

Событийно-ориентированное программирование

Делегаты, события, лямбда-функции

Юрий Литвинов
yurii.litvinov@gmail.com

5

Паттерн “Наблюдатель”

TODO: Картинка

Делегаты

```
public delegate void Feedback(Int32 value);
```

- ▶ Типобезопасный callback
- ▶ **delegate** объявляет тип функции-делегата
 - ▶ На самом деле, автоматически генерируемый класс
 - ▶ Наследник System.MulticastDelegate, который в свою очередь наследник System.Delegate
 - ▶ Методы Invoke, BeginInvoke, EndInvoke

Как это устроено

```
Feedback fbStatic = new Feedback(Program.FeedbackToConsole);  
Feedback fbInstance = new Feedback(new Program().FeedbackToFile);
```

TODO: Картинка

Пример использования

```
public delegate int HashFunction(string str, int hashSize);
```

```
private static class HashFunctions {  
    public static int Hash1(string str, int hashSize) {  
        return str[0] % hashSize;  
    }  
}
```

```
public static int Hash2(string str, int hashSize) {  
    int result = 0;  
    foreach (var ch in str)  
        result += ch;  
    return result % hashSize;  
}  
}
```

```
static void Main(string[] args) {  
    var h = new HashFunction(HashFunctions.Hash1);  
    var result = h("ololo", 10);  
}
```

Ещё пример, который будем дальше мучить

Цикл обработки событий

```
public class EventLoop {  
    public void Run() {  
        while (true) {  
            var key = Console.ReadKey();  
            switch (key.Key) {  
                case ConsoleKey.LeftArrow:  
                    // Сделать что-то по нажатию на "влево"  
                    break;  
                case ConsoleKey.RightArrow:  
                    // Сделать что-то по нажатию на "вправо"  
                    break;  
            }  
        }  
    }  
}
```

Цикл с делегатом

```
public delegate void ArrowHandler();

public class EventLoop {
    public void Run(ArrowHandler left, ArrowHandler right) {
        while (true) {
            var key = Console.ReadKey(true);
            switch (key.Key) {
                case ConsoleKey.LeftArrow:
                    left();
                    break;
                case ConsoleKey.RightArrow:
                    right();
                    break;
            }
        }
    }
}
```

Обработчики

```
public class Game
{
    public void OnLeft()
    {
        Console.WriteLine("Going left");
    }

    public void OnRight()
    {
        Console.WriteLine("Going right");
    }
}
```


Main

```
static void Main(string[] args)
{
    var eventLoop = new EventLoop();
    var game = new Game();
    eventLoop.Run(
        new ArrowHandler(game.OnLeft),
        new ArrowHandler(game.OnRight)
    );
}
```

Delegate chaining

```
public void Register(SomeDelegateType someDelegate)
{
    currentDelegate = Delegate.Combine(currentDelegate, someDelegate);
}
```

Или

```
public void Register(SomeDelegateType someDelegate)
{
    currentDelegate += someDelegate;
}
```

Цикл обработки событий с регистрацией (1)

```
public class EventLoop
{
    private ArrowHandler leftHandler;
    private ArrowHandler rightHandler;

    public void RegisterLeftHandler(ArrowHandler left)
    {
        leftHandler += left;
    }

    public void RegisterRightHandler(ArrowHandler right)
    {
        rightHandler += right;
    }
}
```

Цикл обработки событий с регистрацией (2)

```
public void Run() {  
    while (true) {  
        var key = Console.ReadKey(true);  
        switch (key.Key) {  
            case ConsoleKey.LeftArrow:  
                if (leftHandler != null)  
                    leftHandler();  
                break;  
            case ConsoleKey.RightArrow:  
                if (rightHandler != null)  
                    rightHandler();  
                break;  
        }  
    }  
}
```

