

Cơ chế Delegate & Event

ThS Nguyễn Duy Hải

Nội dung

- Delegate
 - Khái niệm delegate
 - Thực thi delegate
 - Multicast delegate
 - Giải pháp cho hàm Sort tổng quát
- Event
 - Khái niệm event
 - Event & delegate
 - Cơ chế publishing & subscribing
 - Minh họa cơ chế event

Delegate

- Lớp đóng gói các phương thức (method signature)
- Dùng trong event-handling model của C#
- Đặc tính
 - Type safe
 - Object oriented mechanism
- Delegate là class:
 - Có instance
 - Có thể chứa những tham chiếu đến 1 hay nhiều method

Delegate

- Một delegate định nghĩa một signature
 - Return type
 - Sequence of parameter types
- Tất cả các method có cùng signature có thể được add vào thể hiện của delegate
- Delegate instance có một danh sách các tham chiếu method
 - Cho phép add (+) các method
 - Có thể remove (-) các method

Define delegate

public delegate void MyDelegate1(int x, int y)



**Delegate cho dạng hàm:
void Method(int, int)**

public delegate string MyDelegate2(float f)



**Delegate cho dạng hàm:
string Method(float)**

Instance delegate

```
public void Method1(int x, int y)
```

```
{
```

```
    ...
```

```
}
```

```
...
```

```
MyDelegate1 del1 = new MyDelegate1(Method1);
```

```
public string Method2(float f)
```

```
{
```

```
    ...
```

```
}
```

```
...
```

```
MyDelegate2 del2 = new MyDelegate2(Method2);
```

Call Delegate

Gọi del1

**int x = 5, y = 10;
del1(x, y);**

del1(10, 20);

**int y = 2;
del1(100, y);**

Gọi del2

**float f = 0.5f;
string s;
s = del2(f);**

string s = del2(100f);

Multi Cast

```
void Print(int x,int y) {  
    Console.WriteLine("x = {0}, y = {1}", x, y);  
}  
void Sum(int x, int y) {  
    Console.WriteLine("Tong = {0}", x+y);  
}
```

```
MyDelegate1 mulDel = new MyDelegate1(Print);  
mulDel += new MyDelegate1(Sum);
```

```
mulDel(5, 10);
```

```
mulDel -= new MyDelegate1(Print);  
mulDel(5,10);
```


Problem

**Xây dựng hàm Sort
tổng quát cho cho
mảng đối tượng có
kiểu bất kỳ**



Solution

- Phân tích
 - Nếu đối tượng là kiểu số như int, long, float thì ko có vấn đề
 - Trường hợp đối tượng phức khác?

**So sánh theo
quy tắc nào**



Solution

- Giải pháp:
 - Cho phép đối tượng tự quy định thứ tự của chúng
 - Sử dụng delegate để truyền phương thức so sánh này vào hàm Sort

void Sort(object[] list, CompareObj cmp)



Delegate này sẽ tham chiếu tới hàm Compare của lớp MyClass. Chính lớp MyClass sẽ quy định thứ tự của các đối tượng

Solution

- Mô tả delegate CompareObj cho hàm Sort:

Tên của delegate

```
public delegate bool CompareObj(object o1,object o2)
```

**Trả về true: nếu o1 “trước” o2
false: ngược lại**

2 đối tượng cần so sánh

Solution

Định nghĩa hàm Sort tổng quát cho các lớp

Delegate sẽ trở tới hàm Compare riêng của lớp tương ứng


```
public static void Sort(object[] objs, CompareObj cmp)
{
    for(int i=0; i < objs.Length-1; i++)
        for(int j=objs.Length-1; j>i; j--)
            if ( cmp( objs[j], objs[j-1] ) )
            {
                Swap( objs[j], objs[j-1] );
            }
}
```

