SỰ KIỆN

Event là gì?

Trong bài học trước bạn đã nắm được khái niệm và cách sử dụng cơ bản của delegate trong C# .NET. Một trong những vai trò của delegate mà chúng ta đã nhắc tới là giúp object tương tác với object khác.

Khả năng hỗ trợ tương tác của delegate được vận dụng trong một mô hình lập trình gọi là *publisher/subscriber*, thường gọi tắt là mô hình *pub/sub*.

Bạn có thể dễ dàng hình dung về mô hình này tương tự như khi bạn đăng ký theo dõi một kênh Youtube. Trong đó kênh Youtube đóng vai trò **publisher** (hoặc cũng được gọi là **broadcaster**). Bạn và những người đăng ký khác đóng vai trò **subscriber**.

Mỗi khi có video mới, kênh sẽ thực hiện thông báo cho người theo dõi. Hành động này trong mô hình **event** được gọi là *raise/invoke/broadcast* (tạm dịch là *phát*). Khi một event được raise/invoke/broadcast, các subscriber sẽ thực hiện các hoạt động riêng của mình (như có bạn thì thích xem ngay, có bạn lại muốn để vào danh sách xem sau, người thì muốn tải về máy, v.v.).

Khi sử dụng delegate bạn hoàn toàn có thể thực hiện mô hình pub/sub.

Tuy nhiên mô hình pub/sub đưa ra thêm hai yêu cầu quan trọng:

1. các subscriber không được biết và không ảnh hưởng lẫn nhau;
2. Việc raise/invoke một event chỉ được phép thực hiện bởi broadcaster.

Delegate không đáp ứng được các yêu cầu này do:

1. delegate cho phép subscriber sử dụng phép gán `=`. Khi đó tất cả các subscriber sẵn có sẽ bị hủy.
2. client code cũng có thể kích hoạt delegate.

Vì vậy .NET đưa thêm vào từ khóa `event` để dễ dàng vận dụng delegate vào mô hình pub/sub.

Như vậy, event trong .NET thực chất chỉ là một dạng hạn chế của delegate để phù hợp với mô hình pub/sub. Hoặc cũng có thể nói rằng event được xây dựng bên trên delegate. Do đó, trước khi làm việc với event, bạn phải hiểu delegate.

Kỹ thuật lập trình với Event

Để hiểu kỹ thuật lập trình với event trong C#, hãy cùng xem ví dụ sau:

```
using static System.Console;
namespace ConsoleApp1 {
    internal class Program {
        private static void Main(string[] args) {
            Title = "C# events";

            var gasoline = new Gasoline();

            gasoline.PriceChanged += Stock_PriceChanged1;
            gasoline.PriceChanged += Stock_PriceChanged2;

            gasoline.Price += 50;

            gasoline.Price += 20;

            gasoline.PriceChanged -= Stock_PriceChanged1;

            gasoline.Price += 10;

            ReadKey();
        }

        private static void Stock_PriceChanged1(decimal oldPrice, decimal newPrice) {
            WriteLine($"Damn it! The price changed again to {newPrice}đ");
        }

        private static void Stock_PriceChanged2(decimal oldPrice, decimal newPrice) {
            WriteLine($"The price has been changed from ${oldPrice} to {newPrice}đ");
        }
    }

    internal delegate void PriceChangedHandler(decimal oldPrice, decimal newPrice);

    internal class Gasoline {
        private decimal _price = 11000;
        public event PriceChangedHandler PriceChanged;

        public decimal Price {
            get => _price;
            set {
                if (_price == value) return;
                var oldPrice = _price;
                _price = value;
                PriceChanged?.Invoke(oldPrice, _price);
            }
        }
    }
}
```

Đây là một chương trình đơn giản minh họa việc tăng giá xăng và phản ứng từ hai người theo dõi. Trong ví dụ này, `Gasoline` là *publisher/broadcaster*, hai phương thức `Stock_PriceChanged1` và `Stock_PriceChanged2` là các *subscriber* và cũng được gọi là các *event handler* (phương thức xử lý sự kiện).

