《电脑编程技巧与维护》杂志十周年庆典暨真情回馈读者活动
《电脑编程技巧与维护》杂志社策划

编程技巧典型案例集锦系列

C# 编程技巧 典型案例解析

《电脑编程技巧与维护》杂志社 编著

- 用 C# 实现注册表编程
- 设计 C# 定制控件实现贝塞尔曲线
- 在C#中利用DataGrid控件实现主—从式查询
- 使用Crystal Report 做报表的数据库编程
- 局域网点对点通信程序之 C# 的实现
- 用 C# 开发手机短信收发程序
- 用 C# 编写多线程搜索引擎

超值 1CD，36 个实例，42000 条代码，
编程高手经验汇集，现学现用

中国电力出版社
www.infopower.com.cn

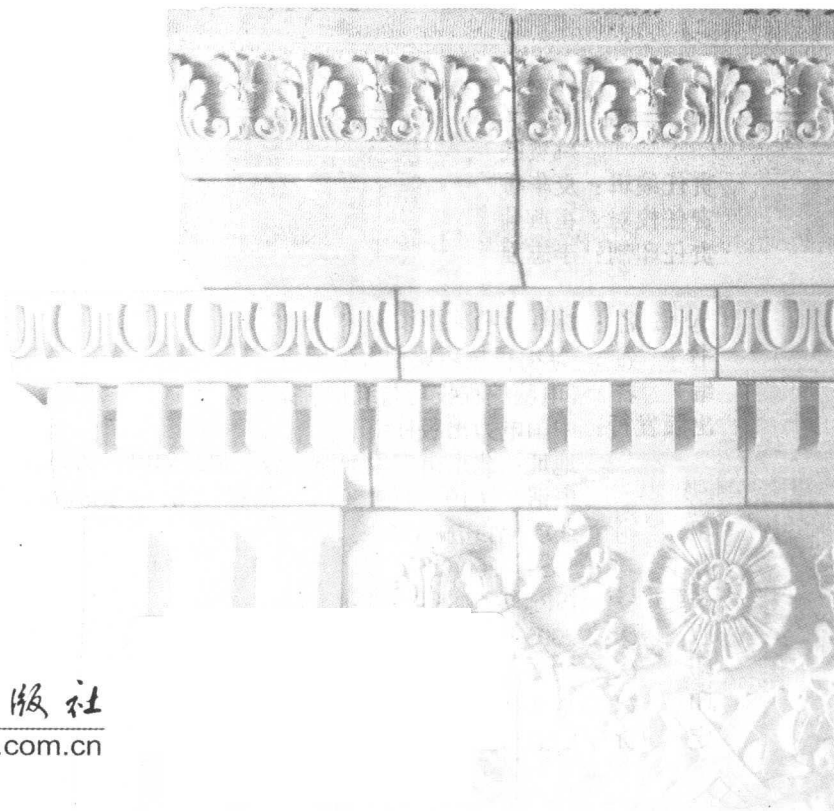
《电脑编程技巧与维护》杂志十周年庆典暨真情回馈读者活动
《电脑编程技巧与维护》杂志社策划

编程技巧典型案例集锦系列

C#编程技巧

典型案例解析

《电脑编程技巧与维护》杂志社 编著



中国电力出版社
www.infopower.com.cn

内 容 简 介

本书是《电脑编程技巧与维护》杂志社2005年的倾情奉献。主要通过48个实例，从C#编程的基础与应用、数据库应用、网络与通信应用3个方面对C#编程的方法和应用的技巧进行了深入探讨，其中包括Visual C#套接字编程、多线程编程、注册表编程、数据库连接、数据绑定、使用Crystal Report做报表、树视图、用C#实现UDP协议的实际应用、C#网络程序设计、点对点通信等。

本书实例丰富，每个实例都有很强的实用性和代表性，是Visual C#程序员、数据库编程人员、网络编程和维护人员及广大编程爱好者和计算机相关专业的学生的理想参考书。