Yêu cầu lớp tự so sánh

Solution

- Các lớp hỗ trợ Sort thì phải
 - Cung cấp hàm Compare riêng
 - Signature phải thoả delegate CompareObj

```
class Person {  
    private string name;  
    private int weight;  
    private int yearOfBirth;  
    public static bool CompareName(object p1, object p2) {  
        if (string.Compare(((Person)p1).name, ((Person)p2).name)<0)  
            return true;  
        return false;  
    }  
}
```



Cùng signature

Solution

```
public delegate bool CompareObj(object o1,object o2);
```

```
...
```

```
Person[ ] persons = new Person[4];
```

```
persons[0] = new Person("Quy Mui", 2, 2004);
```

```
persons[1] = new Person("Ha Giang", 65, 1978);
```

```
persons[2] = new Person("Ngoc Thao", 47, 1979);
```

```
persons[3] = new Person("Ha Nam", 65, 1932);
```

```
CompareObj cmp = new CompareObj(Person.CompareName);
```

```
HaGLib.Sort( persons, cmp );
```

Gọi hàm static Sort

Lớp chứa hàm Sort



Event

Event

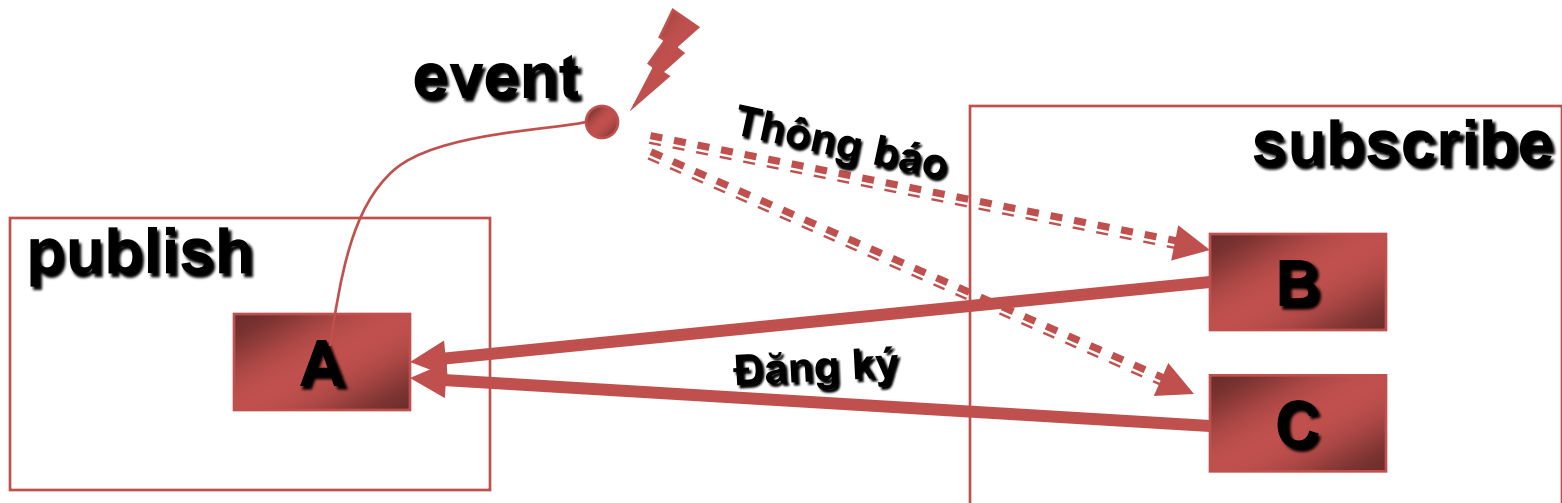
- Cơ chế thông điệp giữa các lớp hay các đối tượng
- Có thể thông báo cho lớp khác biết được khi một lớp có phát sinh điều gì đó
- **Publisher**: lớp phát sinh sự kiện
- **Subscriber**: lớp nhận hay xử lý khi sự kiện xảy ra

Event

- Trong môi trường giao diện GUIs (Graphical User Interfaces: GUIs):
 - Button đưa ra sự kiện “*Click*”, cho phép lớp khác có thể đáp ứng (xử lý) khi sự kiện này xảy ra.
- VD: Button “*Add*” trong Form, khi sự kiện click xảy ra thì Form thực hiện lấy dữ liệu từ các TextBox đưa vào ListBox...