Kết quả chạy chương trình

Hãy để ý, hai lần đầu tăng giá thì cả hai người đều phản ứng. Riêng lần cuối cùng chỉ có 1 người phản ứng vì người kia đã hủy theo dõi sự kiện.

Khai báo kiểu delegate sử dụng cho event

Do event thực chất cũng là cách tương tác giữa các object, bạn cần đến một kiểu delegate để quy định hình thức của các phương thức subscriber:

```
delegate void PriceChangedHandler(decimal oldPrice, decimal newPrice);
```

Khai báo "biến" event

Chúng ta dùng kiểu delegate này để khai báo một sự kiện với từ khóa event:

```
public event PriceChangedHandler PriceChanged;
```

Với kiểu delegate này bạn hoàn toàn có thể khai báo một biến chấp nhận các phương thức có cùng mô tả `(decimal, decimal)->void`. Khai báo biến so với khai báo event chỉ khác biệt duy nhất là không có từ khóa event. Do vậy bạn cũng có thể hình dung khai báo event thực chất cũng chỉ là khai báo một biến delegate (đặc biệt).

Quy ước đặt tên event: event được đặt tên theo quy ước PascalCase (viết hoa chữ cái đầu mỗi từ).

Kích hoạt sự kiện

Mỗi khi thay đổi giá xăng (gán giá trị mới cho property Price) thì kích hoạt sự kiện, tức là gửi thông báo đến tất cả subscriber (mà thực tế mình chưa biết).

```
PriceChanged?.Invoke(oldPrice, _price);
```

Đây thực chất là lời gọi delegate. Trên thực tế, việc sử dụng event bên trong class publisher hoàn toàn không có gì khác biệt với sử dụng biến delegate.

Ở trên chúng ta sử dụng phương thức Invoke của kiểu Delegate cơ sở (mà mọi kiểu delegate đều kế thừa). Đồng thời chúng ta sử dụng thêm phép toán kiểm tra null `?`.

Đây là cách kích hoạt event an toàn (mà bạn cũng nên sử dụng với biến delegate thông thường). Lý do là nếu không có subscriber nào, biến event PriceChanged sẽ nhận giá trị null. Mọi lệnh từ biến null sẽ đều dẫn

đến `NullException`.

Dĩ nhiên bạn cũng có thể kích hoạt event như một delegate thông thường bằng lời gọi trực tiếp:

```
PriceChanged(oldPrice, _price);
```

Tuy nhiên cách làm này không an toàn và không khuyến khích sử dụng.

Đăng ký/hủy đăng ký theo dõi sự kiện

Sự khác biệt của event so với delegate là ở cách sử dụng trong client code. Đối với event bạn chỉ có thể dùng phép toán `+=` (đăng ký theo dõi) hoặc `-=` (hủy đăng ký), không thể sử dụng phép gán `=`.

