



Empowering Digital Skills For The Jobs Of The Future



by



Academy .NET

Delegate

Sommario

- I delegate
- Definizione di un delegate
- Multicasting
- GetInvocationList()
- Delegate Func<>
- Delegate Action<>
- Covarianza
- Controvarianza
- Lambda

I delegate

Nella piattaforma .NET il tipo **delegate** è un oggetto indipendente dai tipi che "punta" a un metodo o un insieme di metodi che possono essere richiamati in un secondo momento.

A differenza di un tradizionale puntatore a funzione C ++, i delegate .NET sono classi che dispongono di supporto incorporato per **multicast** e invocazione di metodi asincroni.

I delegate

Un **delegate** mantiene tre informazioni principali:

- L'indirizzo del metodo su cui effettua le chiamate
- I parametri (se presenti) di questo metodo
- Il **return** (se presente) di questo metodo

Dopo che un oggetto delegate è stato creato e sono state fornite le informazioni necessarie, può dinamicamente invocare i metodi a cui punta in fase di esecuzione.

Definizione di un delegate

Per creare un tipo delegate in C#, si utilizza la parola chiave **delegate**.

È necessario definire il delegate in modo che la sua firma corrisponda al metodo o ai metodi su cui punterà.

```
public delegate int delegateName(int x, int y);
```

Definizione di un delegate

Quando il compilatore C# elabora i tipi delegate, genera automaticamente una classe **sealed** derivante da **System.MulticastDelegate**.

Questa classe (insieme alla sua classe di base, **System.Delegate**) fornisce l'infrastruttura necessaria affinché il delegate conservi un elenco di metodi da richiamare in un secondo momento.

Delegate - Esempio

```
public delegate int delegateName(int x, int y);

public class SimpleMath
{
    public static int Add(int x, int y) => x + y;
    public static int Subtract(int x, int y) => x - y;
}

class Program
{
    static void Main(string[] args)
    {
        delegateName b = new delegateName(SimpleMath.Add);
        Console.WriteLine("10 + 10 is {0}", b(10, 10));
    }
}
```

Multicasting

I **delegate** hanno capacità di **multicast**.

Un oggetto delegate può mantenere un elenco di metodi da chiamare.

Quando si vuole aggiungere più metodi a un oggetto delegate, si utilizza semplicemente l'operatore **+=** anziché un'assegnazione diretta.

Multicasting

Considerare il seguente delegate e i due metodi:

```
public delegate void EmptyDelegate();

public void Method1()
{
    Console.WriteLine("Method1");
}

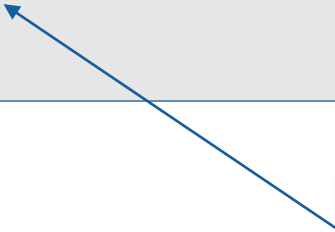
public void Method2()
{
    Console.WriteLine("Method2");
}
```

Multicasting

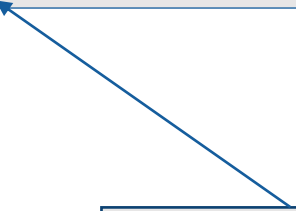
Multicasting:

```
EmptyDelegate multicast = Method1;  
multicast += Method2;
```

Creazione istanza
delegate



Aggiunta secondo
metodo al delegate



I delegate

- I delegate sono **immutabili**, l'operatore **+** crea un nuovo delegate internamente.
- Uno stesso metodo può essere aggiunto più volte alla lista di invocazione di un delegate.
- Tramite l'invocazione, il metodo verrà eseguito tante volte quante si presente in lista.

