

# 目 录

实验一. 利用 ArcGIS Engine 中的控件创建简单的 GIS 小程序 .....	3
实验二. ArcObjects 组件的事件处理机制 .....	19
实验三. Map 组件中的图形元素操作 .....	34
实验四. 利用 Geodatabase API 读取 shapefiles 数据 .....	43
实验五. 矢量要素的查询和编辑 .....	62
实验六. 矢量要素的渲染 .....	80
实验七. 网络数据集环境下的路径分析 .....	98

## 实验一. 利用 ArcGIS Engine 中的控件创建简单的 GIS 小程序

### 一、实验目的

掌握 AE 中常用的可视化交互控件 MapControl、ToolBarControl 和 TOCControl 的使用方法,能够利用这些控件快速搭建一个小型的 GIS 应用程序。以可视化交互控件的使用作为学习的第一步,几乎没有写代码的过程,仅通过在开发环境中进行命令操作来生成一个基本的可视化应用程序外壳,并具备了基础的 GIS 操作功能,有助于快速建立 AE 二次开发的初步直观印象,为后续深入学习提供合适的切入点和效果展示途径。

### 二、实验仪器

常规配置微机, Windows 操作系统, Visual Studio2005 及以上版本, ArcGIS Engine9.2 及以上版本。本指导书将以 Visual Studio 2010 和 ArcGIS Engine10.0 为基本开发环境,在其他版本的 Visual Studio 和 ArcGIS Engine 中进行开发的过程仅有细微差别,读者可自行查找相关技术资料进行详细了解和处理。

### 三、实验要求

了解启动 AE 二次开发的基本步骤,了解 MapControl、ToolBarControl、TOCControl 和 LicenseControl 的基本功能,掌握 C#环境中在 Windows Form 窗体上布局和配置 AE 控件的基本方法,认识 Visual Studio 环境中命令操作与代码联动的基本概念和意义。

### 四、实验内容

1. 在 Visual Studio 中创建 AE 二次开发的项目。
2. 在 Windows Form 窗体中合理布置 MapControl、ToolBarControl、TOCControl 及 LicenseControl,形成具有地图主窗口、地图鹰眼窗口、工具栏和图层信息列表的基本窗体布局。
3. 了解上述控件的参数设置方法,按常用模式配置窗体中的各控件。
4. 了解 Visual Studio 中命令操作与代码联动的基本概念,认识实验操作过程引发的主要代码联动结果,并思考其意义。

### 五、实验步骤

1. 在 Visual Studio 中创建 AE 二次开发项目的实验步骤
  - (1) 正确安装 Visual Studio 和 ArcGIS Engine (略)。
  - (2) 打开 Visual Studio 集成开发环境,依次选择“文件”->“新建”->“项目”