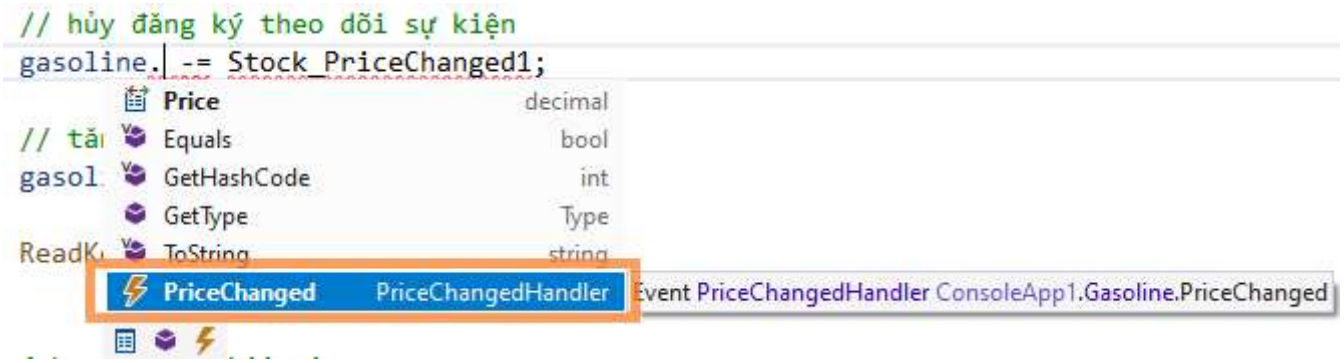
```
gasoline.PriceChanged += Stock_PriceChanged1;
gasoline.PriceChanged += Stock_PriceChanged2;

gasoline.PriceChanged -= Stock_PriceChanged1;
```

Sở dĩ bạn chỉ sử dụng được phép toán `+=` hoặc `-=` là để đảm bảo rằng các handler (subscriber) không ảnh hưởng (ghi đè, xóa lãn nhau) đến nhau. Đây là cách event hạn chế tác dụng của delegate nhằm đảm bảo thực thi mô hình pub/sub.

Dĩ nhiên, ở vị trí của subscriber bạn có thể sử dụng phương thức thành viên, phương thức tĩnh, hàm lambda hoặc hàm cục bộ, giống hệt như đối với delegate.

Một khi hủy đăng ký, khi broadcaster kích hoạt sự kiện, phương thức subscriber sẽ không được kích hoạt nữa.

Trong Visual Studio, event được hiển thị với biểu tượng khác biệt với biến và property:



Nếu bạn đặt dấu chấm sau tên event, bạn sẽ thấy rằng intellisense không hề hiển thị thêm cái gì. Đây là cách event ngăn chặn truy cập vào các thành viên của delegate đứng sau event.

Như vậy qua phần này bạn có thể thấy, làm việc với event thực chất là làm việc với delegate nhưng ở một dạng hạn chế để đảm bảo phù hợp với yêu cầu của mô hình pub/sub.

Mô hình sự kiện chuẩn

Ở phần trên bạn đã thấy lập trình với event thực chất là làm việc với delegate. Bạn có thể tiếp tục sử dụng mô hình như trên.

Tuy nhiên, .NET đưa ra một mẫu tiêu chuẩn để làm việc với event với mục đích giúp lập trình event được chuẩn hóa ở cả các class của .NET lẫn ở client code (người dùng viết).

Mô hình này đưa ra hai yêu cầu:

Thứ nhất, .NET yêu cầu gộp tất cả các thông tin broadcaster cần gửi cho subscriber vào một class con của lớp `EventArgs` (`System.EventArgs`).

Lấy ví dụ, nếu cần cung cấp thông tin về giá cũ và giá mới cho event handler (subscriber), bạn cần xây dựng một class như sau:

```
class PriceChangedEventArgs : EventArgs {
    public readonly decimal LastPrice;
    public readonly decimal NewPrice;
    public PriceChangedEventArgs(decimal lastPrice, decimal newPrice) {
        LastPrice = lastPrice;
        NewPrice = newPrice;
    }
}
```

Class này kế thừa từ EventArgs và chứa hai thông tin cần cho các subscriber LastPrice và NewPrice.

Để dễ phân biệt, các class con kế thừa từ EventArgs cũng thường có hậu tố (postfix) là EventArgs như bạn đã thấy ở lớp PriceChangedEventArgs.

Yêu cầu này thống nhất việc truyền thông tin từ publisher tới subscriber.

Thứ hai, delegate đứng sau mỗi event cần tuân thủ 3 quy định:

1. kiểu trả về phải là `void`.
2. phải có hai tham số: tham số thứ nhất thuộc kiểu `object`, tham số thứ hai là class con của `EventArgs` (mà bạn đã xây dựng theo yêu cầu thứ nhất).

