回唯一的应用程序域标识符。
	GetHashCode	返回当前线程的哈希代码。（重写 Object.GetHashCode() 。） 在 XNA Framework 3.0 GetHashCode() 。 在 GetHashCode() 。
	GetNamedDataSlot	查找已命名的数据槽。 为了获得更好的性能，请改用以 ThreadStaticAttribute 特性标记的字段。
	GetType	获取当前实例的 Type 。（继承自 Object 。）
	Interrupt	中断处于 WaitSleepJoin 线程状态的线程。
	Join()	在继续执行标准的 COM 和 SendMessage 消息泵处理期间，阻塞调用线程，直到某个线程终止为止。
	Join(Int32)	在继续执行标准的 COM 和 SendMessage 消息泵处理期间，阻塞调用线程，直到某个线程终止或经过了指定时间为止。
	Join(TimeSpan)	在继续执行标准的 COM 和 SendMessage 消息泵处理期间，阻塞调用线程，直到某个线程终止或经过了指定时间为止。
	MemberwiseClone	创建当前 Object 的浅表副本。（继承自 Object 。）
	MemoryBarrier	按如下方式同步内存访问：执行当前线程的处理器在对指令重新排序时，不能采用先执行 MemoryBarrier 调用之后的内存访问，再执行 MemoryBarrier 调用之前的内存访问的方式。
	ResetAbort	取消为当前线程请求的 Abort 。
	Resume	已过时。 继续已挂起的线程。
	SetApartmentState	在线程启动前设置其单元状态。
	SetCompressedStack	已过时。 对当前线程应用捕获的 CompressedStack 。
	SetData	在当前正在运行的线程上为此线程的当前域在指定槽中设置数据。 为了获得更好的性能，请改用以 ThreadStaticAttribute 特性标记的字段。
	SetProcessorAffinity	在用于 Xbox 360 的 .NET Compact Framework 中，

		为托管线程设置处理器关联。 运行线程的处理器取决于处理器关联。
	<code>Sleep(Int32)</code>	将当前线程挂起指定的时间。
	<code>Sleep(TimeSpan)</code>	将当前线程阻塞指定的时间。
	<code>SpinWait</code>	导致线程等待由 <code>iterations</code> 参数定义的时间量。
	<code>Start()</code>	导致操作系统将当前实例的状态更改为 <code>ThreadState.Running</code> 。
	<code>Start(Object)</code>	使操作系统将当前实例的状态更改为 <code>ThreadState.Running</code> ，并选择提供包含线程执行的方法要使用的数据的对象。
	<code>Suspend</code>	已过时。 挂起线程，或者如果线程已挂起，则不起作用。
	<code>ToString</code>	返回表示当前对象的字符串。（继承自 <code>Object</code> 。）
	<code>TrySetApartmentState</code>	在线程启动前设置其单元状态。
	<code>VolatileRead(Byte)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(Double)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(Int16)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(Int32)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(Int64)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(IntPtr)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(Object)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(SByte)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(Single)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(UInt16)</code>	读取字段值。 无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。

	<code>VolatileRead(UInt32)</code>	读取字段值。无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(UInt64)</code>	读取字段值。无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileRead(UIntPtr)</code>	读取字段值。无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。
	<code>VolatileWrite(Byte, Byte)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(Double, Double)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(Int16, Int16)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(Int32, Int32)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(Int64, Int64)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(IntPtr, IntPtr)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(Object, Object)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(SByte, SByte)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(Single, Single)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(UInt16, UInt16)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(UInt32, UInt32)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(UInt64, UInt64)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>VolatileWrite(UIntPtr, UIntPtr)</code>	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。
	<code>Yield</code>	导致调用线程执行准备好在当前处理器上运行的另一个线程。 由操作系统选择要执行的线程。

显式接口实现

显示: ☒ 继承 ☒ 保护 

	名称	说明
	_Thread.GetIDsOfNames	将一组名称映射为对应的一组调度标识符。
	_Thread.GetTypeInfo	检索对象的类型信息，然后可以使用该信息获取接口的类型信息。
	_Thread.GetTypeInfoCount	检索对象提供的类型信息接口的数量（0 或 1）。
	_Thread.Invoke	提供对某一对象公开的属性和方法的访问。


[页首](#)

