# 多线程

## 概念

windows系统是一个多线程的操作系统。一个程序至少有一个进程,一个进程至少有一个线程。进程是线程的容器，一个C#客户端程序开始于一个单独的线程，CLR(公共语言运行库)为该进程创建了一个线程，该线程称为主线程。例如当我们创建一个C#控制台程序，程序的入口是Main()函数，Main()函数是始于一个主线程的。它的功能主要 是产生新的线程和执行程序。C#是一门支持多线程的编程语言，通过Thread类创建子线程，引入using System.Threading命名空间。

### 优点

1、 多线程可以提高CPU的利用率，因为当一个线程处于等待状态的时候，CPU会去执行另外的线程

2、 提高了CPU的利用率，就可以直接提高程序的整体执行速度

### 缺点

1、线程开的越多，内存占用越大

2、协调和管理代码的难度加大，需要CPU时间跟踪线程

3、线程之间对资源的共享可能会产生可不遇知的问题

### 前台线程和后台线程

C#中的线程分为前台线程和后台线程，线程创建时不做设置默认是前台线程。即线程属性IsBackground=false。

Thread.IsBackground = false;//false:设置为前台线程，系统默认为前台线程。

这两者的区别就是：应用程序必须运行完所有的前台线程才可以退出；而对于后台线程，应用程序则可以不考虑其是否已经运行完毕而直接退出，所有的后台线程在应用程序退出时都会自动结束。一般后台线程用于处理时间较短的任务，如在一个Web服务器中可以利用后台线程来处理客户端发过来的请求信息。而前台线程一般用于处理需要长时间等待的任务，如在Web服务器中的监听客户端请求的程序。

线程是寄托在进程上的，进程都结束了，线程也就不复存在了！

只要有一个前台线程未退出，进程就不会终止！即说的就是程序不会关闭！（即在资源管理器中可以看到进程未结束。）

## 多线程的创建

### 线程的生命周期

class Program

{

//我们的控制台程序入口是main函数。它所在的线程即是主线程

static void Main(string[] args)

{

Thread thread = new Thread(ThreadMethod); //执行的必须是无返回值的方法

thread.Name = "子线程";

//thread.Start("王建"); //在此方法内传递参数，类型为object，发送和接收涉及到拆装箱操作

thread.Start();

Console.ReadKey();

}

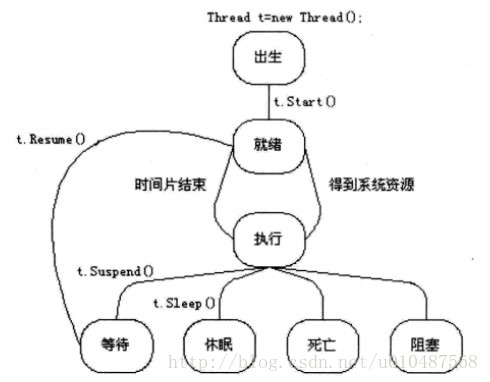
public static void ThreadMethod(object parameter) //方法内可以有参数，也可以没有参数

{

Console.WriteLine("{0}开始执行。", Thread.CurrentThread.Name);

}

}



### 线程的属性和基本操作



#### ThreadState



#### Abort()方法

Abort()方法用来终止线程，调用此方法强制停止正在执行的线程，它会抛出一个ThreadAbortException异常从而导致目标线程的终止。

Thread.Abort方法应谨慎使用。 尤其是当您调用它来中止当前线程以外的线程，您不知道哪些代码已执行或失败时要执行**[ThreadAbortException](https://docs.microsoft.com/zh-cn/dotnet/api/system.threading.threadabortexception?view=netframework-4.8)**引发，也不能为特定的应用程序的状态或任何应用程序和用户的状态负责保留。 例如，调用Thread.Abort可能会阻止执行静态构造函数或阻止非托管资源的释放。(FileStream)

下面我们来解释一下Abort方法是如何工作的。因为公用语言运行时管理了所有的托管的线程，同样它能在每个线程内抛出异常。Abort方法能在目标线程中抛出一个ThreadAbortException异常从而导致目标线程的终止。不过Abort方法被调用后，目标线程可能并不是马上就终止了。因为只要目标线程正在调用非托管的代码而且还没有返回的话，该线程就不会立即终止。而如果目标线程在调用非托管的代码而且陷入了一个死循环的话，该目标线程就根本不会终止。不过这种情况只是一些特例，更多的情况是目标线程在调用托管的代码，一旦Abort被调用那么该线程就立即终止了。

