**线程生命周期**

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。
* **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**：下面的几种情况下线程是不可运行的：

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。

**主线程**

进程中第一个被执行的线程称为**主线程**。

您可以使用 Thread 类的**CurrentThread** 属性访问线程。

下面的程序演示了主线程的执行：

using System;using System.Threading;

namespace MultithreadingApplication{

class MainThreadProgram

{

static void Main(string[] args)

{

Thread th = Thread.CurrentThread;

th.Name = "MainThread";

Console.WriteLine("This is {0}", th.Name);

Console.ReadKey();

}

}}

当上面的代码被编译和执行时，它会产生下列结果：

This is MainThread

## Thread 类常用的属性和方法

下表列出了 **Thread** 类的一些常用的 **属性**：

|  |  |
| --- | --- |
| **属性** | **描述** |
| CurrentContext | 获取线程正在其中执行的当前上下文。 |
| CurrentCulture | 获取或设置当前线程的区域性。 |
| CurrentPrinciple | 获取或设置线程的当前负责人（对基于角色的安全性而言）。 |
| CurrentThread | 获取当前正在运行的线程。 |
| CurrentUICulture | 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。 |
| IsAlive | 获取一个值，该值指示当前线程的执行状态。 |
| IsBackground | 获取或设置一个值，该值指示某个线程是否为后台线程。 |
| IsThreadPoolThread | 获取一个值，该值指示线程是否属于托管线程池。 |
| ManagedThreadId | 获取当前托管线程的唯一标识符。 |
| Name | 获取或设置线程的名称。 |
| Priority | 获取或设置一个值，该值指示线程的调度优先级。 |
| ThreadState | 获取一个值，该值包含当前线程的状态。 |

下表列出了 **Thread** 类的一些常用的 **方法**：

|  |  |
| --- | --- |
| **序号** | **方法名 & 描述** |
| 1 | **public void Abort()** 引发 ThreadAbortException，终止当前线程 |
| 10 | **public static AppDomain GetDomain()** 返回当前线程所在的应用程序域。 |
| 11 | **public static AppDomain GetDomainID()** 返回唯一的应用程序域标识符。 |
| 13 | **public void Interrupt()** 中断处于 WaitSleepJoin 线程状态的线程。 |
| 18 | **public void Start()** 开始一个线程。 |
| 19 | **public static void Sleep( int millisecondsTimeout )** 让线程暂停一段时间。 |
| 20 | **public static void SpinWait( int iterations )** 导致线程等待由 iterations 参数定义的时间量。 |

**创建线程**

线程是通过扩展 Thread 类创建的。扩展的 Thread 类调用 **Start()** 方法来开始子线程的执行。

public static void CallToChildThread()

{

Console.WriteLine("Child thread starts");

}

static void Main(string[] args)

{

ThreadStart childref = new ThreadStart(CallToChildThread);

Thread childThread = new Thread(childref);

childThread.Start();

Console.ReadKey();

}

**管理线程**

Thread 类提供了各种管理线程的方法。

下面的实例演示了 **sleep()** 方法的使用，用于暂停线程。

public static void CallToChildThread()

{

Console.WriteLine("Child thread starts");

// 线程暂停 5000 毫秒

int sleepfor = 5000;

Console.WriteLine("Child Thread Paused for {0} seconds",

sleepfor / 1000);

Thread.Sleep(sleepfor);

Console.WriteLine("Child thread resumes");

}

static void Main(string[] args)

{

ThreadStart childref = new ThreadStart(CallToChildThread);

Console.WriteLine("In Main: Creating the Child thread");

Thread childThread = new Thread(childref);

childThread.Start();

Console.ReadKey();

}

}

**销毁线程**

**Abort()** 方法用于销毁线程。

通过抛出 **threadabortexception** 异常中止线程。这个异常不能被捕获

下面的程序说明了这点：

class ThreadCreationProgram

{

public static void CallToChildThread()

{

try

{

Console.WriteLine("Child thread starts");

// 计数到 10

for (int counter = 0; counter <= 10; counter++)

{

Thread.Sleep(500);

Console.WriteLine(counter);

}

Console.WriteLine("Child Thread Completed");

}

catch (ThreadAbortException e)

{

Console.WriteLine("Thread Abort Exception");

}

finally

{

Console.WriteLine("Couldn't catch the Thread Exception");

}

}

static void Main(string[] args)

{

ThreadStart childref = new ThreadStart(CallToChildThread);

Console.WriteLine("In Main: Creating the Child thread");

Thread childThread = new Thread(childref);

childThread.Start();

// 停止主线程一段时间

Thread.Sleep(2000);

// 现在中止子线程

Console.WriteLine("In Main: Aborting the Child thread");

childThread.Abort();

Console.ReadKey();

}

}