Логгер (ещё один наблюдатель)

```
public class Logger
{
    private List<string> log = new List<string>();

    public void LeftPressed()
    {
        log.Add("left");
    }

    public void RightPressed()
    {
        log.Add("right");
    }
}
```

Main

```
static void Main(string[] args)
{
    var eventLoop = new EventLoop();
    var game = new Game();
    var logger = new Logger();

    eventLoop.RegisterLeftHandler(game.OnLeft);
    eventLoop.RegisterRightHandler(game.OnRight);

    eventLoop.RegisterLeftHandler(logger.LeftPressed);
    eventLoop.RegisterRightHandler(logger.RightPressed);

    eventLoop.Run();
}
```

Отписывание от событий

```
public void UnregisterLeftHandler(ArrowHandler left)
{
    leftHandler = (ArrowHandler) Delegate.Remove(leftHandler , left);
}
```

или

```
public void UnregisterLeftHandler(ArrowHandler left)
{
    leftHandler -= left;
}
```

Как примерно выглядит Invoke для цепочки

```
public Int32 Invoke(Int32 value) {  
    Int32 result;  
    Delegate[] delegateSet = _invocationList as Delegate[];  
    if (delegateSet != null) {  
        foreach (Feedback d in delegateSet)  
            result = d(value);  
    } else {  
        result = _methodPtr.Invoke(_target, value);  
    }  
    ^^|}  
  
    return result;  
}
```


Вызов делегатов из цепочки вручную

```
Delegate[] arrayOfDelegates = leftHandler.GetInvocationList();  
foreach (ArrowHandler handler in arrayOfDelegates) {  
    try {  
        handler.Invoke();  
    }  
    catch (InvalidOperationException e) {  
        Object component = handler.Target;  
        Console.WriteLine(  
            "Failed to call handler from {1}{2}{0} Error: {3}{0}{0}",  
            Environment.NewLine,  
            ((component == null) ? "" : component.GetType() + "."),  
            handler.GetMethodInfo().Name,  
            e.Message);  
    }  
}
```

Шаблонные типы делегатов

```
public delegate int HashFunction(string str, int hashSize);
```



```
Func<string, int, int>
```

- ▶ Func<T1, T2, ..., TResult>
- ▶ Action<T1, T2, ...>
- ▶ Не дружат с **ref** и **out**

События

`eventLoop.RegisterLeftHandler(game.OnLeft);` — много работы
`eventLoop.leftHandler += game.OnLeft;` — очень плохо.

Надо (в `EventLoop`):

```
public event Action LeftHandler;  
public event Action RightHandler;
```

и (в `Main`):

```
eventLoop.LeftHandler += game.OnLeft;  
eventLoop.RightHandler += game.OnRight;
```

```
eventLoop.LeftHandler += logger.LeftPressed;  
eventLoop.RightHandler += logger.RightPressed;
```

Анонимные методы

```
static void Main(string[] args) {  
    var eventLoop = new EventLoop();  
    var game = new Game();  
    eventLoop.LeftHandler += game.OnLeft;  
    eventLoop.RightHandler += game.OnRight;  
    var log = new List<string>();  
    eventLoop.LeftHandler += delegate {  
        log.Add("left");  
    };  
    eventLoop.RightHandler += delegate {  
        log.Add("right");  
    };  
    eventLoop.Run();  
}
```

Замыкания (1)

```
static Func<Point, Point> CreateRemapFunction(
    Rectangle rect1, Rectangle rect2)
{
    var xScale = rect2.Width() / rect1.Width();
    var yScale = rect2.Height() / rect1.Height();
    Func<Point, Point> result = delegate(Point point) {
        var returnValue = new Point();
        returnValue.x = point.x * xScale;
        returnValue.y = point.y * yScale;
        return returnValue;
    };
    return result;
}
```