菜单项，打开新建项目对话框，选择 Visual C#语言并创建“Windows 窗体应用程序”（如图 1.1 所示）：

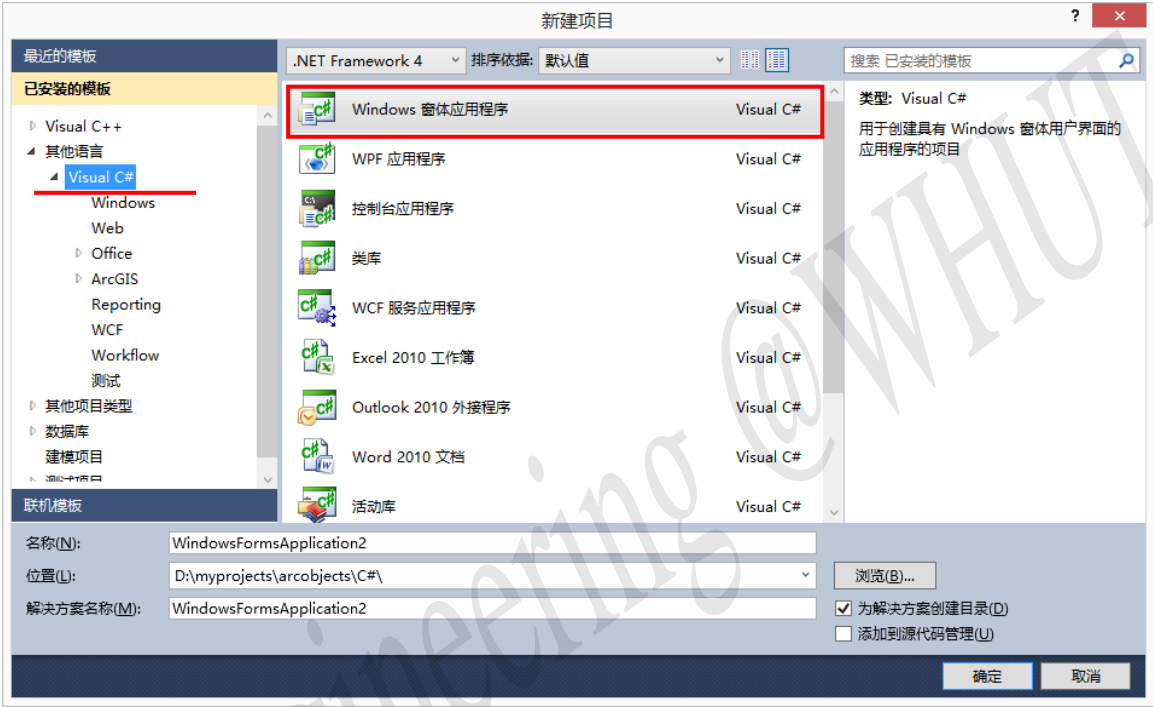


图 1.1. 项目类型设置

(3) 在上述窗口中设置好应用程序的名称和项目的存储路径（如图 1.2 所示，本指导书中项目名称以“GISTest”为例，存储路径以“D:\myprojects\arcobjects\C#\”为例）：

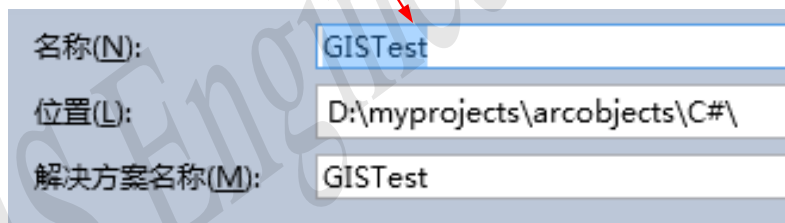
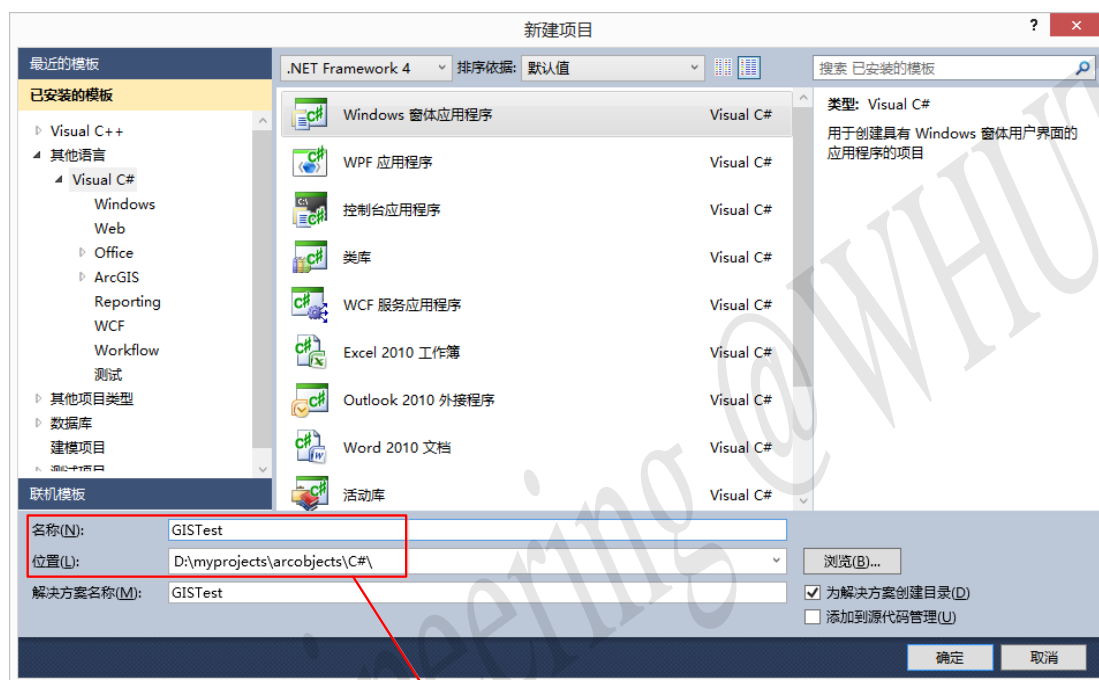


