**在派生类中引发基类事件（C# 编程指南）**

以下简单示例演示了在基类中声明可从派生类引发的事件的标准方法。此模式广泛应用于 .NET Framework 类库中的 Windows 窗体类。

在创建可用作其他类的基类的类时，应考虑如下事实：事件是特殊类型的委托，只可以从声明它们的类中调用。派生类无法直接调用基类中声明的事件。尽管有时需要事件仅由基类引发，但在大多数情形下，应该允许派生类调用基类事件。为此，您可以在包含该事件的基类中创建一个受保护的调用方法。通过调用或重写此调用方法，派生类便可以间接调用该事件。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 不要在基类中声明虚拟事件，也不要在派生类中重写这些事件。C# 编译器无法正确处理这些事件，并且无法预知的该派生的事件的用户是否真正订阅了基类事件。 |

namespace BaseClassEvents

{

using System;

using System.Collections.Generic;

// Special EventArgs class to hold info about Shapes.

public class ShapeEventArgs : EventArgs

{

private double newArea;

public ShapeEventArgs(double d)

{

newArea = d;

}

public double NewArea

{

get { return newArea; }

}

}

// Base class event publisher

public abstract class Shape

{

protected double area;

public double Area

{

get { return area; }

set { area = value; }

}

// The event. Note that by using the generic EventHandler<T> event type

// we do not need to declare a separate delegate type.

public event EventHandler<ShapeEventArgs> ShapeChanged;

public abstract void Draw();

//The event-invoking method that derived classes can override.

protected virtual void OnShapeChanged(ShapeEventArgs e)

{

// Make a temporary copy of the event to avoid possibility of

// a race condition if the last subscriber unsubscribes

// immediately after the null check and before the event is raised.

EventHandler<ShapeEventArgs> handler = ShapeChanged;

if (handler != null)

{

handler(this, e);

}

}

}

public class Circle : Shape

{

private double radius;

public Circle(double d)

{

radius = d;

area = 3.14 \* radius \* radius;

}

public void Update(double d)

{

radius = d;

area = 3.14 \* radius \* radius;

OnShapeChanged(new ShapeEventArgs(area));

}

protected override void OnShapeChanged(ShapeEventArgs e)

{

// Do any circle-specific processing here.

// Call the base class event invocation method.

base.OnShapeChanged(e);

}

public override void Draw()

{

Console.WriteLine("Drawing a circle");

}

}

public class Rectangle : Shape

{

private double length;

private double width;

public Rectangle(double length, double width)

{

this.length = length;

this.width = width;

area = length \* width;

}

public void Update(double length, double width)

{

this.length = length;

this.width = width;

area = length \* width;

OnShapeChanged(new ShapeEventArgs(area));

}

protected override void OnShapeChanged(ShapeEventArgs e)

{

// Do any rectangle-specific processing here.

// Call the base class event invocation method.

base.OnShapeChanged(e);

}

public override void Draw()

{

Console.WriteLine("Drawing a rectangle");

}

}

// Represents the surface on which the shapes are drawn

// Subscribes to shape events so that it knows

// when to redraw a shape.

public class ShapeContainer

{

List<Shape> \_list;

public ShapeContainer()

{

\_list = new List<Shape>();

}

public void AddShape(Shape s)

{

\_list.Add(s);

// Subscribe to the base class event.

s.ShapeChanged += HandleShapeChanged;

}

// ...Other methods to draw, resize, etc.

private void HandleShapeChanged(object sender, ShapeEventArgs e)

{

Shape s = (Shape)sender;

// Diagnostic message for demonstration purposes.

Console.WriteLine("Received event. Shape area is now {0}", e.NewArea);

// Redraw the shape here.

s.Draw();

}

}

class Test

{

static void Main(string[] args)

{

//Create the event publishers and subscriber

Circle c1 = new Circle(54);

Rectangle r1 = new Rectangle(12, 9);

ShapeContainer sc = new ShapeContainer();

// Add the shapes to the container.

sc.AddShape(c1);

sc.AddShape(r1);

// Cause some events to be raised.

c1.Update(57);

r1.Update(7, 7);

// Keep the console window open in debug mode.

System.Console.WriteLine("Press any key to exit.");

System.Console.ReadKey();

}

}

}

/\* Output:

Received event. Shape area is now 10201.86

Drawing a circle

Received event. Shape area is now 49

Drawing a rectangle

\*/