图书在版编目（CIP）数据

C#编程技巧典型案例解析／《电脑编程技巧与维护》杂志社编著. —北京：中国电力出版社，2005
（编程技巧典型案例集锦系列）
ISBN 7-5083-3262-8

Ⅰ.C… Ⅱ.电… Ⅲ.C语言－程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2005）第017449号

策　　划：裴红义
　　姚贵胜
责任编辑：夏华香
责任校对：崔燕菊
责任印制：李志强

丛 书 名：编程技巧典型案例集锦系列
书　　名：C#编程技巧典型案例解析
编　　著：《电脑编程技巧与维护》杂志社
出版发行：中国电力出版社
　　地址：北京市三里河路6号　邮政编码：100044
　　电话：（010）88515918　传真：（010）88518169
印　　刷：北京同江印刷厂印刷
开本尺寸：185 × 260　　印　张：24
书　　号：ISBN 7-5083-3262-8
版　　次：2005年8月北京第1版
印　　次：2005年8月第1次印刷
印　　数：1~4000
定　　价：42.00元（含1CD）

丛 书 序

在《电脑编程技巧与维护》杂志创刊10周年之际，为了真诚回报多年来一直关爱和支持本刊的广大读者，《电脑编程技巧与维护》杂志社和中国电力出版社共同策划出版了《编程技巧典型案例集锦系列》丛书。《电脑编程技巧与维护》杂志是为从事电脑编程、系统应用开发人员创办的专业性和实用性都很强的技术刊物，它从1994年创刊，十多年来始终遵循着“实用第一，智慧密集”的办刊宗旨，紧跟计算机软硬件技术发展和应用趋势，不断求变创新，针对软件开发过程中许多关键技术问题，着重提供各类解决方案。对电脑编程人员来说，程序开发能力的提高，除了对语言和算法的学习外，还要集思广益，充分借鉴参考别人的长处，深入透彻地理解其中的精髓，然后融入到自己的设计方案中去，这样无论是对于自身还是整体都有莫大的提高，这也正是我们编写这套系列丛书的初衷。

本丛书包括《Visual C++编程技巧典型案例解析——基础与应用篇（上）》、《Visual C++编程技巧典型案例解析——基础与应用篇(下)》、《Visual C++编程技巧典型案例解析——图形图像处理与数据库篇》、《Visual C++编程技巧典型案例解析——网络与通信及计算机安全与维护篇》、《Visual Basic编程技巧典型案例解析》、《Delphi编程技巧典型案例解析》、《C#编程技巧典型案例解析》、《Java编程技巧典型案例解析》、《PowerBuilder管理信息系统编程技巧典型案例解析》9册共545个典型案例。每册书的编程案例，均依不同的编程应用分成若干章，条目清晰可查，使用极为方便。

本丛书选编了《电脑编程技巧与维护》杂志近一两年发表的和一部分尚未发表而又极为实用、精彩的典型编程实例，特点是：其各册内容均来自编程高手的智慧，凝结了500余位编程高手与名家的心血，关键技术专家点评；其案例是从实际项目提炼出的开发范例，超过800个技术要点的经典解决方案。案例讲解部分先给出设计目标，然后介绍实现目标的基本思想和方法，最后详细给出其核心程序的源代码，对程序的关键部分进行讲解并给出程序的运行效果；其编程技巧新颖实用，构思巧妙，汇集了众多顶级程序员和业界知名专家的成功经验，告诉读者最好的创意和最实用的方法。全套书既讲究内容的深入性、专业性和权威性，同时兼顾轻松、通俗易懂、时效性强的特点，带给读者的是一份清