图 1.2. 项目名称和存储位置设置

(4) 上述步骤完成后将创建一个用于 AE 二次开发的项目，下面简单了解项目中的重要部分。点击上图的“确定”按钮后，开发环境将显示如图 1.3 所示界面，其中的重要部分如图 1.4 所示。图 1.4 标注了“解决方案管理器”列表中包含的主要信息项，其中的“引用”、“窗体功能代码源文件”和“窗体设计代码源文件”在后续实验中将会频繁使用，而右侧的“窗体设计器”窗口是用来进行控件布置的主要操作界面，窗体设计器中的效果基本上表达了程序运行后的窗体效果：

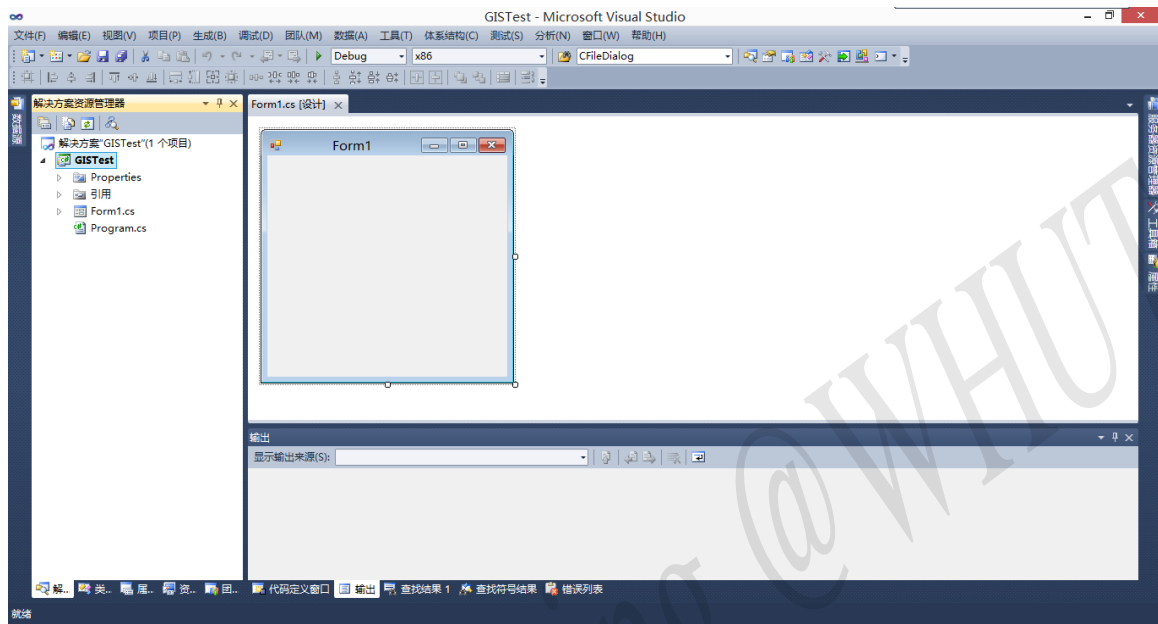


图 1.3.创建项目后的初始状态

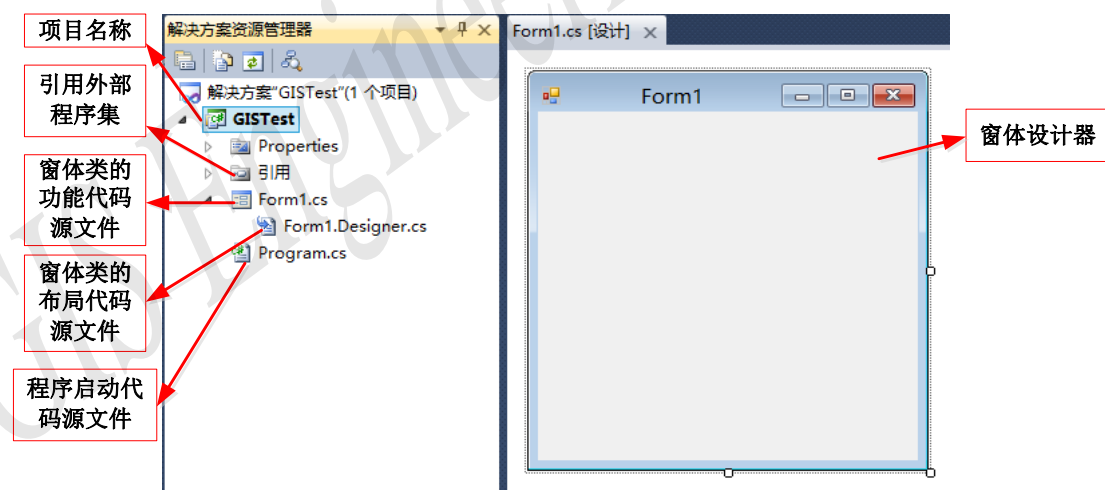


图 1.4. 项目中的重要组成部分

## 2. 布置 MapControl、ToolbarControl、TOCControl 及 LicenseControl 的实验步骤

(1) 布置控件的主要工作就是将控件合理的放置在容器窗体的窗体设计器中。在 AE 中利用 MapControl、ToolbarControl 和 TOCControl 三者就可以构建一个简单但完整的 GIS 数据可视化显示和操控界面体系，因此本实验主要介绍这三者的操作，而 LicenseControl 是 AE 提供给用户用于快速设置许可参数的辅助控件，需要一并安放在窗体设计器中。