Publishing & Subscribing

- Một lớp có publish một tập các event cho phép các lớp khác subscribe
 - Button là lớp publish đưa ra event: click
 - Form là lớp subscribe có phần xử lý riêng khi “*click*” của Button kích hoạt.



Event & Delegate

- Sự kiện trong C# được thực thi nhờ uỷ thác
 - Lớp publishing định nghĩa uỷ thác
 - Những lớp subscribing phải thực thi
 - Khi sự kiện xuất hiện thì phương thức của lớp subscribing được gọi thông qua uỷ thác.
- Phương thức để xử lý sự kiện gọi là **trình xử lý sự kiện** (event handler)

Event & Delegate

- Trình xử lý sự kiện trong .NET Framework được mô tả như sau:
 - Trả về giá trị void
 - Tham số 1: nguồn phát sinh sự kiện, đây chính là đối tượng publisher
 - Tham số 2: là đối tượng thuộc lớp dẫn xuất từ EventArgs
- Phải thực hiện trình xử lý sự kiện theo đúng mẫu trên!

Event & Delegate

- Khai báo delegate xử lý sự kiện

```
public delegate void HandlerName(object obj, EventArgs arg);
```

- Khai báo event

```
public event HandlerName OnEventName;
```

- Các lớp muốn xử lý khi sự kiện OnEventName phát sinh thì phải thực thi event handler

Minh họa 1

- Xây dựng 1 lớp thực hiện yêu cầu: “*cứ mỗi giây sẽ phát sinh 1 sự kiện*”
- Cho phép 2 lớp khác đăng ký xử lý sự kiện này, mỗi lớp có cách xử lý riêng:
 - **Lớp A**: hiển thị thời gian theo “*mô phỏng đồng hồ analog*”
 - **Lớp B**: hiển thị thời gian theo “*mô phỏng đồng hồ digital*”

Minh họa 1

- Tạo một lớp Clock:
 - Khai báo một event: OnSecondChange
 - Một phương thức Run: cứ 1s thì phát sinh sự kiện OnSecondChange
- Tạo 2 lớp: AnalogClock và DigitalClock nhận xử lý sự kiện OnSecondChange của lớp Clock

Minh họa 1

- Khai báo delegate xử lý event

Tên delegate xử lý sự kiện

```
delegate void SecondChangeHandler(object clock, EventArgs info);
```

Đối tượng phát sinh event

Tham số kiểu EventArgs

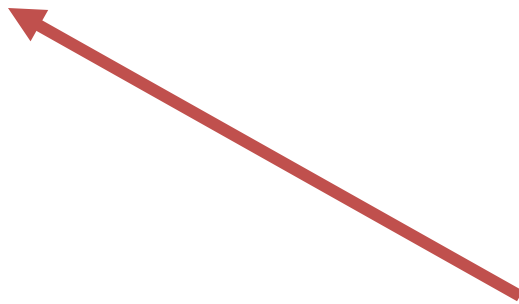
Minh họa 1

- Khai báo event có hàm xử lý mô tả trên

Kiểu delegate



event **SecondChangeHandler** **OnSecondChange**;



Tên của event



Từ khóa event: thể hiện cơ chế publishing & subscribing

Minh họa 1

- Kích hoạt sự kiện

**Kiểm tra xem có hàm xử lý
được đăng ký hay không?**

```
if (OnSecondChange != null)  
    OnSecondChange(this, new EventArgs());
```