新、纯粹的体验感受。

本丛书是《电脑编程技巧与维护》杂志资源的二次开发，浓缩了当前主流编程语言Visual C++、Visual Basic、Delphi、Java、C#、PowerBuilder等程序设计的精华，其目的是力求为读者建造一个真正的知识整合，是编程思想、编程技术、技巧交流的平台，让读者从中学习到编程高手的诀窍，丰富读者的编程技巧，拓宽读者的编程思路，迅速提升读者的程序开发能力。该丛书可作为高等院校学生进行课程项目开发、毕业项目设计的参考教材，软件从业人员及编程爱好者的珍藏宝典，也可作为高等培训学校的案例教程。

实例导航学编程，自学成才成高手，思想、智慧、理念、经验、技巧无处不在……

《电脑编程技巧与维护》杂志社

2005年1月

前　言

C#是Microsoft新一代.net平台的基石，是为创建高性能Windows、Web应用程序及组件而诞生的高级语言。C#无论是基于XML的Web服务(Web Services)，还是中间层业务对象、系统级应用等都不愧是一个优秀的开发工具。C#继承了Java和C++的诸多特性，学习掌握C#不仅可以增强读者的Web编程能力，还能提升读者的生产效率。

学习编程不难，难的是如何精通编程；做一名程序员不难，难的是如何做个优秀的程序员。为了让更多的电脑编程人员比较集中地学习和参考C#应用编程的实践经验、心得体会和技巧，在《编程技巧典型案例集锦系列》丛书中，《C#编程技巧典型案例解析》一书精选了《电脑编程技巧与维护》杂志近两年半共30期已发表的精彩编程实例48例。根据C#的不同应用对象，将精选的48个应用实例分为3章。第1章C#编程基础与应用编程实例，为初学者提供C#基础应用编程入门的实例；第2章C#数据库应用编程实例，为编程人员提供使用C#进行数据库编程方法和技巧的实例；第3章C#网络与通信应用编程实例，介绍使用C#实现网络与通信应用编程的技巧。全书每一章都本着实用第一的原则，紧紧围绕一个主题展开，由浅入深，通过一个个应用实例介绍使用C#进行应用程序开发的方法与技巧。

本书的主要特色如下：第一，每一章都是通过一个个的实例来介绍C#应用编程方法和技巧，避免了枯燥、空洞的理论，并且每一个实例都具有很强的实用性和代表性。在实例的讲解上一般都是先给出设计目标，然后介绍实现该目标的基本思想和方法，最后详细给出其核心程序的源代码，并对程序的关键部分进行讲解，给出程序的运行效果。第二，所选的每一个实例都是从事C#应用编程人员的经验总结，具有很强的实用性，其中很多编程技巧可供借鉴。第三，每一个实例的程序源代码都经过上机调试通过，给程序开发人员移植源代码带来了方便，加快编程应用的步伐。

本书是《电脑编程技巧与维护》杂志的二次开发，浓缩了C#程序设计的精华，其目的是提升读者C#程序开发能力，把应用C#进行编程的心得体会、经验与读者共享。该书定位于有C#应用基础的编程人员和应用开发人员，对初学C#编程的新手也有一定的参考价值。该书内容全面、概念清晰、层次分明、实例典型而实用，但不足甚至疏漏之处在所难免，恳请广大读者批评指正。

《电脑编程技巧与维护》杂志社

2005年1月

目　录

第 3 章 C#网络与通信应用编程实例

第 1 章
C#编程基础与应用编程实例

■实例 1
C#中消息的处理

Windows 应用程序是靠消息驱动的，所谓消息就是描述事件发生的信息。Windows 程序的执行顺序取决于事件发生的顺序，程序的执行顺序是由顺序产生的消息驱动的。程序员可以针对消息类型编写消息处理程序以处理接受的消息，也可以发出其他消息以驱动其他处理程序。在 Visual C + + 中有系统定义消息和自定义消息，系统定义消息可以通过 Class Wizard 为某窗口类添加消息处理函数，Class Wizard 将自动添加消息映射。在 C#中如何处理消息呢？

