**参考文档1**

如果想编写引发事件时调用的自定义代码，则可以订阅由其他类发布的事件。 例如，可以订阅某个按钮的 click 事件，以使应用程序在用户单击该按钮时执行一些有用的操作。

### 使用 Visual Studio IDE 订阅事件

1. 如果看不到“属性”窗口，请在“设计”视图中，右键单击要为其创建事件处理程序的窗体或控件，然后选择“属性” 。
2. 在“属性”窗口的顶部，单击“事件”图标 。
3. 双击要创建的事件，例如 Load 事件。

Visual C# 会创建一个空事件处理程序方法，并将其添加到你的代码中。 或者，也可以在“代码”视图中手动添加代码。 例如，下面的代码行声明了一个在 Form 类引发 Load 事件时调用的事件处理程序方法。

C#复制

private void Form1\_Load(object sender, System.EventArgs e)

{

// Add your form load event handling code here.

}

还会在项目的 Form1.Designer.cs 文件的 InitializeComponent 方法中自动生成订阅该事件所需的代码行。 该代码行类似于：

C#复制

this.Load += new System.EventHandler(this.Form1\_Load);

### 以编程方式订阅事件

1. 定义一个事件处理程序方法，其签名与该事件的委托签名匹配。 例如，如果事件基于 [EventHandler](https://learn.microsoft.com/zh-cn/dotnet/api/system.eventhandler) 委托类型，则下面的代码表示方法存根：

C#复制

void HandleCustomEvent(object sender, CustomEventArgs a)

{

// Do something useful here.

}

1. 使用加法赋值运算符 (+=) 来为事件附加事件处理程序。 在下面的示例中，假设名为 publisher 的对象拥有一个名为 RaiseCustomEvent 的事件。 请注意，订户类需要引用发行者类才能订阅其事件。

C#复制

publisher.RaiseCustomEvent += HandleCustomEvent;

还可以使用 [Lambda 表达式](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/lambda-expressions)来指定事件处理程序：

C#复制

public Form1()

{

InitializeComponent();

this.Click += (s,e) =>

{

MessageBox.Show(((MouseEventArgs)e).Location.ToString());

};

}

### 使用匿名函数订阅事件

如果以后不必取消订阅某个事件，则可以使用加法赋值运算符 (+=) 将匿名函数作为事件处理程序进行附加。 在下面的示例中，假设名为 publisher 的对象拥有一个名为 RaiseCustomEvent 的事件，并且还定义了一个 CustomEventArgs 类以承载某些类型的专用事件信息。 请注意，订户类需要引用 publisher 才能订阅其事件。

C#复制

publisher.RaiseCustomEvent += (object o, CustomEventArgs e) =>

{

string s = o.ToString() + " " + e.ToString();

Console.WriteLine(s);

};

如果使用匿名函数订阅事件，事件的取消订阅过程将比较麻烦。 这种情况下若要取消订阅，请返回到该事件的订阅代码，将该匿名函数存储在委托变量中，然后将此委托添加到该事件中。 如果必须在后面的代码中取消订阅某个事件，则建议不要使用匿名函数订阅此事件。 有关匿名函数的详细信息，请参阅 [Lambda 表达式](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/lambda-expressions)。

## 取消订阅

若要防止在引发事件时调用事件处理程序，请取消订阅该事件。 为了防止资源泄露，应在释放订户对象之前取消订阅事件。 在取消订阅事件之前，在发布对象中作为该事件的基础的多播委托会引用封装了订户的事件处理程序的委托。 只要发布对象保持该引用，垃圾回收功能就不会删除订户对象。

#### 取消订阅事件

* 使用减法赋值运算符 (-=) 取消订阅事件：

C#复制

publisher.RaiseCustomEvent -= HandleCustomEvent;

所有订户都取消订阅事件后，发行者类中的事件实例将设置为 null。

### 请参阅

* [事件](https://learn.microsoft.com/zh-cn/dotnet/csharp/programming-guide/events/)
* [event](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/keywords/event)
* [如何发布符合 .NET 准则的事件](https://learn.microsoft.com/zh-cn/dotnet/standard/events)
* [- 和 -= 运算符](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/subtraction-operator)
* [+ 和 += 运算符](https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/addition-operator)

# 参考文档2

## c#3.0参考手册: <https://learn.microsoft.com/zh-cn/previous-versions/visualstudio/visual-studio-2008/ff518995(v=orm.10)>

# 参考文档3

**通过事件使用委托**

事件在类中声明且生成，且通过使用同一个类或其他类中的委托与事件处理程序关联。包含事件的类用于发布事件。这被称为 **发布器（publisher）** 类。其他接受该事件的类被称为 **订阅器（subscriber）** 类。事件使用 **发布-订阅（publisher-subscriber）** 模型。

