Eventi

Implementazione di eventi

Ambiente: .NET 4.0/C# 4.0

Anno 2014/2015

Indice generale

1 Introduzione

Un *evento* implementa un meccanismo usato per notificare che è "accaduto o sta accadendo qualcosa". L'oggetto o modulo che solleva l'evento si chiama ***sender*** dell'evento (mittente). Gli oggetti o moduli che ricevono la notifica si chiamano ***subscribers*** (sottoscrittori) dell'evento.

In pratica, un evento (in generale un *delegate*) è un meccanismo per eseguire indirettamente uno o più metodi, convenzionalmente chiamati *event handler* (gestori di evento). Indirettamente, perché i metodi che saranno eseguiti non sono stabiliti dal *sender*, ma dai *subscribers*.

Gli eventi si possono suddividere in due categorie:

- **eventi di sola notifica**: si limitano a notificare senza aggiungere ulteriori informazioni;
- **eventi con parametro**: forniscono informazioni che qualificano l'evento. (Ad esempio il codice del testo premuto nell'evento `KeyPress`.)

1.1 Eventi di sola notifica

Si ipotizzi di progettare una classe che debba eseguire un determinato procedimento; si vuole che il completamente del procedimento venga notificato ai *subscribers*. A questo scopo la classe definisce il metodo `Esegui()` e l'evento `Completato`, che sarà eseguito al termine del metodo:

```
public class Worker
{
    public event EventHandler Completato;  //definisce l'evento

    public void Esegui()
    {
        Thread.Sleep(1000);                    // simula l'esecuzione di un compito
        Completato(this, EventArgs.Empty);    // esegue (solleva) l'evento

    }
}
```

Ci sono alcune considerazioni da fare:

- L'evento viene eseguito come se fosse un metodo con due parametri. Il primo parametro, di tipo `object`, rappresenta il mittente (*sender*) dell'evento, e infatti viene valorizzato con `this`. Il secondo parametro definisce le informazioni associate all'evento; in questo caso, non essendoci informazioni, viene passato il valore predefinito `EventArgs.Empty`.
- Nella definizione, `Completato` è di tipo `EventHandler`; questo è un *tipo delegate* e stabilisce la forma (*signature*) dei metodi che possono essere collegati all'evento:

  ```
  public delegate void EventHandler(object sender, EventArgs e);
  ```

  `EventHandler` stabilisce, appunto, che i metodi collegati all'evento devono dichiarare due parametri: il primo di tipo `object`, il secondo di tipo `EventArgs`.

1.1.1 "Consumazione" dell'evento

L'esecuzione di un evento non produce alcun risultato se non c'è almeno un metodo collegato ad esso. Occorre dunque "sottoscriversi" all'evento:

```
static void Main(string[] args)
{
    Worker worker = new Worker();
    worker.Completato += w_Completato;   // sottoscrizione all'evento

    Console.WriteLine("Avvio compito");
    w.Esegui();
    Console.ReadLine();
}

// gestore dell'evento, destinatario della chiamata: Completato(this, EventArgs.Empty);
static void w_Completato(object sender, EventArgs e)
{
    Console.WriteLine("Compito completato");
}
```

All'evento viene collegato l'*event handler* `w_Completato()`; questo viene invocato quando, nel metodo `Esegui()`, viene eseguita l'istruzione: `Completato(this, EventArgs.Empty)`.

1.1.2 Verificare la sottoscrizione dell'evento

L'attuale implementazione di `Completato` non è corretta, poiché presuppone che almeno un *event handler* venga collegato all'evento, ma non è detto che sia così; in questo caso, l'esecuzione dell'evento produrrebbe una `NullReferenceException`. L'approccio corretto prevede di verificare l'effettiva sottoscrizione all'evento:

```
public void Esegui()
{
    Thread.Sleep(1000);
    if (Completato != null)    // verifica che l'evento sia stato sottoscritto
        Completato(this, EventArgs.Empty);
}
```

Dalla versione 6 di C#, la `if()` può essere sostituita dall'operatore `?.`, che svolge in pratica lo stesso lavoro:

```
public void Esegui()
{
    Thread.Sleep(1000);
    Completato?.Invoke(this, EventArgs.Empty);  //verifica sottoscrizione e solleva evento
}
```

1.2 Eventi con parametro

In alcuni scenari la notifica dell'evento in sé non contiene l'informazione necessaria ai *subscribers*. Un esempio tipico sono gli eventi di tastiera: nella maggior parte dei casi, il codice che gestisce l'evento deve conoscere il tasto che è stato premuto.

L'implementazione di eventi con parametro è più elaborata rispetto a quelli di sola notifica:

- Occorre implementare una nuova classe che rappresenta il tipo del parametro dell'evento. La classe deve derivare da `EventArgs` e deve definire le informazioni pertinenti per l'evento.
- Occorre utilizzare il *delegate* generico `EventHandler<>`.

Supponiamo di voler modificare l'implementazione dell'evento `Completato`, in modo che fornisca il tempo impiegato a eseguire il procedimento. Occorre innanzitutto definire una nuova classe che definisca l'informazione richiesta:

```
public class CompletatoEventArgs: EventArgs
{
    public TimeSpan TempoImpiegato { get; set; } // dato associato all'evento
}
```

Nota bene: per convenzione, il nome della classe è dato da "*nome-evento+EventArgs*".

Il secondo passo è quello di modificare la definizione dell'evento:

```

public class Worker
{
    public event EventHandler<CompletatoEventArgs> Completato;
    ...
}
```

Nota bene: il delegate generico `EventHandler<>` specifica il tipo del parametro.

Infine, occorre modificare il codice che solleva l'evento:

```
public class Worker
{
    public event EventHandler<CompletatoEventArgs> Completato;

    public void Esegui()
    {
        Stopwatch sw = new Stopwatch();
        sw.Start();
        Thread.Sleep(1000);
        sw.Stop();

        // crea il parametro da passare all'evento
        var e = new CompletatoEventArgs { TempoImpiegato = sw.Elapsed };
        Completato?.Invoke(this, e);
    }
}
```

1.2.1 “Consumazione” di un evento con parametro

Per il codice che gestisce l’evento non cambia niente, se non la *signature* dell’*event handler*. Il secondo parametro non sarà più di tipo `EventArgs`, ma `CompletatoEventArgs`:

```
static void Main(string[] args)
{
    Worker w = new Worker();
    w.Completato += w_Completato;

    Console.WriteLine("Avvio compito");
    w.Esegui();
    Console.ReadLine();
}

// Destinatario della chiamata: Completato(this, e)
static void w_Completato(object sender, CompletatoEventArgs e)
{
    Console.WriteLine("Compito completato in: {0} sec", e.TempoImpiegato.Seconds);
}
```