一、系统定义消息的处理

1. 利用委派和事件处理消息

事件和委派与 Windows 应用程序有什么关系？基于图形化用户界面的程序主要是事件驱动。用户在界面上操作，发出各种消息，程序必须响应这些事件，处理这些消息。一般情况下，代表用户界面中控件的对象由事件通知。事件提供了一种有效的方式，程序员可以通过这种方式了解可能对该对象进行的操作。

在 C#中，事件是用委派声明的。事件是类允许客户代码把委派传送给方法的一种方式，当该事件发生时，就调用客户代码所传送的委派。Event 关键字允许在代码中指定一个事件发生时调用的委派。

C#语言允许事件使用任何类型的委派，但 . net Framework 对用事件处理消息时所使用的委派类型有比较严格的限制，即在 . net Framework 中使用事件处理消息时，事件所使用的委派类型应带两个参数："对象源"参数表示事件的源；"e"参数则是封装了该事件的其他信息，"e"参数的类应派生于 EventArgs 类。

Windows Forms 利用委派使事件处理成为结构的一部分。除了方法和属性外，所有的控件都有事件集合，更具体地说，控件有事件处理程序集合。如果要捕获一个事件，只需编写一个方法，把它添加到要处理事件的集合中即可。

对于每个控件，Visual Studio. net 都对应一个最基本的事件和相应的方法，如按钮对应的是单击，文本框对应的是文本内容的改变。处理这种消息非常容易，只要把文本框拖到表单上，然后双击该文本框，Visual Studio. net 会在 InitializeComponent 方法中添加一行事件处理：

```
this. textBox1. TextChanged + =new System. EventHandler(this. textBox1_TextChanged);
```

以及一个新方法：

```
private void textBox1_TextChanged(object sender, System. EventArgs e){}
```

这些都是自动完成的。需要做的就是实现事件相应函数 textBox1_TextChanged()。我们如下实现处理函数，显示一个消息框：

```
private void textBox1_TextChanged(object sender, System.EventArgs e)
{
    MessageBox.Show("文本内容发生变化");
}
```

如果基本事件不满足要求，我们可以利用上述方法添加事件及方法。我们仍以文本框为例，当单击该文本框时，显示一个消息框，再拖一个文本框到表单上，然后在 InitializeComponent 方法中添加如下一行事件处理：

```
this.textBox2.Click += new System.EventHandler(this.textBox2_Click);
```

在代码窗口中添加相应的方法如下：

```
private void textBox2_Click(object sender, System.EventArgs e)
{
        MessageBox.Show("单击文本框");
}
```

2. 对消息的重载

在 C#中处理消息与 MFC 的消息处理是类似的，但前者更为简单。MFC 中需要使用 DECLARE_MESSAGE_MAP 来定义消息映射，在 C#中就不需要了。如对 WM_PAINT 消息，只需重载父类中的 OnPaint 虚拟方法即可，方法如下：

在菜单 View | Other Windows | Object Browser 打开对象浏览器窗口，在工程名下找到 Form 并选中，这时在右边的窗口列出所有 Form 类的成员函数。

选中 OnPaint(System.WinForms.PaintEventArgs)，此时在下面会显示完整的 OnPaint 函数：

```
protected virtual void OnPaint ( System.Windows.Forms.PaintEventArgs e )
```

将这一行字符串拷贝下来。打开 Form1.cs 代码窗口，把刚才拷贝下来的函数定义复制到 Form1 类里面。去掉其中的 virtual，加上 override 关键字，此时便可以在里面添加消息处理代码了，详见如下清单：

```
protected override void OnPaint ( System.Windows.Forms.PaintEventArgs e )
{
    Font font = new Font("黑体", 18);
    SolidBrush bluepen = new SolidBrush(Color.Blue);
    e.Graphics.DrawString("C#重载 OnPaint 方法", font, bluepen, 15, 25);
}
```