(2) 拖拽窗体设计器边框上的操作点（如图 1.5 红色框部分所示），将窗体调整至合适大小。

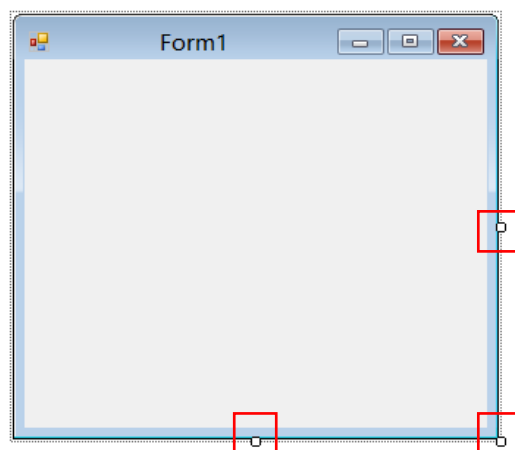


图 1.5. 调整窗体尺寸的操作

(3) 打开 Visual Studio 的“工具箱”操作窗口，在工具列表中找到“ArcGIS Windows Forms”部分（如图 1.6 所示），该部分包括的控件就是 AE 为我们提供的主要控件，其中可以找到 MapControl、ToolBarControl、TOCControl 及 LicenseControl。

(4) 将 MapControl、ToolBarControl 和 TOCControl 用鼠标拖拽至容器窗体的窗体设计器中，并调整到合适的位置和尺寸。本实验要求的基本布局如图 1.7 所示，形成地图主窗口、地图鹰眼窗口、工具栏和图层信息列表的标准形式。注意控件区域中的“Name”信息指明了该控件在程序中对应的对象变量名称，而控件区域中的“Note”信息提示程序中缺乏调用控件所需的许可设置，此时只需将 LicenseControl 拖拽至窗体设计器中的任意位置就可消除这些“Note”信息，该控件在程序运行时不会被显示。

