事件：从本质上来说，事件是一种信号机制，在某些操作发生时自动地发出通知。

如果对某个事件感兴趣，则对象为该事件注册一个事件处理程序，在事件发生时，会调用所有注册在此事件上的处理程序。通过委托表示事件处理程序。

常用的事件声明：

event event-delegate event-name;

例：程序event\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 事件

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace event\_test1

{

delegate void MyEventHandler();

class MyEvent

{

public event MyEventHandler SomeEvent;

public void OnSomeEvent()

{

if (SomeEvent != null)

{

SomeEvent();

}

}

}

class EventTest

{

// 事件处理程序，需要与事件的委托匹配

static void Handler()

{

*Console*.*WriteLine*("Event occurred.");

}

static void Main(string[] args)

{

MyEvent evt = new MyEvent();

// 添加事件处理程序到事件链表

evt.SomeEvent += Handler;

// 触发事件

evt.OnSomeEvent();

}

}

}

多播委托事件：

例：程序event\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 多播委托事件

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace event\_test2

{

delegate void MyEventHandler();

class MyEvent

{

public event MyEventHandler SomeEvent;

public void OnSomeEvent()

{

if (SomeEvent != null)

{

SomeEvent();

}

}

}

class X

{

public void Xhandler()

{

*Console*.*WriteLine*("Event received by X object.");

}

}

class Y

{

public void Yhandler()

{

*Console*.*WriteLine*("Event received by Y object.");

}

}

class EventTest

{

static void Handler()

{

*Console*.*WriteLine*("Event received by Test.");

}

static void Main(string[] args)

{

MyEvent evt = new MyEvent();

X obj\_x = new X();

Y obj\_y = new Y();

// 多播委托的事件

evt.SomeEvent += Handler;

evt.SomeEvent += obj\_x.Xhandler;

evt.SomeEvent += obj\_y.Yhandler;

evt.OnSomeEvent();

}

}

}

输出结果：

Event received by Test.

Event received by X object.

Event received by Y object.

实例方法和static方法都能够作为事件处理程序，使用实例方法时，事件通知将发送给特定的对象实例，如果想接收事件通知，每个类对象都必须单独注册。使用static方法，事件处理就不与任何实例相关。

事件存取器：多线程中经常使用。

可以在接口中指定事件，实现该接口的类必须提供事件。

事件可以为abstract类型，派生类必须实现该事件。但基于存取器的事件不可以为abstract类型。

事件可以指定为sealed类型。

事件可以是virtual事件，可以在派生类中重新该事件。

.NET事件指导原则

C#允许编写各种类型的事件，但为了与.NET Framwork组件兼容，必须遵守Microsoft的指导原则。事件处理程序必须有两个参数。第一个参数是一个引用，指向产生该事件的对象。第二个参数为EventArgs类型，包含处理程序所需的其它信息。与.NET兼容的事件处理程序：

void handler(object sender, EventArgs e) {

}

例：程序event\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 与.NET兼容的事件处理

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace event\_test3

{

class MyEventArgs : *EventArgs*

{

public int EventNumber;

}

delegate void MyEventHandler(object sender, MyEventArgs e);

class MyEvent

{

static int count = 0;

// 声明一个事件

public event MyEventHandler SomeEvent;

public void OnSomeEvent()

{

MyEventArgs arg = new MyEventArgs();

if (SomeEvent != null)

{

arg.EventNumber = count++;

SomeEvent(this, arg);

}

}

}

class X

{

public void Handler(object sender, MyEventArgs e)

{

*Console*.*WriteLine*("Event " + e.EventNumber

+ " received by an X object.");

*Console*.*WriteLine*("Source is " + sender);

*Console*.*WriteLine*();

}

}

class EventTest

{

static void Main(string[] args)

{

X obj = new X();

MyEvent evt = new MyEvent();

evt.SomeEvent += obj.Handler;

evt.OnSomeEvent();

}

}

}

使用EventHandler<TEventArgs>和EventHandler委托：

EventHandler<TEventArgs>是一内置泛型委托，TEventArgs为指定传递给事件的EventArgs实参类型。

EventHandler是一个非泛型委托。

例：程序event\_test4

// Copyright 2016.刘珅珅

// author：刘珅珅

// 使用EventHandler委托

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace event\_test4

{

class MyEvent

{

public event *EventHandler* SomeEvent;

public void OnSomeEvent()

{

if (SomeEvent != null)

{

SomeEvent(this, *EventArgs*.*Empty*);

}

}

}

class EventTest

{

static void Handler(object sender, *EventArgs* e)

{

*Console*.*WriteLine*("Event occurred.");

*Console*.*WriteLine*("Source is " + sender);

}

static void Main(string[] args)

{

MyEvent evt = new MyEvent();

// 注册事件处理程序到事件链表

evt.SomeEvent += Handler;

evt.OnSomeEvent();

}

}

}

事件的简单应用：

例：程序event\_test5

// Copyright 2016.刘珅珅

// author：刘珅珅

// 事件的应用

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace event\_test5

{

class KeyEventArgs : *EventArgs*

{

public char ch;

}

class KeyEvent

{

public event *EventHandler*<KeyEventArgs> KeyPress;

public void OnKeyPress(char key)

{

KeyEventArgs k = new KeyEventArgs();

if (KeyPress != null)

{

k.ch = key;

KeyPress(this, k);

}

}

}

class EventTest

{

static void Main(string[] args)

{

KeyEvent kevt = new KeyEvent();

*ConsoleKeyInfo* key;

int count = 0;

kevt.KeyPress += (sender, e) =>

*Console*.*WriteLine*(" Received keystroke: " + e.ch);

kevt.KeyPress += (sender, e) =>

count++;

*Console*.*WriteLine*("Enter some characters.");

do

{

key = *Console*.*ReadKey*();

kevt.OnKeyPress(key.*KeyChar*);

} while (key.*KeyChar* != '.');

*Console*.*WriteLine*(count + " keys press.");

}

}

}

输出结果为：

Enter some characters.

t Received keystroke: t

e Received keystroke: e

s Received keystroke: s

t Received keystroke: t

. Received keystroke: .

5 keys press.

在键盘上点击了test.5个按键，从结果中可以看出：count变量是一个局部变量，每次点击按键会自动加1，最后的结果是count = 5.参考pdf中438页，在匿名方法中使用的外部变量，在Lambda中类似。count变量尽管是个局部变量，但其作用域包含Lambda表达式，对Lambda表达式而言其是一个外部变量。在Lambda表达式中使用外部变量时，就表示捕获了这个外部变量，捕获的外部变量将一直存在，直到垃圾回收捕获到它的委托。对于Lambda表达式而言，其就相当于一个全局变量的作用。