Замыкания (2, пример использования)

```
static void Main(string[] args) {  
    var rect1 = new Rectangle() {  
        topLeft = new Point() { x = 0, y = 0 },  
        bottomRight = new Point() { x = 2, y = 2 }  
    };  
    var rect2 = new Rectangle() {  
        topLeft = new Point() { x = 0, y = 0 },  
        bottomRight = new Point() { x = 4, y = 6 }  
    };  
  
    var remap = CreateRemapFunction(rect1, rect2);  
    var point = new Point() { x = 1, y = 1 };  
    var transformedPoint = remap(point);  
    Console.WriteLine("Transformed point: x = {0}, y = {1}",  
        transformedPoint.x, transformedPoint.y);  
}
```

Лямбда-выражения

delegate(список параметров)

```
{  
    тело  
};
```



(список параметров) => { тело };

Примеры

```
eventLoop.LeftHandler += () => log.Add("left");  
eventLoop.RightHandler += () => log.Add("right");
```

```
Func<int, int> x2 = x => x * 2;  
var n = x2(1); // n будет 2
```

```
List<int> list = new List<int>() { 20, 1, 4, 8, 9, 44 };  
List<int> evenNumbers = list.FindAll(i => (i % 2) == 0);
```


Каноничное объявление события (1)

Наследник EventArgs, содержащий в себе параметры события

- ▶ Если параметров нет, то обработчики события всё равно должны принимать EventArgs
 - ▶ Передавать при вызове имеет смысл EventArgs.Empty

```
internal class NewMailEventArgs : EventArgs {  
    private readonly String from, to, subject;  
    public NewMailEventArgs(String from, String to, String subject) {  
        this.from = from; this.to = to; this.subject = subject;  
    }  
    public String From { get { return from; } }  
    public String To { get { return to; } }  
    public String Subject { get { return subject; } }  
}
```

Каноничное объявление события (2)

Само событие в наблюдаемом классе

- ▶ Инстанциация шаблона EventHandler
- ▶ **public delegate void** EventHandler<TEventArgs>(Object sender, TEventArgs e);

```
internal class MailManager {  
    public event EventHandler<NewMailEventArgs> NewMail;  
    ...  
}
```

Каноничное объявление события (3)

Вспомогательный метод, кидающий событие

- ▶ Сюда идёт проверка списка подписчиков на null
- ▶ Вызов лучше делать потокобезопасным

```
internal class MailManager {  
    ...  
    protected virtual void OnNewMail(NewMailEventArgs e) {  
        EventHandler<NewMailEventArgs> temp =  
            Volatile.Read(ref NewMail);  
        if (temp != null)  
            temp(this, e);  
    }  
    ...  
}
```

Каноничное объявление события (4)

Кидание события

- ▶ Создаём наследника EventArgs
- ▶ Вызываем метод, отправляющий событие наблюдателям

```
internal class MailManager {  
    public void SimulateNewMail(String from, String to, String subject) {  
        NewMailEventArgs e = new NewMailEventArgs(from, to, subject);  
        OnNewMail(e);  
    }  
}
```

Принимающий событие

```
internal sealed class Fax {  
    public Fax(MailManager mm) {  
        mm.NewMail += FaxMsg;  
    }  
  
    private void FaxMsg(Object sender, NewMailEventArgs e) {  
        Console.WriteLine("Faxing mail message:");  
        Console.WriteLine(" From={0}, To={1}, Subject={2}",  
            e.From, e.To, e.Subject);  
    }  
  
    public void Unregister(MailManager mm) {  
        mm.NewMail -= FaxMsg;  
    }  
}
```

Ручное управление подписчиками

```
public event EventHandler<FooEventArgs> Foo {  
    add { eventSet.Add(fooEventKey, value); }  
    remove { eventSet.Remove(fooEventKey, value); }  
}
```

```
protected virtual void OnFoo(FooEventArgs e) {  
    eventSet.Raise(fooEventKey, this, e);  
}
```

eventSet — рукописный словарь с цепочками обработчиков, не поместившийся в презентацию