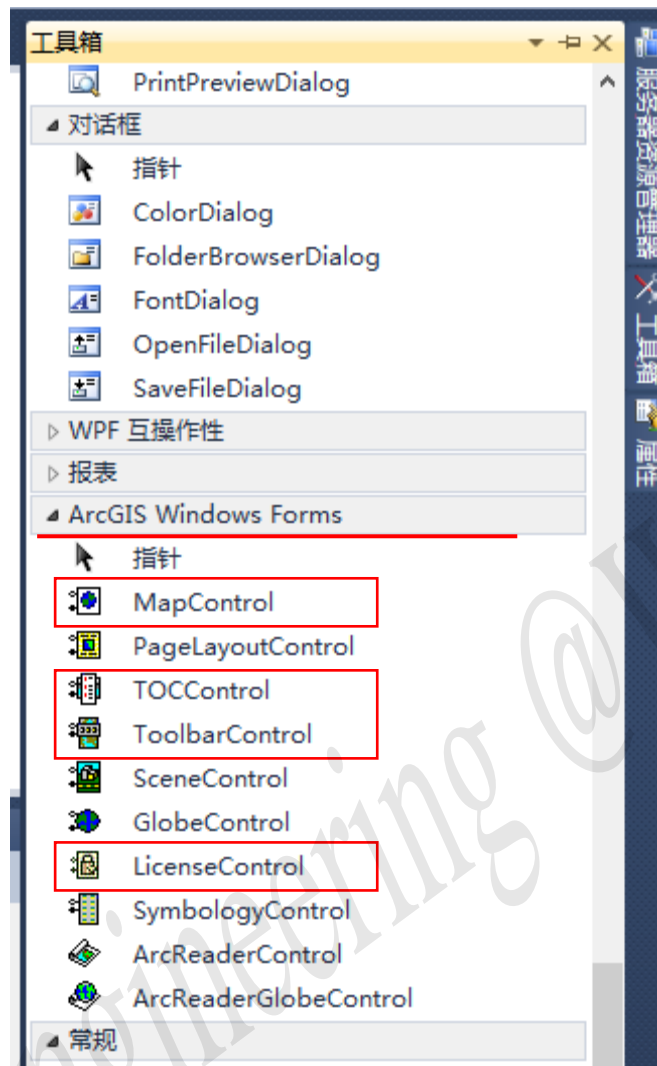


图 1.6. AE 控件列表

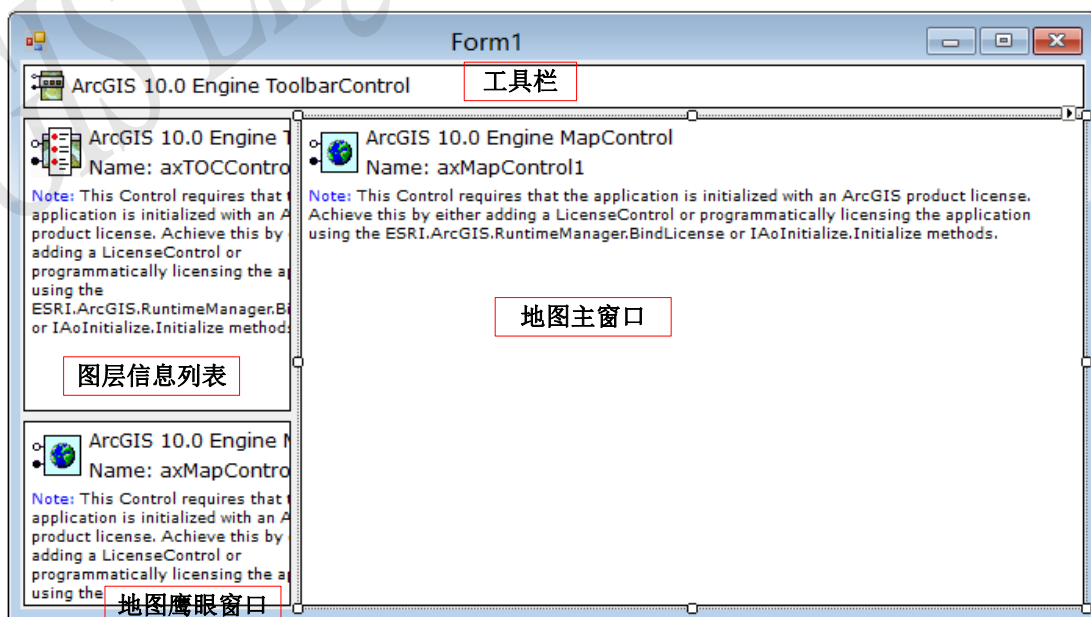


图 1.7. 窗体基本布局

(5) 如果是在 ArcGIS Engine10.0 环境中开发,除了添加 LicenseControl 外,还需在 Program.cs 文件的 main 函数中加入如图 1.8 红色框中所示代码,才能完整的设置桌面版应用程序许可信息。