备注

一个进程可以创建一个或多个线程以执行与该进程关联的部分程序代码。使用 [ThreadStart](#) 委托或 [ParameterizedThreadStart](#) 委托指定由线程执行的程序代码。使用 [ParameterizedThreadStart](#) 委托可以将数据传递到线程过程。


在线程存在期间，它总是处于由 [ThreadState](#) 定义的一个或多个状态中。可以为线程请求由 [ThreadPriority](#) 定义的调度优先级，但不能保证操作系统会接受该优先级。

[GetHashCode](#) 提供托管线程的标识。在线程的生存期内，无论获取该值的应用程序域如何，它都不会和任何来自其他线程的值冲突。

 注意

因为非托管宿主可以控制托管线程和非托管线程之间的关系，所以操作系统 ThreadId 与托管线程之间没有固定的关系。特别是，复杂的宿主可以使用 CLR 承载 API 针对相同的操作系统线程调度很多托管线程，或者在不同的操作系统线程之间移动托管线程。

一旦启动线程，便不必保留对 Thread 对象的引用。线程将继续执行，直到该线程过程完成。

 重要事项

从 .NET Framework 4 版 开始，更改某些线程构造函数的行为：只有完全受信任的代码可以将最大堆栈大小设置为大于默认堆栈大小（1 MB）的值。如果在部分信任的情况下运行代码时指定更大的值，则更大的值将被忽略，并且使用默认堆栈大小。不引发异常。任何信任级别的代码可以将最大堆栈大小设置为小于默认堆栈大小的值。

示例

下面的代码示例说明简单的线程处理功能。

C#

```
using System;
using System.Threading;

// Simple threading scenario: Start a static method running
// on a second thread.
public class ThreadExample {
    // The ThreadProc method is called when the thread starts.
    // It loops ten times, writing to the console and yielding
    // the rest of its time slice each time, and then ends.
    public static void ThreadProc() {
        for (int i = 0; i < 10; i++) {
            Console.WriteLine("ThreadProc: {0}", i);
            // Yield the rest of the time slice.
            Thread.Sleep(0);
        }
    }

    public static void Main() {
        Console.WriteLine("Main thread: Start a second thread.");
        // The constructor for the Thread class requires a ThreadStart
        // delegate that represents the method to be executed on the
        // thread. C# simplifies the creation of this delegate.
        Thread t = new Thread(new ThreadStart(ThreadProc));

        // Start ThreadProc. Note that on a uniprocessor, the new
        // thread does not get any processor time until the main thread
        // is preempted or yields. Uncomment the Thread.Sleep that
        // follows t.Start() to see the difference.
        t.Start();
        //Thread.Sleep(0);

        for (int i = 0; i < 4; i++) {
            Console.WriteLine("Main thread: Do some work.");
            Thread.Sleep(0);
        }

        Console.WriteLine("Main thread: Call Join(), to wait until ThreadProc ends.");
        t.Join();
        Console.WriteLine("Main thread: ThreadProc.Join has returned. Press Enter to end pr");
        Console.ReadLine();
    }
}
```


此代码产生的输出类似如下内容：

```
[VB, C++, C#]
Main thread: Start a second thread.
Main thread: Do some work.
ThreadProc: 0
Main thread: Do some work.
ThreadProc: 1
Main thread: Do some work.
ThreadProc: 2
Main thread: Do some work.
ThreadProc: 3
Main thread: Call Join(), to wait until ThreadProc ends.
ThreadProc: 4
ThreadProc: 5
ThreadProc: 6
ThreadProc: 7
ThreadProc: 8
ThreadProc: 9
Main thread: ThreadProc.Join has returned. Press Enter to end program.
```

版本信息

.NET Framework

受以下版本支持：4、3.5、3.0、2.0、1.1、1.0

.NET Framework Client Profile

受以下版本支持：4、3.5 SP1

受以下版本支持：

平台

Windows 7, Windows Vista SP1 或更高版本, Windows XP SP3, Windows XP SP2 x64 Edition, Windows Server 2008 (不支持服务器核心), Windows Server 2008 R2 (支持 SP1 或更高版本的服务器核心), Windows Server 2003 SP2

.NET Framework 并不是对每个平台的所有版本都提供支持。有关支持的版本的列表，请参见[.NET Framework 系统要求](#)。

线程安全

此类型是线程安全的。

请参见

参考

[System.Threading](#) 命名空间

其他资源

[线程与线程处理](#)

[使用线程和线程处理](#)

社区附加资源

© 2015 Microsoft