Torres Godoy Carlos Emmanuel

Programación III (visual)

04/11/19

**Eventos y delegados**

**Delegados**

Un delegado es una interfaz que contiene un solo método. Se especifica la firma de un método, y cuando tiene la instancia requerida para trabajar con él se llama como un simple método.

Los delegados se declaran con la palabra “delegado”. Estos se utilizan de manera única o anidados dentro de una clase.

namespace DelegateArticle

{

public delegate string FirstDelegate (int x);

public class Sample

{

public delegate void SecondDelegate (char a, char b);

}

}

**Creación de instancias**

Las siguientes dos expresiones de creación son equivalentes, donde InstanceMethod es un método de instancia en la clase que contiene la expresión de creación (o una clase base). El objetivo es "this".

FirstDelegate d1 = new FirstDelegate(InstanceMethod);

FirstDelegate d2 = new FirstDelegate(this.InstanceMethod);

Aquí creamos una instancia de delegado que hace referencia al mismo método como los dos primeros ejemplos, pero con un objetivo diferente.

FirstDelegate d3 = new FirstDelegate(anotherInstance.InstanceMethod);

Esta instancia delegada usa un método de instancia en una clase diferente, especificando el objetivo para llamar al método.

FirstDelegate d4 = new FirstDelegate(instanceOfOtherClass.OtherInstanceMethod);

Esta instancia de delegado utiliza un método estático en esta clase que contiene la expresión de creación (o una clase base).

FirstDelegate d5 = new FirstDelegate(StaticMethod);

Esta instancia de delegado usa un método estático en una clase diferente.

FirstDelegate d6 = new FirstDelegate(OtherClass.OtherStaticMethod);

**Llamar delegados**

Las instancias de los delegados se llaman como si fueran métodos.

string result = d1(10);

Se llaman a través de objetos, si hay algún método se regresa el resultado.

using System;

public delegate string FirstDelegate (int x);

class DelegateTest

{

string name;

static void Main()

{

FirstDelegate d1 = new FirstDelegate(DelegateTest.StaticMethod);

DelegateTest instance = new DelegateTest();

instance.name = "My instance";

FirstDelegate d2 = new FirstDelegate(instance.InstanceMethod);

Console.WriteLine (d1(10)); // Writes out "Static method: 10"

Console.WriteLine (d2(5)); // Writes out "My instance: 5"

}

static string StaticMethod (int i)

{

return string.Format ("Static method: {0}", i);

}

string InstanceMethod (int i)

{

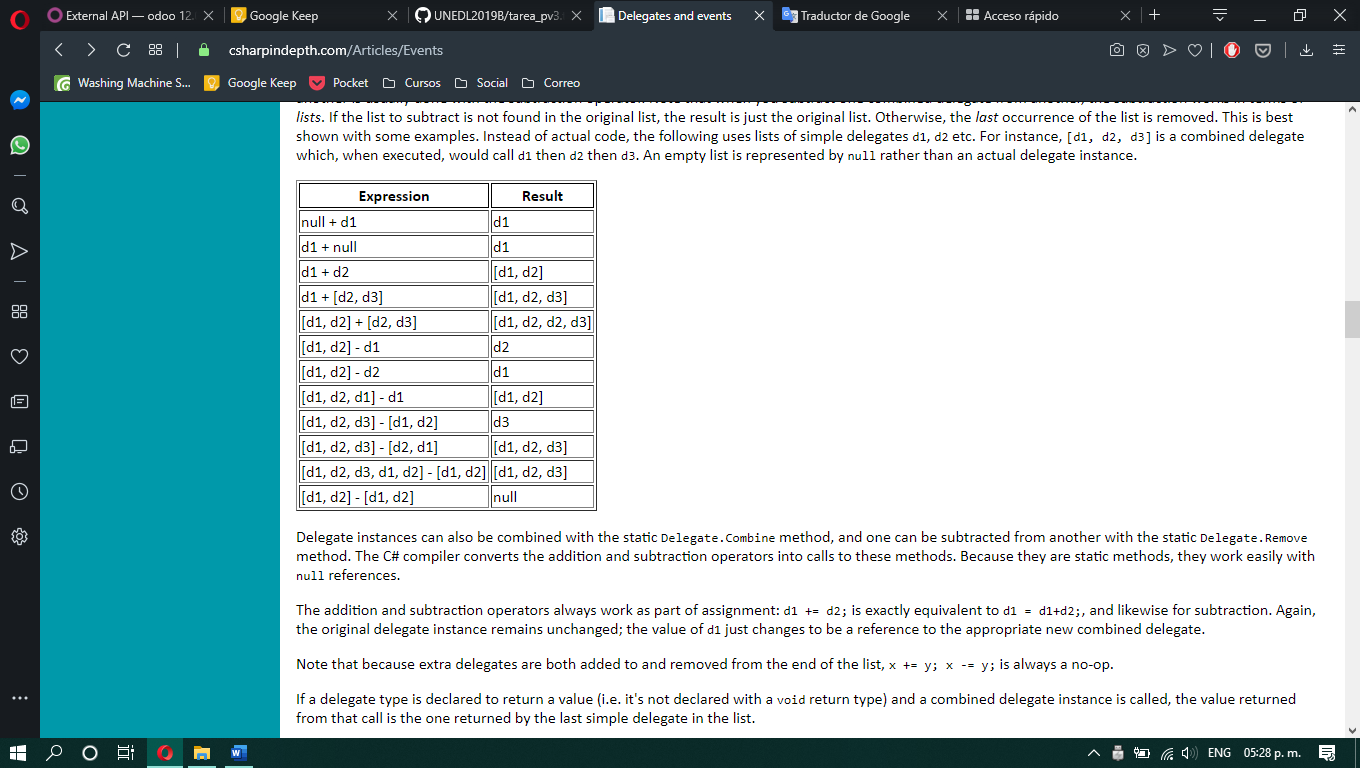
return string.Format ("{0}: {1}", name, i);