Tham số thứ nhất sẽ được sử dụng để chứa thông tin về broadcaster. Tham số thứ hai chứa thông tin broadcaster gửi cho subscriber.

3. tên của delegate phải có hậu tố `EventHandler`.

Yêu cầu này thống nhất cách xây dựng kiểu delegate chống lưng cho event.

Lấy ví dụ delegate trong ví dụ trên cần được cải tạo thành:

```
delegate void PriceChangedEventHandler(object sender, PriceChangedEventArgs args);
```

Lớp Gasoline viết lại theo mô hình event tiêu chuẩn của .NET như sau:

```
delegate void PriceChangedEventHandler(object sender, PriceChangedEventArgs args);
internal class Gasoline {
    private decimal _price = 11000;

    public event PriceChangedEventHandler PriceChanged;
    public decimal Price {
        get => _price;
        set {
            if (_price == value) return;
            var oldPrice = _price;
            _price = value;

            var args = new PriceChangedEventArgs(oldPrice, _price);
            PriceChanged?.Invoke(this, args);
        }
    }
}
```

Lối viết như trên mặc dù đúng tiêu chuẩn nhưng là phương pháp đã cũ.

Từ C# 2.0, .NET định nghĩa thêm generic delegate `EventHandler<T>` để đơn giản hóa việc định nghĩa event. Khi sử dụng `EventHandler<T>` bạn có thể trực tiếp khai báo event trong class mà không cần định nghĩa kiểu delegate nữa:

```
public event EventHandler<PriceChangedEventArgs> PriceChanged;
```

Class Gasoline có thể viết lại theo mô hình tiêu chuẩn mới như sau:

```
internal class Gasoline {
    private decimal _price = 11000;
    public event EventHandler<PriceChangedEventArgs> PriceChanged;
    public decimal Price {
        get => _price;
        set {
            if (_price == value) return;
            var oldPrice = _price;
            _price = value;

            var args = new PriceChangedEventArgs(oldPrice, _price);
            PriceChanged?.Invoke(this, args);
        }
    }
}
```

Đây là phương pháp chuẩn nhất khi lập trình sự kiện trong C# .NET mà bạn cần tuân thủ.