二、自定义消息的处理

以上介绍的是系统定义消息的处理，如同 MFC 中一样，用户在进行 API 编程时常常用到自定义消息，在 C#中也提供了相应的实现方法。C#自定义消息的处理与 MFC 中的处理基本是一样的，为了处理的方便，在这里将把使用到的自定义消息封装成一个类，在代码窗口中添加 TestMessage 类，并为 TestMessage 类添加两个成员，具体代码如下：

```
public class TestMessage
{
    public const int USER = 0x0600;
    public const int TEST1 = USER + 1;
}
```

在 Form1 中添加一个按钮，并为该按钮添加事件处理代码，向主窗口发送 TEST1 自定义消息，具体代码如下：

```
protected void button1_Click(object sender, System.EventArgs e)
{
```

```
        SendMessage(TestMessage.TEST1, 120, 220);
    }
```

消息已经发出了，在 Form1 中如何对该消息作出响应呢？可以重载 DefWndProc 方法。详细清单如下：

```
    protected override void DefWndProc ( ref System.Windows.Forms.Message m )
    {
        switch(m.Msg)
        {
            case TestMessage.TEST1:
                string message = string.Format("收到消息！参数为{0},{1}", m.LParam,
                m.WParam);
                MessageBox.Show(message); //显示一个消息框
                break;
            default:
                base.DefWndProc(ref m); //基类函数处理非自定义消息
                break;
        }
    }
```

在 C#中处理消息比 MFC 中简单，给编程带来了方便。

（蔺新华）

■实例 2
深入浅出 C#消息

C#(发音 c sharp) 是 Microsoft 公司新推出的 Visual Studio. net 中的一门新的语言。微软官方说,它是一种简单但功能强大的编程语言,用于编写企业级的应用程序。本人是从 Visual Studio. net beta1 开始接触 C#的。经过几个月的学习和应用,我觉得 C#确实是一门比较“酷”的语言。本文以下的内容主要是介绍我在使用 C#开发基于 Windows Forms 应用程序过程中的一点心得。我希望我的文章如我的 program 一样严谨,但正如所有的 program 一样,难免要有些 bug。

一、消息概述

我们知道在 Windows 下应用程序的执行是用消息驱动的。消息就是我们整个程序工作的引擎,所以理解掌握我们使用的语言是如何封装消息的原理,就等同于我们拥有了程序运作的全部动力。程序的质量、灵活度在一定程度上取决于我们运用消息原理的质量和灵活度。在我具体阐述 C#的消息机理前,让我先解说几个基本的概念,如果你是 API SDK 的高手,可以略过这一节。

1. 什么是消息

我个人认为消息就是通知和命令。在早期使用 API 编程的时候,我们可以看到消息在 WINUSER. H 中是使用一个 struct 定义的。

```
typedef struct tagMSG {
    HWND        hwnd;
    UINT        message;
    WPARAM      wParam;
    LPARAM      lParam;
    DWORD       time;
    POINT       pt;
} MSG;
```

其中,hwnd 域为消息发向窗口的标识;message 是消息类型的标识;wParam 为一个 16 位的消息参数(早期平台为 16 位,现在为 32 位);lParam 为一个 32 位的消息参数;wParam,lParam 在不同的 message 下有不同的定义。

在 .net 框架类库的 System. Windows. Forms 命名空间中,微软用面向对象的方式重新定义了 Message。新的 Message 结构的 Public 属性基本和早期的一样,不过它是面向对象的。

公共属性如表 2 - 1 所示。

2. 消息驱动的过程

所有的外部事件,如键盘输入、鼠标移动、按动鼠标都由 Windows 系统转换成相应的消息发到应用程序队列。每个应用程序都有一段相应的程序代码来检索、分发这些消息到相应的窗口,然后由窗口函数来处理。

表 2－1 公共属性

HWnd	获取或设置消息的窗口句柄
lParam	指定消息的 lParam 字段
Msg	或设置消息的 ID 号
Result	指定为响应消息处理而向 Windows 返回的值
wRaram	获取或设置消息的 wRaram 字段

