Что может быть проще: делегаты и события

Михаил Щербаков
Independent Consultant

SPB DOT NET

MSK DOT NET

intelli

Обо мне

- Консультант, Upwork’er,
- Разработчик проекта IntelliDebugger http://intelliegg.com
- Координатор сообществ .NET программистов Москвы и Санкт-Петербурга http://mskdotnet.org/ http://spbdotnet.org
- В прошлом менеджер по продуктам и тимлид в Cezurity, Acronis, Luxoft, Boeing

О MSK .NET Community

http://mskdotnet.org

https://vk.com/mskdotnet

https://twitter.com/mskdotnet

https://facebook.com/mskdotnet

О делегатах

О делегатах

Делегат - **?...**

О делегатах

Делегат - это тип, который представляет собой ссылки на методы с определенным списком параметров и возвращаемым типом.

О делегатах

Делегат - это **ТИП**, который представляет собой ссылк**И** на методы с определенным списком параметров и возвращаемым типом.

О делегатах

```
public class Bar
{
    public delegate string Foo(int arg);
}
```

О делегатах

```
.class nested public sealed auto ansi Foo extends [mscorlib]System.MulticastDelegate
{
    .method public hidebysig specialname rtspecialname instance
      void .ctor(object @object, native int @method) runtime managed {}
    .method public hidebysig virtual newslot instance
      string Invoke(int32 arg) runtime managed {}
    .method public hidebysig virtual newslot instance class [mscorlib]System.IAsyncResult
      BeginInvoke(int32 arg, class [mscorlib]System.AsyncCallback callback,
        object @object) runtime managed {}
    .method public hidebysig virtual newslot instance
      string EndInvoke(class [mscorlib]System.IAsyncResult result) runtime managed {}
}
```

MulticastDelegate. Properties

	Name	Description
	Method	Gets the method represented by the delegate.(Inherited from Delegate.)
	Target	Gets the class instance on which the current delegate invokes the instance method.(Inherited from Delegate.)

MulticastDelegate. Methods

	Name	Description
	DynamicInvoke(Object[])	Dynamically invokes (late-bound) the method represented by the current delegate.(Inherited from Delegate.)
	GetInvocationList()	Returns the invocation list of this multicast delegate, in invocation order.(Overrides Delegate.GetInvocationList().)
S	CreateDelegate(Type, Object, String, Boolean, Boolean)	Creates a delegate of the specified type that represents the specified instance method to invoke on the specified class instance, with the specified case-sensitivity and the specified behavior on failure to bind.
S	CreateDelegate(Type, Type, String, Boolean, Boolean)	Creates a delegate of the specified type that represents the specified static method of the specified class, with the specified case-sensitivity and the specified behavior on failure to bind.

MulticastDelegate. Methods

	Name	Description
S	Combine(Delegate, Delegate)	Concatenates the invocation lists of two delegates.
S	Combine(Delegate[])	Concatenates the invocation lists of an array of delegates.
S	Remove(Delegate, Delegate)	Removes the last occurrence of the invocation list of a delegate from the invocation list of another delegate.
S	RemoveAll(Delegate, Delegate)	Removes all occurrences of the invocation list of a delegate from the invocation list of another delegate.

Invoke(…), .ctor(…)

```
.class nested public sealed auto ansi Foo extends [mscorlib]System.MulticastDelegate
{
    .method public hidebysig specialname rtspecialname instance
      void .ctor(object @object, native int @method) runtime managed {}
    .method public hidebysig virtual newslot instance
      string Invoke(int32 arg) runtime managed {}
    .method public hidebysig virtual newslot instance class [mscorlib]System.IAsyncResult
      BeginInvoke(int32 arg, class [mscorlib]System.AsyncCallback callback,
        object @object) runtime managed {}
    .method public hidebysig virtual newslot instance
      string EndInvoke(class [mscorlib]System.IAsyncResult result) runtime managed {}
}
```

О ~~проблемах~~ особенностях реализации делегатов

О проблемах делегатов

```
static void Main(string[] args)
{
    Action a = () => Console.Write("A");
    Action t = a;
    t -= a;
    t();                        //NRE
}
```

О проблемах делегатов

```
static void Main(string[] args)
{
    Action a = () => Console.Write("A");
    Action b = () => Console.Write("B");
    Action c = () => Console.Write("C");
    Action s = a + b + c + a + Console.WriteLine;
    s();                        //ABCA
    (s - a)();                  //ABC
    (s - a - a)();              //BC
    (s - (a + a))();            //ABCA
    (s - (b + c))();            //AA
}
```