}

}

Delegados combinados

Los delegados se combinan para que, al momento de llamar al delegado, se llame a una lista completa de métodos. Potencialmente con diferentes objetos.



El compilador de C # convierte los operadores de suma y resta en llamadas a estos métodos. Debido a que son métodos estáticos, funcionan fácilmente con referencias nulas.

Los operadores de suma y resta siempre funcionan como parte de la asignación: d1 + = d2; es equivalente a d1 = d1 + d2; y también para resta. Nuevamente, la instancia de delegado original permanece sin cambios; el valor de d1 simplemente cambia para ser una referencia al nuevo delegado combinado apropiado.

Tenga en cuenta que debido a que se agregan y eliminan delegados adicionales al final de la lista, x + = y; x - = y; siempre es un no-op.

**Eventos**

Los eventos no son instancias delegadas.

Una forma más fácil de entender los eventos es pensar en ellos un poco como propiedades. Si bien las propiedades parecen ser campos, definitivamente no lo son, y puede escribir propiedades que no usan campos en absoluto. Del mismo modo, si bien los eventos parecen instancias delegadas en términos de la forma en que expresa las operaciones de agregar y quitar, no lo son.

Los eventos son pares de métodos, adecuadamente decorados en IL para unirlos y hacer que los idiomas sepan que los métodos representan eventos. Los métodos corresponden a operaciones de agregar y quitar, cada una de las cuales toma un parámetro de instancia de delegado del mismo tipo (el tipo del evento).

using System;

class Test

{

public event EventHandler MyEvent

{

add

{

Console.WriteLine ("add operation");

}

remove

{

Console.WriteLine ("remove operation");

}

}

static void Main()

{

Test t = new Test();

t.MyEvent += new EventHandler (t.DoNothing);

t.MyEvent -= null;

}

void DoNothing (object sender, EventArgs e)

{

}

}

**Un atajo: eventos de campo**

C # proporciona una manera simple de declarar una variable delegada y un evento al mismo tiempo. Esto se llama un evento similar a un campo y se declara de manera muy simple: es lo mismo que la declaración de evento "longhand", pero sin la parte "body":

public event EventHandler MyEvent;

Esto crea una variable delegada y un evento, ambos con el mismo tipo. El acceso al evento está determinado por la declaración del evento (por lo que el ejemplo anterior crea un evento público) pero la variable delegada siempre es privada. El cuerpo implícito del evento es el obvio para agregar / eliminar instancias de delegado a la variable delegada, pero los cambios se realizan dentro de un bloqueo.

private EventHandler \_myEvent;

public event EventHandler MyEvent

{

add

{

lock (this)

{

\_myEvent += value;

}

}

remove

{

lock (this)

{

\_myEvent -= value;

}

}

}

¿Cuál es el punto de los delegados?

Ahora que sabemos cuáles son, ¿qué sentido tiene tener delegados y eventos? La respuesta es la encapsulación.

**Delegar instancias: otros métodos**

Los tipos de delegados también pueden permitir un comportamiento asincrónico utilizando el par BeginInvoke / EndInvoke. Estos son opcionales en lo que respecta a la especificación CLI, pero los tipos de delegado de C# siempre los proporcionan.

Siguen el mismo modelo para la ejecución asincrónica que el resto de .NET, lo que permite proporcionar un controlador de devolución de llamada, junto con un objeto para almacenar información de estado. Los delegados se ejecutan en subprocesos creados por el grupo de subprocesos del sistema.

using System.Threading;

delegate int SampleDelegate(string data);

class AsyncDelegateExample1

{

static void Main()

{

SampleDelegate counter = new SampleDelegate(CountCharacters);

SampleDelegate parser = new SampleDelegate(Parse);

IAsyncResult counterResult = counter.BeginInvoke ("hello", null, null);

IAsyncResult parserResult = parser.BeginInvoke ("10", null, null);

Console.WriteLine ("Main thread continuing");

Console.WriteLine ("Counter returned {0}", counter.EndInvoke(counterResult));

Console.WriteLine ("Parser returned {0}", parser.EndInvoke(parserResult));

Console.WriteLine ("Done");

}

static int CountCharacters (string text)

{

Thread.Sleep (2000);

Console.WriteLine ("Counting characters in {0}", text);

return text.Length;

}

static int Parse (string text)

{

Thread.Sleep (100);

Console.WriteLine ("Parsing text {0}", text);

return int.Parse(text);

}

}

Main thread continuing

Parsing text 10

Counting characters in hello

Counter returned 5

Parser returned 10

Done