早期的基于 API 的编程，我们可以很清楚地看到这个处理过程。那时的 Windows 应用程序的基本框架一般如图 2－1 所示。

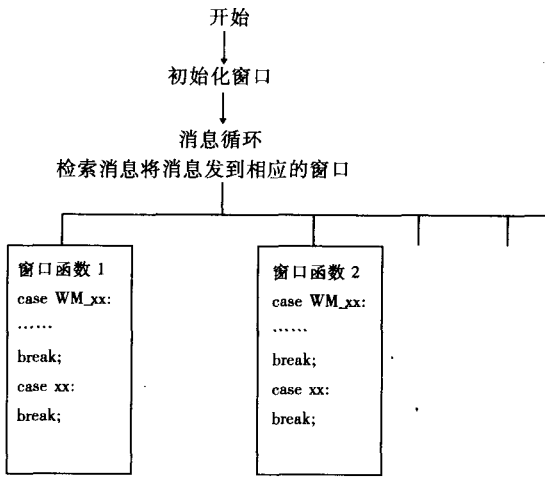


图 2－1

二、C#中的消息封装

为了简化用 C#开发 Form 应用程序，创建纯面向对象的接口，C#对消息重新进行了面向对象的封装，在 C#中消息被封装成了事件。让我们先看一个类：System. Windows. Forms. Application 类。

Application 类具有用于启动和停止应用程序和线程以及处理 Windows 消息的方法。调用 Run 以启动当前线程上的应用程序消息循环，并可以选择使某窗体可见。调用 Exit 或 ExitThread 来停止消息循环。从上面的 Application 类的介绍中，我们可以知道 C#中是用 Application 类来处理消息的接收和发送的。消息的循环是由它负责的。本质上来讲，每个窗体一般都对应一个窗口过程。那么，C#的一个 Form 实例（相当于一个窗口）收到消息后是如何处理消息的呢？其实，这个问题的分析也就展示了 C#的消息封装原理。让我们从几个“简单”而又奇怪的例子去看本质吧！

1. Example 1

实现鼠标左键单击消息的响应 （WM_LBUTTONDOWM）。

假设我们已生成一个窗体 Form1：

```
this. MouseDown + =newSystem. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown);
this. MouseDown + =newSystem. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown2);
private void Form1_MouseDown(object sender, System. Windows. Forms. MouseEventArgs e)
{
    if(e. Button = =System. Windows. Forms. MouseButtons. Left)
```

```
        System. Windows. Forms. MessageBox. Show("鼠标左键按下 1");
    }
    private void Form1_MouseDown2 ( object sender, System. Windows. Forms. MouseEventArgs e)
    {
      if(e. Button = =System. Windows. Forms. MouseButtons. Left)
        System. Windows. Forms. MessageBox. Show("鼠标左键按下 2");
    }
```

通过如上的程序片断，我们在 Form1 窗体上单击鼠标左键的时候将顺序地出现两个标题为“鼠标左键按下”的消息框，上面 this. MouseDown 是 C#中的一个事件。它的定义如下：

```
public event MouseEventHandler MouseDown;
```

那么事件在 C#中是什么？为理解这个问题，我们先看看 MouseEventHandler 定义：

```
public delegate void MouseEventHandler(
  object sender,
  MouseEventArgs e
);
```

看看该语句像不像函数原型的声明，如果你能看到这一点，说明你靠近了目标一点点。

其实那是定义了一个委托类型。委托启用其他语言（如 C + + 、Pascal 和 Modula）已经用函数指针解决的方案。与 C + + 函数指针不同，委托是完全面向对象的；与指向成员函数的 C + + 指针不同，委托同时封装对象实例和方法。本质上委托把一个实例和该实例上的方法函数封装成一个可调用的实体，它是面向对象的、安全的。

分析到这里我们可以把

```
this. MouseDown + =newSystem. Windows. Forms. MouseEventHandle
(this. Form1_MouseDown);
```

