Eventi

Implementazione di eventi

Anno 2014/2015

Indice generale

1 Introduzione

Un *evento* implementa un meccanismo usato per notificare che è "accaduto o sta accadendo qualcosa". L'oggetto che solleva l'evento si chiama ***sender*** dell'evento (mittente). Gli oggetti che ricevono la notifica si chiamano ***subscribers*** (sottoscrittori) dell'evento[1].

Un evento rappresenta un meccanismo per eseguire indirettamente un metodo (o più metodi), convenzionalmente chiamato *event handler* (*gestore di evento*). Indirettamente, poiché il metodo, definito nel *subscriber*, viene eseguito dal *sender* attraverso l'evento.

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`.)

È possibile implementare gli eventi mediante qualsiasi tipo di *delegate*; di norma, si utilizzano i *delegate* `EventHandler` e `EventHandler<>`.

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:

- Il *delegate* `EventHandler` stabilisce la forma (*signature*) dei metodi che possono essere sottoscritti all'evento:

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

  Il *delegate* corrisponde a un metodo con due parametri. Il primo memorizza il mittente dell'evento, il secondo può essere ignorato.

- Nell'esecuzione dell'evento, al primo argomento viene passato `this`, e cioè l'oggetto mittente. Non essendoci informazioni associate all'evento, il secondo parametro viene valorizzato con il valore predefinito `EventArgs.Empty`.

1 Anche i moduli possono implementare e gestire gli eventi.

1.1.1 "Consumazione" dell'evento

L'operazione di "sottoscrizione" stabilisce il metodo da eseguire quando l'evento viene sollevato:

```
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");
}
```

Nell'esempio, all'evento viene collegato il metodo `w_Completato()`; questo sarà 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 metodo 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 lo stesso lavoro:

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

1.2 Eventi con dati

In alcuni scenari l'evento in sé non contiene l'informazione necessaria ai *subscribers*. Un esempio è fornito dagli eventi di tastiera: il codice che gestisce l'evento vuole conoscere il tasto che è stato premuto.

L'implementazione di eventi ai quali sono associati dei dati è più elaborata rispetto a quelli di sola notifica, poiché occorre utilizzare un delegate che consenta di fornire i dati dell'evento. Di norma:

- Si implementa una classe che definisce i dati dell'evento.
- Si utilizza il *delegate* generico `EventHandler<>`.

> **EventHandler<> e dati dell'evento**
>
> Nella versioni di .NET antecedenti alla 4.5, la classe che definisce i dati dell'evento doveva derivare dalla classe `EvenArgs`. Adesso non è più così: è possibile utilizzare qualsiasi tipo di dato.
>
> Per convenzione, comunque, è opportuno utilizzare una classe che definisce il dato o i dati dell'evento. Il nome della classe dovrebbe terminare con `EventArgs`.

Di seguito modifico l'implementazione dell'evento `Completato`, in modo che fornisca il tempo impiegato a eseguire il procedimento. Innanzitutto implemento una nuova classe che definisca l'informazione richiesta: `CompletatoEventArgs`:

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

Il secondo passo è quello di modificare la definizione dell'evento, utilizzando il delegate generico `EventHandler<>`:

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

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 dati*

La gestione dell'evento non cambia, a parte nella *signature* del metodo. 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
static void w_Completato(object sender, CompletatoEventArgs e)
{
    Console.WriteLine("Compito completato in: {0} sec", e.TempoImpiegato.Seconds);
}
```