**发布器（publisher）** 是一个包含事件和委托定义的对象。事件和委托之间的联系也定义在这个对象中。发布器（publisher）类的对象调用这个事件，并通知其他的对象。

**订阅器（subscriber）** 是一个接受事件并提供事件处理程序的对象。在发布器（publisher）类中的委托调用订阅器（subscriber）类中的方法（事件处理程序）。

**声明事件（Event）**

在类的内部声明事件，首先必须声明该事件的委托类型。例如：

public delegate void BoilerLogHandler(string status);

然后，声明事件本身，使用 **event** 关键字：

// 基于上面的委托定义事件

public event BoilerLogHandler BoilerEventLog;

上面的代码定义了一个名为 *BoilerLogHandler* 的委托和一个名为 *BoilerEventLog* 的事件，该事件在生成的时候会调用委托。

以下示例展示了如何在 C# 中使用事件：

**实例**

**using** System;  
  
**namespace** EventDemo  
{  
    *// 定义一个委托类型，用于事件处理程序*  
    **public** **delegate** **void** NotifyEventHandler(**object** sender, EventArgs e);  
  
    *// 发布者类*  
    **public** **class** ProcessBusinessLogic  
    {  
        *// 声明事件*  
        **public** **event** NotifyEventHandler ProcessCompleted;  
  
        *// 触发事件的方法*  
        **protected** **virtual** **void** OnProcessCompleted(EventArgs e)  
        {  
            ProcessCompleted?.Invoke(**this**, e);  
        }  
  
        *// 模拟业务逻辑过程并触发事件*  
        **public** **void** StartProcess()  
        {  
            Console.WriteLine("Process Started!");  
  
            *// 这里可以加入实际的业务逻辑*  
  
            *// 业务逻辑完成，触发事件*  
            OnProcessCompleted(EventArgs.Empty);  
        }  
    }  
  
    *// 订阅者类*  
    **public** **class** EventSubscriber  
    {  
        **public** **void** Subscribe(ProcessBusinessLogic process)  
        {  
            process.ProcessCompleted += Process\_ProcessCompleted;  
        }  
  
        **private** **void** Process\_ProcessCompleted(**object** sender, EventArgs e)  
        {  
            Console.WriteLine("Process Completed!");  
        }  
    }  
  
    **class** Program  
    {  
        **static** **void** Main(**string**[] args)  
        {  
            ProcessBusinessLogic process = new ProcessBusinessLogic();  
            EventSubscriber subscriber = new EventSubscriber();  
  
            *// 订阅事件*  
            subscriber.Subscribe(process);  
  
            *// 启动过程*  
            process.StartProcess();  
  
            Console.ReadLine();  
        }  
    }  
}

**说明**

1、定义委托类型：

public delegate void NotifyEventHandler(object sender, EventArgs e);

这是一个委托类型，它定义了事件处理程序的签名。通常使用 EventHandler 或 EventHandler<TEventArgs> 来替代自定义的委托。

2、声明事件：

public event NotifyEventHandler ProcessCompleted;

这是一个使用 NotifyEventHandler 委托类型的事件。

3、触发事件：

protected virtual void OnProcessCompleted(EventArgs e)

{

ProcessCompleted?.Invoke(this, e);

}

这是一个受保护的方法，用于触发事件。使用 ?.Invoke 语法来确保只有在有订阅者时才调用事件。

4、订阅和取消订阅事件：

process.ProcessCompleted += Process\_ProcessCompleted;

订阅者使用 += 运算符订阅事件，并定义事件处理程序 Process\_ProcessCompleted。

**实例**

**实例 1**

**using** System;  
**namespace** SimpleEvent  
{  
  **using** System;  
  */\*\*\*\*\*\*\*\*\*\*\*发布器类\*\*\*\*\*\*\*\*\*\*\*/*  
  **public** **class** EventTest  
  {  
    **private** **int** **value**;  
  
    **public** **delegate** **void** NumManipulationHandler();  
  
  
    **public** **event** NumManipulationHandler ChangeNum;  
    **protected** **virtual** **void** OnNumChanged()  
    {  
      **if** ( ChangeNum != **null** )  
      {  
        ChangeNum(); */\* 事件被触发 \*/*  
      }**else** {  
        Console.WriteLine( "event not fire" );  
        Console.ReadKey(); */\* 回车继续 \*/*  
      }  
    }  
  