О проблемах делегатов

```
struct Foo
{
    public void M() { Console.Write("uups!"); }
}

class Bar
{
    static event Action E = delegate { };
    static void Main()
    {
        var foo = new Foo();

        E += foo.M;
        E -= foo.M;

        E(); // ???
    }
}
```

О проблемах делегатов

```
// ClassLibrary.dll

public interface IFoo
{
    void Bar();
}

public class Foo
{
    public void Bar()
    {
        Console.WriteLine("Foo.Bar()");
    }
}
```

О проблемах делегатов

```
// ClassLibrary.dll

public interface IFoo
{
    void Bar();
}

public class Foo
{
    public void Bar()
    {
        Console.WriteLine("Foo.Bar()");
    }
}
```

```
// Application.exe

class DerivedFoo : Foo, IFoo
{
}

static class Boo
{
    private static event Action E = delegate { };
    private static void Main()
    {
        var foo = new DerivedFoo();
        IFoo ifoo = foo;

        E += foo.Bar;
        E -= ifoo.Bar;

        E(); // ???
    }
}
```

О проблемах делегатов

```
internal class DerivedFoo : Foo, IFoo
{
    // .method private final hidebysig virtual newslot instance
    void IFoo.Bar()
    {
        this.Bar();
    }
}
```

О проблемах делегатов. Exceptions

```
public void Foo(Action action)
{
    foreach (Action a in action.GetInvocationList())
    {
        try
        {
            a();
        }
        catch (Exception e)
        {
            // exception handler
        }
    }
}
```

О проблемах делегатов

Блог Александра Шведова

- http://controlflow.github.io/2011/11/14/delegate-equality-proxy.html
- http://controlflow.github.io/2011/10/24/delegate-equality-valuetype.html
- http://controlflow.github.io/2011/10/24/delegate-equality-base.html

О событиях

О событиях

События это член, который позволяет классу или объекту получать уведомления.

https://msdn.microsoft.com/en-us/library/aa664454(v=vs.71)

О событиях

```
public class Bar
{
    public event Foo FooEvent;
}
```

О событиях

```
public class Bar
{
    private Foo FooEvent;

    public event Foo FooEvent
    {
        add { ... }
        remove { ... }
    }
}
```

Auto-Implemented Properties

```
public class Bar
{
    public string FooProperty { get; set; }
}
```

WTF?!!

```
public class Bar
{
    public string FooProperty { get; set; }
}
```

```
public class Bar
{
    public event Foo FooEvent;
}
```

Field-like Events. C# 3

```
public event Foo FooEvent
{
    add
    {
        lock (this)
        {
            this.FooEvent += value;
        }
    }
```

Field-like Events. C# 4

```
public event Foo FooEvent
{
    add
    {
        Foo foo = this.FooEvent;
        Foo comparand;
        do
        {
            comparand = foo;
            foo = Interlocked.CompareExchange<Foo>(ref this.FooEvent,
                (Foo)Delegate.Combine(comparand, value),
                comparand);
        }
        while (foo != comparand);
    }
```

Field-like Events. Synchronization

```
class Bar
{
    public event EventHandler E;
    public void UseE(EventHandler handler)
    {
        // Unsafe in C# 3, safe in C# 4!
        E += handler;
    }
}
```

Field-like Events. Synchronization

```
class Bar
{
    public event EventHandler E;
    public void UseE(EventHandler handler)
    {
        lock (this)
        {
            // Safe in C# 3, unsafe in C# 4!
            E = E + handler;
        }
    }
}
```

Raise Event

```
class Bar
{
    public event Action Foo;

    private void RaiseFoo()
    {
        if (Foo != null)
        {
            Foo();
        }
    }
}
```

Raise Event

```
class Bar
{
    public event Action Foo;

    private void RaiseFoo()
    {
        Foo?.Invoke();
    }
}
```

Raise Event

```
class Bar
{
    public event Action Foo;

    protected void RaiseFoo()
    {
        var handler = Foo;
        if (handler != null)
        {
            handler();
        }
    }
}
```

Raise Event

```
class Bar
{
    public event Action Foo = delegate {};

    private void RaiseFoo()
    {
        Foo();
    }
}
```

