任务：任务一旦完成就不能重新启动它。因此，除非重新创建任务，否则没有方法重新运行任务。

任务ID：Task的属性Id。Id为只读属性，是唯一的，无序的。

CurrentId属性获取当前执行的任务的ID。

例：程序task\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 任务ID

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*;

using *System*.*Threading*.*Tasks*;

namespace task\_test2

{

class TaskTest

{

// 任务函数

static void MyTask()

{

*Console*.*WriteLine*("MyTask() #" + *Task*.*CurrentId* + " starting.");

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

{

*Thread*.*Sleep*(500);

*Console*.*WriteLine*("In MyTask() #" + *Task*.*CurrentId* + ", count is " + i);

}

*Console*.*WriteLine*("MyTask #" + *Task*.*CurrentId* + " terminating");

}

static void Main(string[] args)

{

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

*Task* task1 = new *Task*(MyTask);

*Task* task2 = new *Task*(MyTask);

task1.*Start*();

task2.*Start*();

*Console*.*WriteLine*("Task ID for task1 is " + task1.*Id*);

*Console*.*WriteLine*("Task ID for task2 is " + task2.*Id*);

// 任务执行期间，保持Main()运行不退出

for (int i = 0; i < 60; ++i)

{

*Console*.*Write*(".");

*Thread*.*Sleep*(100);

}

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

}

}

}

输出结果：

MyTask() #2 starting.

.....In MyTask() #2, count is 0

In MyTask() #1, count is 0

....In MyTask() #2, count is 1

.In MyTask() #1, count is 1

....In MyTask() #2, count is 2

.In MyTask() #1, count is 2

....In MyTask() #2, count is 3

.In MyTask() #1, count is 3

....In MyTask() #2, count is 4

.In MyTask() #1, count is 4

....In MyTask() #2, count is 5

In MyTask() #1, count is 5

.....In MyTask() #2, count is 6

In MyTask() #1, count is 6

.....In MyTask() #2, count is 7

In MyTask() #1, count is 7

.....In MyTask() #2, count is 8

In MyTask() #1, count is 8

.....In MyTask() #1, count is 9

MyTask #1 terminating

.In MyTask() #2, count is 9

MyTask #2 terminating

.........Main thread ending.

从结果中可以看出，主线程，task1和task2是并行执行，互不干扰。

任务的等待

例：程序task\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 任务的等待

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace task\_test3

{

class TaskTestcs

{

// 任务函数

static void MyTask()

{

Console.WriteLine("MyTask() #" + Task.CurrentId + " starting.");

for (int i = 0; i < 5; ++i)

{

Thread.Sleep(500);

Console.WriteLine("In MyTask() #" + Task.CurrentId + ", count is " + i);

}

Console.WriteLine("MyTask #" + Task.CurrentId + " terminating");

}

static void Main(string[] args)

{

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

Task task1 = new Task(MyTask);

Task task2 = new Task(MyTask);

// 运行任务

task1.Start();

task2.Start();

Console.WriteLine("Task ID for task1 is " + task1.Id);

Console.WriteLine("Task ID for task2 is " + task2.Id);

// 暂停Main()直到task1和task2完成

task1.Wait();

task2.Wait();

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

}

}

}

输出结果：

Main thread starting.

Task ID for task1 is 1

Task ID for task2 is 2

MyTask() #2 starting.

MyTask() #1 starting.

In MyTask() #2, count is 0

In MyTask() #1, count is 0

In MyTask() #2, count is 1

In MyTask() #1, count is 1

In MyTask() #2, count is 2

In MyTask() #1, count is 2

In MyTask() #2, count is 3

In MyTask() #1, count is 3

In MyTask() #2, count is 4

MyTask #2 terminating

In MyTask() #1, count is 4

MyTask #1 terminating

Main thread ending.

使用TaskFactory类启动任务，将Lambda表达式用作任务

当方法只是作为单一用途的任务时，Lambda表达式就特别有用。

例：程序task\_test4

// Copyright 2016.刘珅珅

// author：刘珅珅

// TaskFactory与Lambda表达式

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace task\_test4

{

class TaskTest

{

static void Main(string[] args)

{

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

// 使用TaskFactory任务工厂启动任务

// 使用Lambda表达式定义一个任务入口方法

Task task = Task.Factory.StartNew(() =>

{

Console.WriteLine("Task starting.");

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

{

Thread.Sleep(500);

Console.WriteLine("Task count is " + i);

}

Console.WriteLine("Task terminating");

});

// 等待直到任务完成

task.Wait();

// 任务的清除

// Dispose()方法必须在Wait()或类似的表示

// 任务完成的方法调用完成之后才能调用

// Dispose()方法主要用于创建并放弃许多任务的程序

task.Dispose();

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

}

}

}

输出结果：

Main thread starting.

Task starting.

Task count is 0

Task count is 1

Task count is 2

Task count is 3

Task count is 4

Task count is 5

Task count is 6

Task count is 7

Task count is 8

Task count is 9

Task terminating

Main thread ending.

任务延续：在一个任务完成时自动开始的任务

public Task ContinueWith(Action<Task> continuationAction);

其中，continuationAction指定在主调任务完成后将运行的任务。这个委托有一个Task类型的参数，其使用的Action委托版本为：

public delegate void Action<in T>(T obj) // 是一个逆变类型的委托