  
    **public** EventTest()  
    {  
      **int** n = 5;  
      SetValue( n );  
    }  
  
  
    **public** **void** SetValue( **int** n )  
    {  
      **if** ( **value** != n )  
      {  
        **value** = n;  
        OnNumChanged();  
      }  
    }  
  }  
  
  
  */\*\*\*\*\*\*\*\*\*\*\*订阅器类\*\*\*\*\*\*\*\*\*\*\*/*  
  
  **public** **class** subscribEvent  
  {  
    **public** **void** printf()  
    {  
      Console.WriteLine( "event fire" );  
      Console.ReadKey(); */\* 回车继续 \*/*  
    }  
  }  
  
  */\*\*\*\*\*\*\*\*\*\*\*触发\*\*\*\*\*\*\*\*\*\*\*/*  
  **public** **class** MainClass  
  {  
    **public** **static** **void** Main()  
    {  
      EventTest e = new EventTest(); */\* 实例化对象,第一次没有触发事件 \*/*  
      subscribEvent v = new subscribEvent(); */\* 实例化对象 \*/*  
      e.ChangeNum += new EventTest.NumManipulationHandler( v.printf ); */\* 注册 \*/*  
      e.SetValue( 7 );  
      e.SetValue( 11 );  
    }  
  }  
}

当上面的代码被编译和执行时，它会产生下列结果：

event not fire

event fire

event fire

本实例提供一个简单的用于热水锅炉系统故障排除的应用程序。当维修工程师检查锅炉时，锅炉的温度和压力会随着维修工程师的备注自动记录到日志文件中。

**实例 2**

**using** System;  
**using** System.IO;  
  
**namespace** BoilerEventAppl  
{  
    *// Boiler 类*  
    **class** Boiler  
    {  
        **public** **int** Temp { **get**; **private** **set**; }  
        **public** **int** Pressure { **get**; **private** **set**; }  
  
        **public** Boiler(**int** temp, **int** pressure)  
        {  
            Temp = temp;  
            Pressure = pressure;  
        }  
    }  
  
    *// 事件发布器*  
    **class** DelegateBoilerEvent  
    {  
        **public** **delegate** **void** BoilerLogHandler(**string** status);  
  
        *// 基于上面的委托定义事件*  
        **public** **event** BoilerLogHandler BoilerEventLog;  
  
        **public** **void** LogProcess()  
        {  
            **string** remarks = "O.K.";  
            Boiler boiler = new Boiler(100, 12);  
            **int** temp = boiler.Temp;  
            **int** pressure = boiler.Pressure;  
  
            **if** (temp > 150 || temp < 80 || pressure < 12 || pressure > 15)  
            {  
                remarks = "Need Maintenance";  
            }  
  
            OnBoilerEventLog($"Logging Info:**\n**Temperature: {temp}**\n**Pressure: {pressure}**\n**Message: {remarks}");  
        }  
  
        **protected** **void** OnBoilerEventLog(**string** message)  
        {  
            BoilerEventLog?.Invoke(message);  
        }  
    }  
  
    *// 该类保留写入日志文件的条款*  
    **class** BoilerInfoLogger : IDisposable  
    {  
        **private** **readonly** StreamWriter \_streamWriter;  
  
        **public** BoilerInfoLogger(**string** filename)  
        {  
            \_streamWriter = new StreamWriter(new FileStream(filename, FileMode.Append, FileAccess.Write));  
        }  
  
        **public** **void** Logger(**string** info)  
        {  
            \_streamWriter.WriteLine(info);  
        }  
  
        **public** **void** Dispose()  
        {  
            \_streamWriter?.Close();  
        }  
    }  
  
    *// 事件订阅器*  
    **public** **class** RecordBoilerInfo  
    {  
        **static** **void** Logger(**string** info)  
        {  
            Console.WriteLine(info);  
        }  
  
        **static** **void** Main(**string**[] args)  
        {  
            **using** (BoilerInfoLogger fileLogger = new BoilerInfoLogger("e:**\\**boiler.txt"))  
            {  
                DelegateBoilerEvent boilerEvent = new DelegateBoilerEvent();  
                boilerEvent.BoilerEventLog += Logger;  
                boilerEvent.BoilerEventLog += fileLogger.Logger;  
                boilerEvent.LogProcess();  
            }  
  
            Console.ReadLine();  
        }  
    }  
}

当上面的代码被编译和执行时，它会产生下列结果：

Logging info:

Temperature 100

Pressure 12

Message: O. K