Custom Event Accessors

```
public class SampleControl : Component
{
    private static readonly object MouseDownEventKey = new object();

    protected EventHandlerList ListEventDelegates = new EventHandlerList();

    public event MouseEventHandler MouseDown
    {
        add
        {
            ListEventDelegates.AddHandler(MouseDownEventKey, value);
        }
        remove
        {
            ListEventDelegates.RemoveHandler(MouseDownEventKey, value);
        }
    }

    protected void OnMouseDown(MouseEventArgs e)
    {
        var mouseEventDelegate =
            (MouseEventHandler)ListEventDelegates[MouseDownEventKey];
        mouseEventDelegate(this, e);
    }
}
```

https://msdn.microsoft.com/en-us/library/8843a9ch.aspx

Custom Event Accessors

```
public class SampleControl : Component
{
    private static readonly object MouseDownEventKey = new object();

    protected EventHandlerList ListEventDelegates = new EventHandlerList();

    public event MouseEventHandler MouseDown
    {
        add
        {
            ListEventDelegates.AddHandler(MouseDownEventKey, value);
        }
        remove
        {
            ListEventDelegates.RemoveHandler(MouseDownEventKey, value);
        }
    }

    protected void OnMouseDown(MouseEventArgs e)
    {
        var mouseEventDelegate =
            (MouseEventHandler)ListEventDelegates[MouseDownEventKey];
        mouseEventDelegate(this, e);
    }
}
```

https://msdn.microsoft.com/en-us/library/8843a9ch.aspx

Single-threaded events

```
private EventHandler myEventField;

public event EventHandler MyEvent
{
    add { myEventField += value; }
    remove { myEventField -= value; }
}

protected void OnMyEvent(EventArgs e)
{
    myEventField?.Invoke(this, e);
}
```

О проблемах событий

Virtual Events

1) Use a virtual method for triggering the event.
2) If you need to override virtual events, write your own handlers.

https://blogs.msdn.microsoft.com/samng/2007/11/26/virtual-events-in-c/

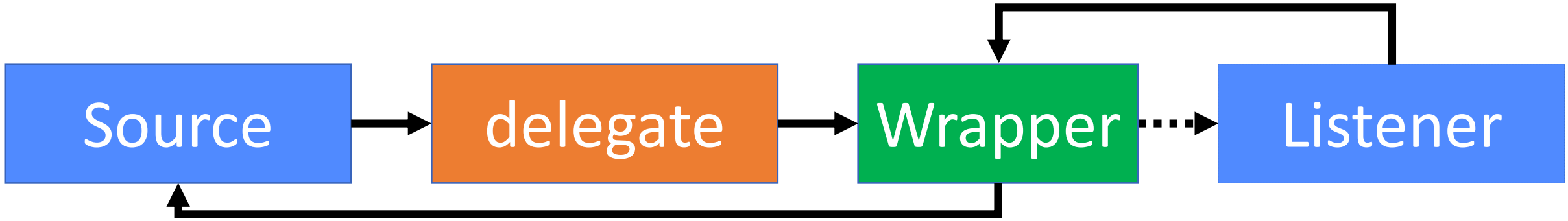
No-op Events

```
interface IBar
{
    event EventHandler MyEvent;
}

class Bar : IBar
{
    event EventHandler IBar.MyEvent
    {
        add { }
        remove { }
    }
}
```