Gọi hàm xử lý sự kiện đã đăng ký

Minh họa 1

```
public class Clock {  
    public delegate void  
        SecondChangeHandler(object clock, EventArgs info);  
  
    public event SecondChangeHandler OnSecondChange;  
  
    public void Run() {  
        while (true) {  
            Thread.Sleep(1000);  
            if (OnSecondChange != null)  
                OnSecondChange(this, new EventArgs());  
        }  
    }  
}
```

Minh họa 1


- Lớp DigitalClock
 - Định nghĩa trình xử lý sự kiện của Clock
 - Đúng mô tả delegate hàm xử lý của lớp Clock
 - Thực hiện một số thao tác riêng của DigitalClock
 - Đăng ký xử lý với trình xử lý sự kiện trên khi có sự kiện OnSecondChange của Clock
 - Chức năng đăng ký với lớp Clock là có xử lý khi sự kiện OnSecondChange của Clock phát sinh
 - Ủy thác cho lớp Clock sẽ gọi trình xử lý định nghĩa bên trên của DigitalClock

Minh họa 1

- Trình xử lý của DigitalClock

Tên của trình xử lý

Đối tượng phát sinh sự kiện



```
public void Show(object obj, EventArgs args)
{
    DateTime date = DateTime.Now;
    Console.WriteLine("Digital Clock: {0}:{1}:{2}",
        date.Hour, date.Minute, date.Second);
}
```

Minh họa 1

- Đăng ký xử lý sự kiện

**Đối tượng này sẽ
phát sinh sự kiện**



```
public void Subscribe(Clock theClock)  
{  
    theClock.OnSecondChange +=  
        new Clock.SecondChangeHandler>Show);  
}
```

**Ủy thác phương thức Show cho
OnSecondChange**



Minh họa 1

```
public class DigitalClock
{
    public void Subscribe(Clock theClock)
    {
        theClock.OnSecondChange +=
            new Clock.SecondChangeHandler(Show);
    }
    public void Show(object obj, EventArgs args)
    {
        DateTime date = DateTime.Now;
        Console.WriteLine("Digital Clock: {0}:{1}:{2}",
            date.Hour, date.Minute, date.Second);
    }
}
```


Minh họa 1

- Lớp AnalogClock
 - Định nghĩa trình xử lý sự kiện của Clock
 - Đúng mô tả delegate hàm xử lý của lớp Clock
 - Thực hiện một số thao tác riêng của AnalogClock
 - Đăng ký xử lý với trình xử lý sự kiện trên khi có sự kiện OnSecondChange của Clock
 - Chức năng đăng ký với lớp Clock là có xử lý khi sự kiện OnSecondChange của Clock phát sinh
 - Ủy thác cho lớp Clock sẽ gọi trình xử lý định nghĩa bên trên của AnalogClock

Minh họa 1

```
public class AnalogClock
{
    public void Subscribe(Clock theClock)
    {
        theClock.OnSecondChange +=
            new Clock.SecondChangeHandler(Show);
    }
    public void Show(object obj, EventArgs args)
    {
        DateTime date = DateTime.Now;
        Console.WriteLine("Analog Clock: {0}:{1}:{2}",
            date.Hour,date.Minute,date.Second);
    }
}
```

Minh họa 1

- Minh họa cơ chế event

```
public class Tester {  
    public static void Main() {  
        Clock myClock = new Clock();  
        AnalogClock c1 = new AnalogClock();  
        DigitalClock c2 = new DigitalClock();  
        c1.Subscribe(myClock);  
        c2.Subscribe(myClock);  
        myClock.Run();  
    }  
}
```

Đăng ký xử lý sự kiện của đối tượng myClock!

Phát sinh sự kiện

Minh họa 1

```
public static void Main( ) {  
    Clock myClock = new Clock();  
    AnalogClock c1 = new AnalogClock();  
    DigitalClock c2 = new DigitalClock();
```

```
    myClock.OnSecondChange += new  
        Clock.SecondChangeHandler(c1.Show);  
    myClock.OnSecondChange += new  
        Clock.SecondChangeHandler(c2.Show);
```

```
    myClock.Run();
```

```
}
```

Được ko?