Ứng dụng event

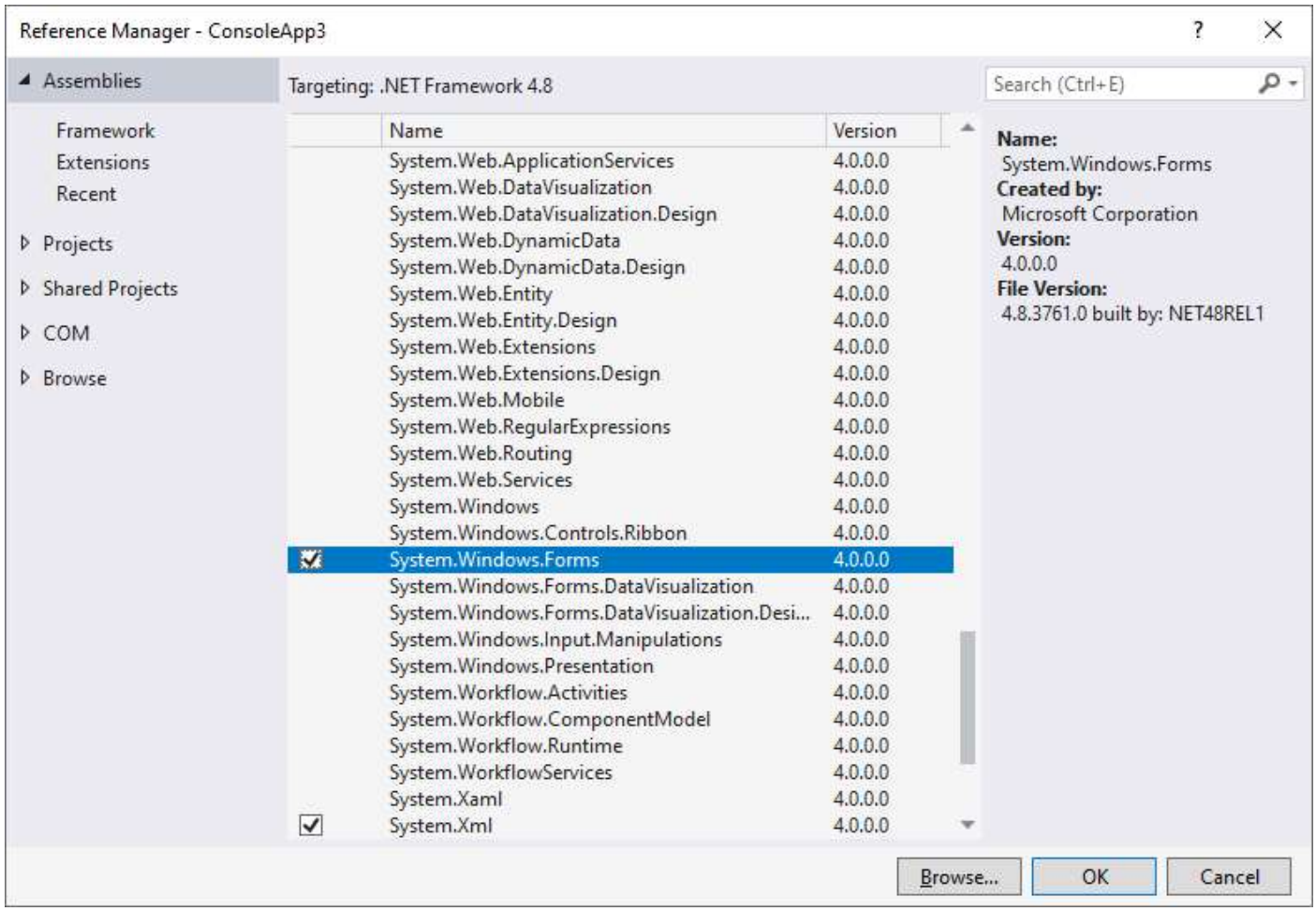
Trong .NET, event được sử dụng ở nhiều nơi, tiêu biểu nhất là trong hai UI framework Windows Forms và WPF. Ngoài ra mô hình này cũng được sử dụng rộng rãi ở những thư viện khác.

Chúng ta sẽ làm một ví dụ nhỏ với Windows Forms để minh họa.

Do đây không phải là bài học về Windows Forms, chúng ta sẽ vẫn sử dụng một Console project thông thường cho đơn giản.

Bước 1. Tạo một Console project mới.

Bước 2. Tham chiếu tới assembly `System.Windows.Forms` như sau:



Viết code cho Program như sau:

```
using System;
using System.Windows.Forms;
namespace ConsoleApp3 {
    internal class Program {
        [STAThread]
        private static void Main() {

            var form = new Form();

            var button = new Button {
                Text = "Press me!",
            };

            button.Click += OnButtonOnClick;

            form.Controls.Add(button);
```

```
            Application.EnableVisualStyles();

            Application.Run(form);
        }

        private static void OnButtonOnClick(object sender, EventArgs args)
        {
            MessageBox.Show("Xin chào. Đây là thông báo từ phương thức xử lý sự kiện click của nút bấm!");
        }
    }
}
```

Hãy để ý cách chúng ta đăng ký xử lý sự kiện `Click` và phương thức xử lý sự kiện `OnButtonOnClick`. Hãy so sánh nó với các yêu cầu và quy định của mô hình xử lý sự kiện .NET mà chúng ta đã học ở phần trên.