#### ResetAbort()方法

Abort方法可以通过跑出ThreadAbortException异常中止线程，而使用ResetAbort方法可以取消中止线程的操作

#### Sleep()方法

Sleep()方法调已阻塞线程，是当前线程进入休眠状态，在休眠过程中占用系统内存但是不占用系统时间，当休眠期过后，继续执行

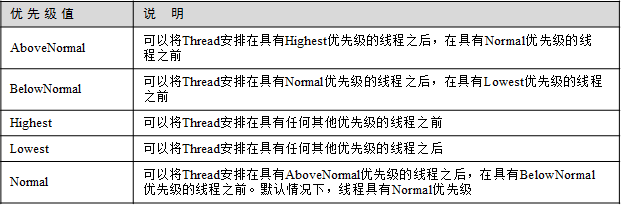
#### join()方法

Join方法主要是用来阻塞调用线程，直到某个线程终止或经过了指定时间为止。官方的解释比较乏味，通俗的说就是创建一个子线程，给它加了这个方法，其它线程就会暂停执行，直到这个线程执行完为止才去执行（包括主线程）。

#### Suspent()和Resume()方法

其实在C# 2.0以后， Suspent()和Resume()方法已经过时了。suspend()方法容易发生死锁。调用suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被”挂起”的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。所以不应该使用suspend()。

### 线程的优先级



当有多个线程**同时**处于可执行状态，系统优先执行优先级较高的线程，但这只意味着优先级较高的线程占有更多的CPU时间，并不意味着一定要先执行完优先级较高的线程，才会执行优先级较低的线程。

优先级越高表示CPU分配给该线程的时间片越多,执行时间就多

优先级越低表示CPU分配给该线程的时间片越少,执行时间就少

## 线程同步

### 什么是线程安全

线程安全是指在当一个线程访问该类的某个数据时，进行保护，其他线程不能进行访问直到该线程读取完，其他线程才可使用。不会出现数据不一致或者数据污染。

线程有可能和其他线程共享一些资源，比如，内存，文件，数据库等。当多个线程同时读写同一份共享资源的时候，可能会引起冲突。这时候，我们需要引入线程“同步”机制，即各位线程之间要有个先来后到，不能一窝蜂挤上去抢作一团。线程同步的真实意思和字面意思恰好相反。线程同步的真实意思，其实是“排队”：几个线程之间要排队，一个一个对共享资源进行操作，而不是同时进行操作。

#### 通过单例模式说明线程同步

单例模式就是保证在整个应用程序的生命周期中，在任何时刻，被指定的类只有一个实例，并为客户程序提供一个获取该实例的全局访问点。但上面代码有一个明显的问题，那就是假如两个线程同时去获取这个对象实例，就会发生不可描述的事情。

public class Singleton

{

private static Singleton instance;

private Singleton() //私有函数，防止实例

{

}

public static Singleton GetInstance()

{

if (instance == null)

{

instance = new Singleton();

}

return instance;

}

}

加了一个 lock(obj)代码块。这样就能够实现同步了

public class Singleton

{

private static Singleton instance;

private static object obj=new object();

private Singleton()　　　　　　　　//私有化构造函数

{

}

public static Singleton GetInstance()

{

if(instance==null)

{

lock(obj) //通过Lock关键字实现同步

{

if(instance==null)

{

instance=new Singleton();

}

}

}

return instance;

}

}

#### 使用Lock关键字实现线程同步

lock(this) 锁定 当前实例对象，如果有多个类实例的话，lock锁定的只是当前类实例，对其它类实例无影响。所有不推荐使用。

lock(typeof(Model))锁定的是model类的所有实例。 这里的Model是指某个类名。

lock(obj)锁定的对象是全局的私有化静态变量。外部无法对该变量进行访问。

lock 确保当一个线程位于代码的临界区时，另一个线程不进入临界区。如果其他线程试图进入锁定的代码，则它将一直等待（即被阻止），直到该对象被释放。

## 参考资料：

https://www.cnblogs.com/mq0036/p/6984508.html