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    ESRI.ArcGIS.RuntimeManager.Bind(ESRI.ArcGIS.ProductCode.EngineOrDesktop);

    Application.Run(new Form1());
}
```

图 1.8. 设置桌面版应用程序许可的代码

(6) 在“窗体设计器”中放置控件的同时,“解决方案管理器”中的“引用”项下也会自动添加与这些控件相关的外部程序集引用,如图 1.9 所示。如果是程序中要用到其他非控件类型的组件,可能需要添加新的外部程序集引用,这种情况下 Visual Studio 不会自动为我们添加,需要进行手动添加,添加方法见后续实验。

另外,如果在 visual studio2010 环境下,建议读者对每个引用的 AO 组件库执行一次“取消嵌入互操作”的操作,否则在后续的编程过程中可能会在某些组件的互操作执行方面出现一些问题,至于为什么要取消嵌入互操作目前作者还无法给出准确原因,操作的执行过程如图 1.10 所示(以 ESRI.ArcGIS.Carto 程序集为例)

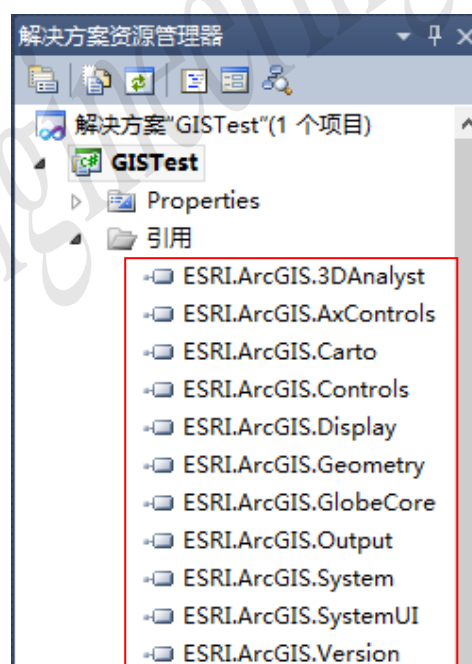