这条语句看成向 this. MouseDown 添加了一个函数指针。让我们总结一下事件和委托的关系。

事件是对象发送的消息，以发信号通知操作的发生。操作可能是由用户交互（例如鼠标单击）引起的，也可能是由某些其他的程序逻辑触发的。引发（触发）事件的对象叫做事件发送方。捕获事件并对其作出响应的对象叫做事件接收方。在事件通信中，事件发送方类不知道哪个对象或方法将接收到（处理）它引发的事件。所需要的是在源和接收方之间存在一个媒介（或类似指针的机制）。.net 框架定义了一个特殊的类型（Delegate），该类型提供函数指针的功能。这样，委托就等效于一个类型安全函数指针或一个回调。在上面的例子中我们向 this. MouseDown 事件添加了两个委托。

```
this. MouseDown + =newSystem. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown);
this. MouseDown + =newSystem. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown2);
```

结果我们的两个函数 this. Form1_MouseDown、this. Form1_MouseDown2 在我们单击鼠标左键的时候都被调用，而且调用的顺序和我们添加委托的顺序一样。通过上面程序运行现象，我可以假设 C#是这样封装消息的。

WM_LBUTTONDOWM 消息首先被 Application 对象从应用程序队列中取出，然后分发到相应的窗体。窗体使用 MouseDowm 事件中的函数指针调用已经添加的响应函数。所以 C#中的事件字段实质上是一个函数指针列表，用来维护一些消息到来时的响应函数的地址。

问题是不是如此？答案不仅仅如此简单。

2. Example 2

我们在上述例子的基础上再作一下修改，修改后的程序基本框架，如图 2 - 2 所示。

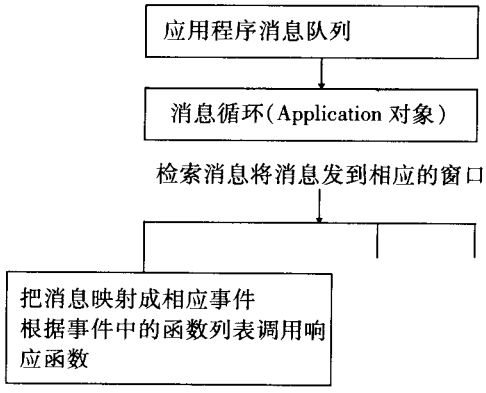


图 2 - 2

```
new public event MouseEventHandler MouseDown;
this. MouseDown + =newSystem. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown);
this. MouseDown + =newSystem. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown2);
private void Form1_MouseDown(object sender, System. Windows. Forms. MouseEventArgs e)
{
    if(e. Button = =System. Windows. Forms. MouseButtons. Left)
      System. Windows. Forms. MessageBox. Show("鼠标左键按下 1");
}
private void Form1_MouseDown2(object sender, System. Windows. Forms. MouseEventArgs e)
{
    if(e. Button = =System. Windows. Forms. MouseButtons. Left)
    System. Windows. Forms. MessageBox. Show("鼠标左键按下 2");
}
```

我们如此修改后让完整的程序运行，结果在窗体上再单击鼠标左键的时候，将看不到消息框的显示，这说明了我们的两个事件响应函数没有被调用。

我们再修改上面的程序片断。

3. Example 3

```
this. MouseDown + =new System. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown);
this. MouseDown + =new System. Windows. Forms. MouseEventHandler
(this. Form1_MouseDown2);
private void Form1_MouseDown( object sender, System. Windows. Forms. MouseEventArgs e)
{
   if(e. Button = =System. Windows. Forms. MouseButtons. Left)
     System. Windows. Forms. MessageBox. Show("鼠标左键按下 1");
}
private void Form1_MouseDown2 ( object sender, System. Windows. Forms. MouseEventArgs e)
{
   if(e. Button = =System. Windows. Forms. MouseButtons. Left)
```