Memory Leak

Memory Leak. Listener-side Fix



Memory Leak. Listener-side Fix

Use

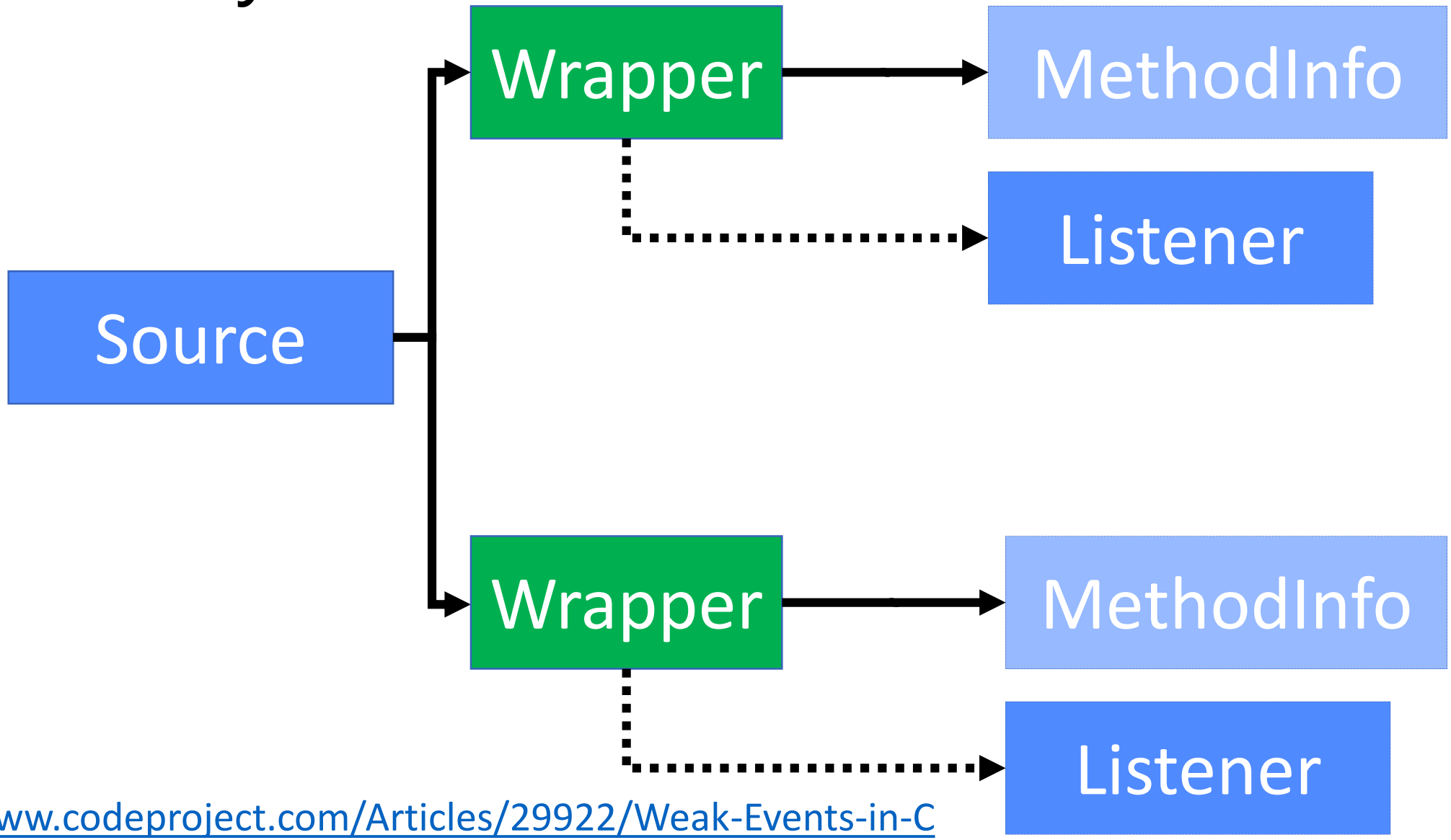
- WeakEventManager
- PropertyChangedEventManager
- …
- WeakEventManager<TEventSource, TEventArgs>

Memory Leak. Listener-side Fix

```
public class WeakEventHandler<TTarget, TEventArgs> : IDisposable
    where TTarget : class where TEventArgs : EventArgs
{
    private readonly WeakReference<TTarget> targetReference;
    private Action<TTarget, object, TEventArgs> targetDelegate;

    public WeakEventHandler(EventHandler<TEventArgs> callback)
    {
        targetReference = new WeakReference<TTarget>((TTarget)callback.Target, true);
        targetDelegate = (Action<TTarget, object, TEventArgs>)Delegate.CreateDelegate(
            typeof(Action<TTarget, object, TEventArgs>), callback.Method, true);
    }
```

Memory Leak. Source-side Fix



http://www.codeproject.com/Articles/29922/Weak-Events-in-C

Memory Leak. Listener-side Fix

```
public class WeakEventHandler<TTarget, TEventArgs> : IDisposable
    where TTarget : class where TEventArgs : EventArgs
{
    private readonly WeakReference<TTarget> targetReference;
    private Action<TTarget, object, TEventArgs> targetDelegate;

    public WeakEventHandler(EventHandler<TEventArgs> callback)
    {
        targetReference = new WeakReference<TTarget>((TTarget)callback.Target, true);
        targetDelegate = (Action<TTarget, object, TEventArgs>)Delegate.CreateDelegate(
            typeof(Action<TTarget, object, TEventArgs>), callback.Method, true);
    }
```

Порефлексируем…

Observer Pattern

```
public interface IObservable<out T>
{
    IDisposable Subscribe(IObserver<T> observer);
}
```

Observer Pattern

```
public interface IObserver<in T>
{
    void OnNext(T value);
    void OnError(Exception error);
    void OnCompleted();
}
```

Reactive Extensions

```
var textChangedObservable = Observable
    .FromEventPattern<TextChangedEventArgs>(textBox, "TextChanged")
    .Select(evt => ((TextBox)evt.Sender).Text)
    .Throttle(TimeSpan.FromMilliseconds(200))
    .Select(timestampedText => timestampedText.Value)
    .Where(text => text != null && text.Length >= 4)
    .DistinctUntilChanged();
```

Спасибо за внимание!

Михаил Щербаков

Independent Consultant

intelliegg.com

spbdotnet.org

github.com/yuske

@yu5k3