图 1.9. 自动添加的外部程序集引用



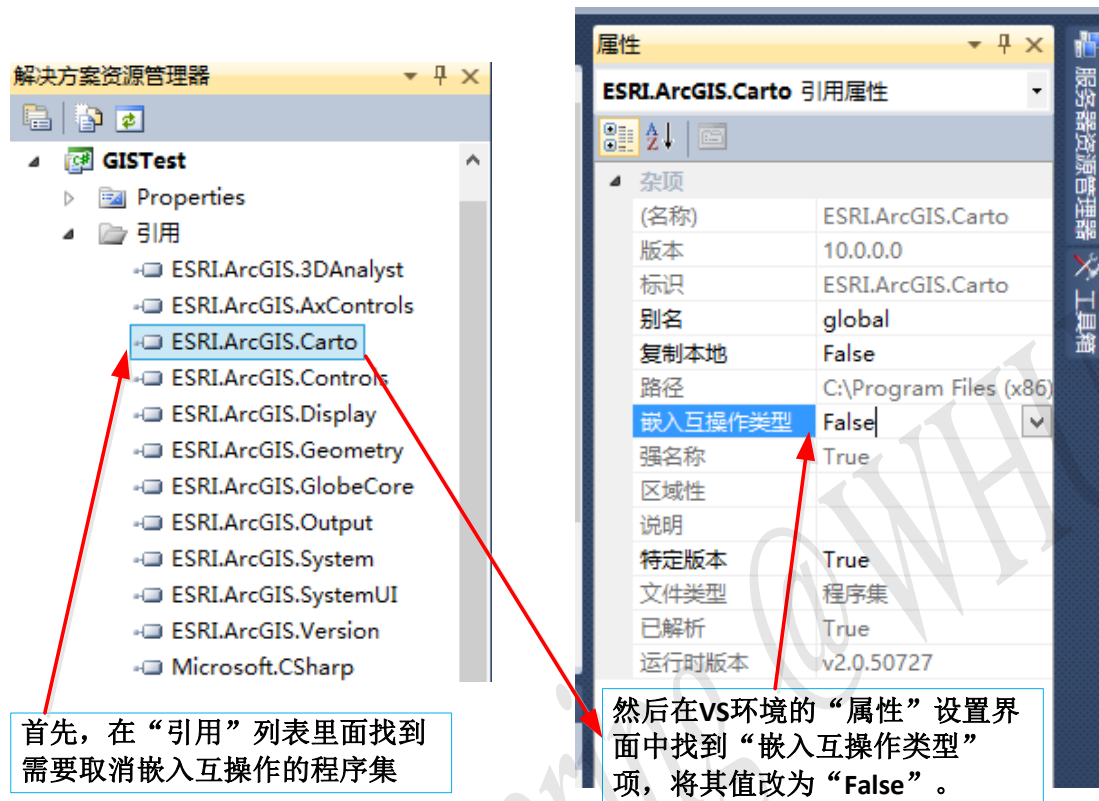


图 1.10. 取消 AO 程序集嵌入互操作的执行过程

3. 控件的参数设置的实验步骤。控件作为一种特殊的窗口元素，有一些 Windows 窗口属性需要设置，这部分内容属于 C#窗体程序开发的基础知识，此处不做介绍，后文主要介绍它们作为 AO 控件的特殊参数设置问题。

(1) 控件布置好之后，点击 Visual Studio 环境的工具栏上的“运行”按钮（如图 1.11 红色框所示）或者在键盘上按“F5”键，将进入调试运行状态，程序运行显示如图 1.12 所示。可以看到程序初始运行并未如期待显示出工具栏和地图操作界面，无法进行用户操作，这是因为没有进行正确的控件参数设置。控件参数设置方法如后续步骤所示。



图 1.11. “运行”按钮的位置

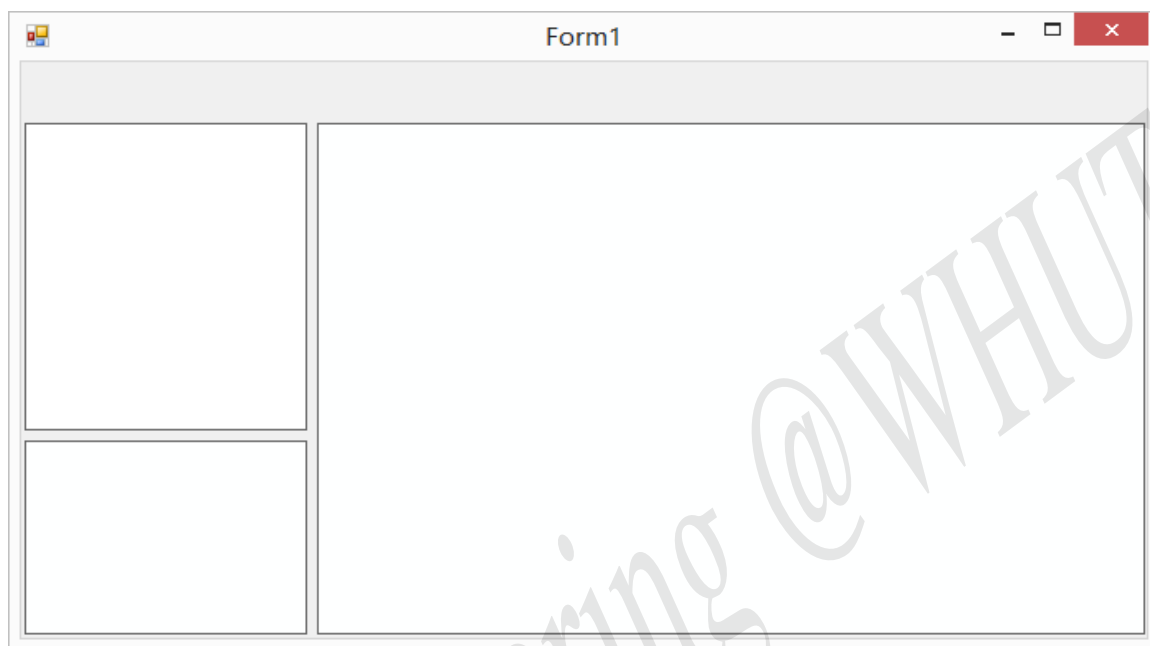


图 1.12. 程序的初始运行效果

(2) 工具栏的设置。ToolBarControl 控件实现了工具栏，但程序所需的工具按钮及其他相关参数需要由开发者在开发时进行手动设置。在“窗体设计器”中选中 ToolBarControl 控件并单击鼠标右键，在弹出的菜单中选择“属性”项目，打开属性设置对话框，如图 1.13 所示：

