.NET Framework定义了两种类型的线程：前台线程和后台线程，默认创建的为前台线程。前台线程与后台线程的唯一区别是：当进程的所有前台线程停止时，后台线程将自动终止。

Thread类：不能继承。

线程的入口函数必须具有void返回类型，并且不能带有任何参数。当线程的入口函数返回时，线程自动停止。

例：程序thread\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 多线程编程

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace thread\_test1

{

class MyThread

{

public int count;

public Thread thread;

public MyThread(string name)

{

count = 0;

thread = new Thread(this.RunTest);

thread.Name = name;

thread.Start();

}

// 线程入口函数

void RunTest()

{

Console.WriteLine(thread.Name + " starting.");

do {

Thread.Sleep(500);

Console.WriteLine("In " + thread.Name + ", Count is " + count);

++count;

} while(count < 10);

}

}

class ThreadTest

{

static void Main(string[] args)

{

Console.WriteLine("Main thread starting.");

MyThread thread = new MyThread("Child #1");

do

{

Console.Write(".");

Thread.Sleep(100);

} while (thread.count != 10);

Console.WriteLine("Main thread ending.");

}

}

}

输出结果：

Main thread starting.

.Child #1 starting.

....In Child #1, Count is 0

.....In Child #1, Count is 1

.....In Child #1, Count is 2

.....In Child #1, Count is 3

.....In Child #1, Count is 4

.....In Child #1, Count is 5

.....In Child #1, Count is 6

.....In Child #1, Count is 7

.....In Child #1, Count is 8

.....In Child #1, Count is 9

Main thread ending.

Thread类确定线程结束：

IsAlive属性判断线程是否在运行

例：程序thread\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 确定线程停止

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace thread\_test2

{

class MyThread

{

public int count;

public Thread thread;

public MyThread(string name)

{

count = 0;

thread = new Thread(this.RunTest);

thread.Name = name;

thread.Start();

}

// 线程入口函数

void RunTest()

{

Console.WriteLine(thread.Name + " starting.");

do

{

Thread.Sleep(500);

Console.WriteLine("In " + thread.Name + ", Count is " + count);

++count;

} while (count < 10);

}

}

class ThreadTest

{

static void Main(string[] args)

{

Console.WriteLine("Main thread starting.");

// 构造3个线程

MyThread thread1 = new MyThread("Child #1");

MyThread thread2 = new MyThread("Child #2");

MyThread thread3 = new MyThread("Child #3");

// IsAlive只读属性判断线程是否停止

do {

Console.Write(".");

Thread.Sleep(100);

} while (thread1.thread.IsAlive

&& thread2.thread.IsAlive

&& thread3.thread.IsAlive);

Console.WriteLine("Main thread ending.");

}

}

}

Join()方法判断线程是否停止：Join()会阻塞主调线程，直到指定的线程“连接”它，即指定的线程返回主调线程。

例：程序thread\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// Join方法判断线程是否停止

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace thread\_test3

{

class MyThread

{

public int count;

public Thread thread;

public MyThread(string name)

{

count = 0;

thread = new Thread(this.RunTest);

thread.Name = name;

thread.Start();

}

// 线程入口函数

void RunTest()

{

Console.WriteLine(thread.Name + " starting.");

do

{

Thread.Sleep(500);

Console.WriteLine("In " + thread.Name + ", Count is " + count);

++count;

} while (count < 10);

}

}

class ThreadTest

{

static void Main(string[] args)

{

Console.WriteLine("Main thread starting.");

MyThread thread1 = new MyThread("Child #1");

MyThread thread2 = new MyThread("Child #2");

MyThread thread3 = new MyThread("Child #3");

// Join()函数判断线程是否停止

thread1.thread.Join();

Console.WriteLine("Child #1 joined.");

thread2.thread.Join();

Console.WriteLine("Child #2 joined.");

thread3.thread.Join();

Console.WriteLine("Child #3 joined.");

Console.WriteLine("Main thread ending.");

}

}

}