GetInvocationList()

GetInvocationList() permette di ottenere una lista di invocazione dei metodi di tipo **Delegate[]**

```
Delegate[] list= multicast.GetInvocationList();

foreach(Delegate del in list)
{
    Console.WriteLine("invocazione {0}", del.Method);
    ((EmptyDelegate)del).Invoke();
}
```

Multicasting

Per rimuovere un delegate dalla lista di invocazione si utilizzano gli operatori - e -=:

```
multicast -= Method1;
```

Delegate: fasi

- Definire un delegate personalizzato che corrisponda al formato del metodo utilizzato.
- Creare un'istanza del delegate personalizzato, passando un metodo come argomento del costruttore.
- Richiamare il metodo indirettamente, tramite una chiamata a **Invoke()** sull'oggetto delegate.

Delegate Func<> e Action<>

In molti casi, si necessita semplicemente di un delegate che accetta una serie di argomenti e possibilmente che abbia un valore di ritorno diverso da void.

In questi casi è possibile utilizzare i tipi delegate **Action<>** e **Func<>** incorporati nel framework.

Action<>

Il delegate **Action<>** **generico** è possibile utilizzarlo per "puntare" un metodo che richiede fino a 16 argomenti.

- Restituisce **void**
- È necessario specificare anche i tipi sottostanti di ogni parametro

```
public delegate void Action<in T>(T obj);
```

Action<> - Esempio

```
class Program
{
    static void Main(string[] args)
    {
        string message = "Hello World";
        Action<string> actionDelegate = printMessage;
        actionDelegate("Hello World");
    }

    public static void printMessage(string msg)
    {
        Console.WriteLine(msg);
    }
}
```

Func<>

Func<>:

- Ha tipo di ritorno
- Tenere presente che il parametro finale di **Func<>** è sempre il **valore restituito dal metodo**.

```
public delegate TResult Func<in T, out TResult>(T arg);
```

Func<> - Esempio

```
class Program
{
    static void Main(string[] args)
    {
        Func<int, int, int> funcDelegate = sum;
        Console.WriteLine(funcDelegate(2, 2));
    }

    public static int sum(int x, int y)
    {
        return x + y;
    }
}
```

Espressioni Lambda

- Un'espressione lambda è una **funzione anonima** contenente espressioni e istruzioni

```
var lambda = (a => a > 5);
```

- Espressioni lambda
 - Usa l'operatore lambda =>
 - Il lato sinistro specifica i parametri di input
 - Il lato destro contiene l'espressione o l'affermazione

Funzioni Lambda

Le funzioni Lambda sono metodi inline (funzioni) che accettano parametri di input e restituiscono valori:

`x => x / 2`



```
static int Func(int x) { return x / 2; }
```

`x => x != 0`



```
static bool Func(int x) { return x != 0; }
```

`() => 42`



```
static int Func() { return 42; }
```

`(x, y) => x+y`



```
static int Func(int x, int y)  
{ return x+y; }
```

Espressioni Lambda

- Per creare un'espressione lambda, è necessario specificare i parametri di input (se presenti) sul lato sinistro dell'operatore lambda e un'espressione o un blocco di istruzioni sul lato destro.
- Qualsiasi espressione lambda può essere convertita in un tipo **delegate**

Espressioni Lambda

Il tipo **delegate** in cui è possibile convertire un'espressione lambda è definito dai tipi dei relativi parametri e dal valore restituito.

- Se un'espressione lambda è **void**, può essere convertita in un tipo delegate **Action<>**
- Se un'espressione lambda ha **tipo di ritorno**, può essere convertita in un tipo **Func<>**

Espressioni Lambda - Esempi

```
static void Main(string[] args)
{
    Func<int, int, int> somma = (x, y) => x + y;
    Func<int, int, int> divisione = (x, y) => x - y;
    Func<int, int, int> moltiplicazione = (x, y) => x * y;

    Console.WriteLine(somma(2, 7));
    Console.WriteLine(divisione(2, 7));
    Console.WriteLine(moltiplicazione(2, 7));
}
```

Espressioni Lambda - Esempi

```
static void Main(string[] args)
{
    Action<int, int> somma = (x, y) =>
    {
        int sum = x + y;
        Console.WriteLine(sum);
    };

    somma(3,3);
}
```

Domande & approfondimenti

Academy .NET