# 事件

[类](https://msdn.microsoft.com/zh-cn/library/0b0thckt.aspx)或对象可以通过事件向其他类或对象通知发生的相关事情。

发送（或引发）事件的类称为“发行者”

接收（或处理）事件的类称为“订户”。

没有订户的事件永远也不会引发。

事件通常用于通知用户操作，例如，图形用户界面中的按钮单击或菜单选择操作。

如果一个事件有多个订户，当引发该事件时，会同步调用多个事件处理程序。

在 .NET Framework 类库中，事件是基于 [EventHandler](https://msdn.microsoft.com/zh-cn/library/system.eventhandler.aspx) 委托和 [EventArgs](https://msdn.microsoft.com/zh-cn/library/system.eventargs.aspx) 基类的。

一处触发,全局响应,实现方式:

发行者声明回调函数容器一个

若干事件类型

发行者触发事件方法若干,(或是一个,这时传递的参数包含事件类型)

订阅者向发行者添加回调函数

订阅者使用发行者引用执行某发行者触发事件方法.

## 订阅事件方式

设计视图中,鼠标操作添加

代码添加

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

lambda表达式

this.Click += (s,e) => { MessageBox.Show(

((MouseEventArgs)e).Location.ToString());};

请务必注意，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。

## 取消订阅

publisher.RaiseCustomEvent -= HandleCustomEvent;

## 自定义事件

基本情况

1.创建自己的事件

public class CustomEventArgs : EventArgs

{

public CustomEventArgs(string s)

{

msg = s;

}

private string msg;

public string Message

{

get { return msg; }

}

}

2.在发布类中声明委托

供订阅者添加回调函数;

public delegate void CustomEventHandler(object sender, CustomEventArgs a);

//如果直接使用EventHandler ,那么不需要声明

3.声明一个事件字段;或是泛型;

=====

无自定义事件

则事件类型: 非泛型 EventHandler 委托,  无需声明委托，因为它已在创建 C# 项目时包含的 [System](https://msdn.microsoft.com/zh-cn/library/system.aspx) 命名空间中进行了声明

直接使用,保存回调函数

public event EventHandler RaiseCustomEvent;

=====

自定义事件,  [EventHandler](https://msdn.microsoft.com/zh-cn/library/system.eventhandler.aspx) 的非泛型版本

public event CustomEventHandler RaiseCustomEvent;

=====

自定义事件,泛型版本

则:委托默认使用EventHandler ,那么不需要声明新的委托,也不用声明委托了;

直接使用

public event EventHandler<CustomEventArgs> RaiseCustomEvent;

=======================================================

触发事件方式

EventHandler<CustomEventArgs> handler = RaiseCustomEvent;//回调函数保存地方(委托可以添加方法数量)

e = new CustomEventArgs("Did something");

handler(this, e);//执行回调函数

//回调函数定义格式

void HandleCustomEvent(object sender, CustomEventArgs e)

## 在派生类中引发基类事件

基类中

1.定义回调函数保存的容器

2.创建虚方法,触发某事件的某徐方法

派生类中,

1.重写虚方法,调用基类的徐方法;

## 使用接口事件

接口定义

public interface IDrawingObject

{

event EventHandler ShapeChanged;

}

接口实现

public class Shape : IDrawingObject

{

//实现回调函数保存的位置

public event EventHandler ShapeChanged;

//定义发送者触发某事件的方法

void ChangeShape()

{

// Do something here before the event…

OnShapeChanged(new MyEventArgs(/\*arguments\*/));

// or do something here after the event.

}

//定义某事件触发后响应的方法;

protected virtual void OnShapeChanged(MyEventArgs e)

{

if(ShapeChanged != null)

{

ShapeChanged(this, e);

}

}

}

?字典存储事件实例

?自定义事件访问器

event EventHandler IDrawingObject.OnDraw

{

add

{

lock (PreDrawEvent)

{

PreDrawEvent += value;

}

}

remove

{

lock (PreDrawEvent)

{

PreDrawEvent -= value;

}

}

}

# 异常和异常处理

各种类型的异常最终都是由 **System.Exception** 派生而来。

在可能引发异常的语句周围使用 **try** 块。

一个包含代码的 [finally](https://msdn.microsoft.com/zh-cn/library/zwc8s4fz.aspx) 块，无论 **try** 块中是否引发异常（例如，释放在 **try** 块中分配的资源），这些代码都会运行。

catch 块可以指定要捕捉的异常的该类型。类型规范称为“异常筛选器”

可以使用 finally 块清理在 try 块中执行的操作。如果存在，finally 块将在最后执行，在 try 块和任何匹配 catch 的块之后执行。不管是否引发异常或者是否找到与异常类型匹配的 catch 块，finally 始终运行。

引发异常

不应使用异常来更改正常执行过程中的程序流程。异常只能用于报告和处理错误条件。

不要从自己的源代码中有意引发 [System.Exception](https://msdn.microsoft.com/zh-cn/library/system.exception.aspx)、[System.SystemException](https://msdn.microsoft.com/zh-cn/library/system.systemexception.aspx)、[System.NullReferenceException](https://msdn.microsoft.com/zh-cn/library/system.nullreferenceexception.aspx) 或[System.IndexOutOfRangeException](https://msdn.microsoft.com/zh-cn/library/system.indexoutofrangeexception.aspx)。

不要创建可在调试模式下引发但不会在发布模式下引发的异常。若要在开发阶段确定运行时错误，请改用调试断言。

只能引发异常，而不能作为返回值或参数返回异常。

定义异常类[Serializable()]

public class InvalidDepartmentException : System.Exception

{

public InvalidDepartmentException() : base() { }

public InvalidDepartmentException(string message) : base(message) { }

public InvalidDepartmentException(string message, System.Exception inner) : base(message, inner) { }

// A constructor is needed for serialization when an

// exception propagates from a remoting server to the client.

protected InvalidDepartmentException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

编译器生成的异常

?捕捉非 CLS 异常