Minh họa EventArgs

- Hạn chế:
 - Mỗi lớp subscribing phải lấy giờ hiện hành riêng \Rightarrow trùng
 - Nên truyền tham số từ lớp publishing ngay khi phát sinh sự kiện
- Giải pháp:
 - Khi phát sinh sự kiện, truyền thời gian hiện hành \Rightarrow lớp subscribing sẽ sử dụng tham số này
 - Tạo một lớp TimeEventArgs kế thừa từ EventArgs: chứa thời gian sẽ truyền đi

Minh họa EventArgs

- Tạo lớp chứa tham số truyền cho trình xử lý sự kiện
 - Lớp dẫn xuất từ EventArgs
 - Chứa các thông tin về: giờ, phút, giây
- Bắt buộc phải dẫn xuất từ EventArgs
 - Do mô tả của trình xử lý sự kiện là tham số thứ 2 phải là lớp dẫn xuất từ EventArgs!

Minh họa EventArgs

```
public class TimeEventArgs : EventArgs  
{  
    public readonly int Second;  
    public readonly int Minute;  
    public readonly int Hour;  
    public TimeEventArgs(int s, int m, int h)  
    {  
        Second = s;  
        Minute = m;  
        Hour = h;  
    }  
}
```

Minh họa EventArgs

- Trong lớp Clock khai báo trình xử lý sự kiện như sau

```
public delegate void  
SecondChangeHandler(object obj, TimeEventArgs arg);
```




**Sử dụng tham số thứ hai có kiểu
TimeEventArgs**

Minh họa EventArgs

- Khi kích hoạt sự kiện thì truyền tham số {giờ, phút, giây}


```
public void Run( ) {  
    while (true) {  
        Thread.Sleep(1000);  
        if (OnSecondChange != null) {  
            DateTime date = DateTime.Now;  
  
            TimeEventArgs timeArg = new  
                TimeEventArgs(date.Second, date.Minute, date.Hour);  
            OnSecondChange(this, timeArg);  
        }  
    }  
}
```



Minh họa EventArgs

- Các lớp DigitalClock và AnalogClock: sử dụng tham số truyền vào

```
public class AnalogClock  
{  
    public void Subscribe(Clock theClock)  
    {  
        theClock.OnSecondChange +=  
            new Clock.SecondChangeHandler(Show);  
    }  
    public void Show(object obj, TimeEventArgs timeArg)  
    {  
        Console.WriteLine("Analog Clock: {0}:{1}:{2}",  
            timeArg.Hour, timeArg.Minute, timeArg.Second);  
    }  
}
```



Minh họa EventArgs

- Các phần khác còn lại tương tự như minh họa 1

Demo Event

Event

- Bài tập

Viết một chương trình đơn giản minh họa quản lý tài khoản ATM: khi rút tiền hoặc chuyển tiền thì hệ thống sẽ gọi tự động tin nhắn đến handphone của chủ tài khoản.

Hướng dẫn:

- Khi rút tiền hoặc chuyển tiền xong: phát sinh sự kiện “đã rút tiền” hoặc “đã chuyển tiền”

Tóm tắt

- Delegate
 - Cho phép tạo thể hiện
 - Tham chiếu đến một phương thức
 - Multi cast delegate tham chiếu đến nhiều phương thức
 - Multi cast là delegate có giá trị trả về phải là **void**
 - Các phương thức được ủy quyền phải thoả signature method của delegate
 - Khi delegate được gọi nó sẽ thực thi tất cả các phương thức được ủy quyền

Tóm tắt

- Event
 - Event được thực thi thông qua delegate
 - Thể hiện sự truyền thông qua lại
 - Lớp phát sinh sự kiện: publishing
 - Những lớp xử lý sự kiện: subscribing
 - Thành phần quan trọng trong GUIs