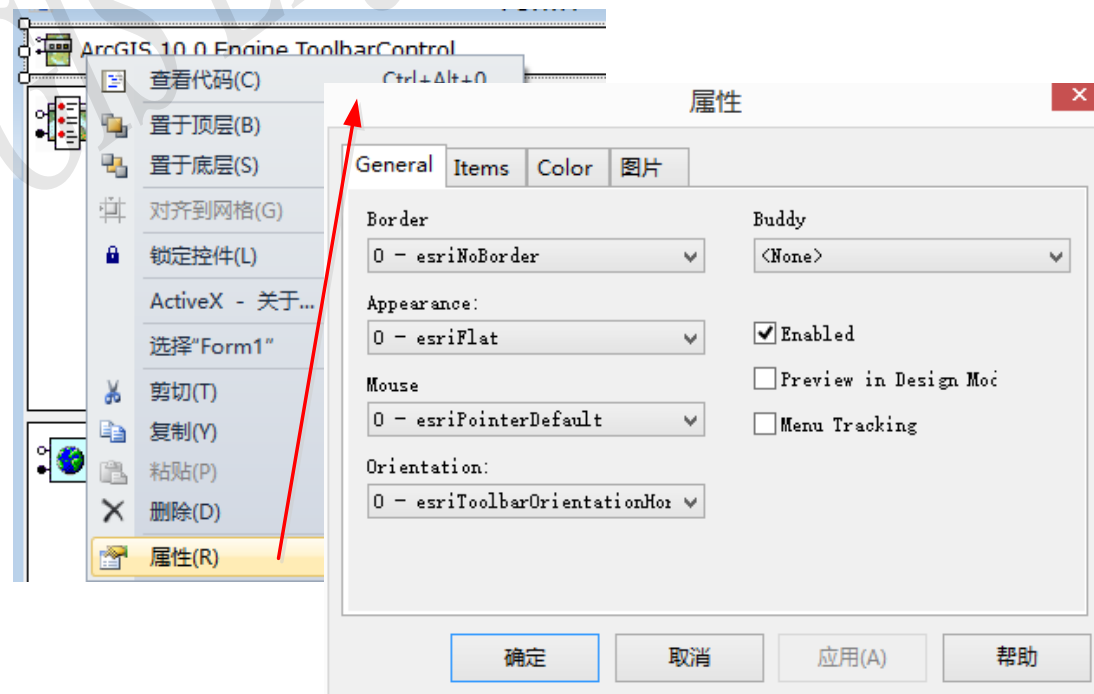


图 1.13. ToolBarControl 控件的属性设置对话框

其中“General”属性页中定义了一些基本显示效果属性，读者可以自由设置并运行测试一下，但需重点注意其中的“Buddy”属性，这个属性的目的是设置该 ToolBarControl 所对应的“伙伴控件”，对于 ToolBarControl 和 TOCControl 这类控

件，它们并不能直接表达空间信息，而是提供对空间信息表达窗口的辅助操作功能（例如地图缩放、查询等），因此它们必须和表达空间信息的窗口合并在一起才会有意义，需要接收它们的操作命令并反馈操作结果的空间信息表达窗口控件就成为它们的“伙伴控件”。在我们的实验中，地图操作应针对地图主窗口进行，因此该 `ToolBarControl` 的“伙伴控件”应设置为地图主窗口对应的 `MapControl` 控件的对象名（控件窗口的“Name”信息显示了对象名，在指导书给出的例子中是“`axMapControl1`”），可以通过该属性下的下拉列表进行选择。工具栏中按钮的设置主要是在属性对话框的“Item”属性页中进行的，如图 1.14 所示。

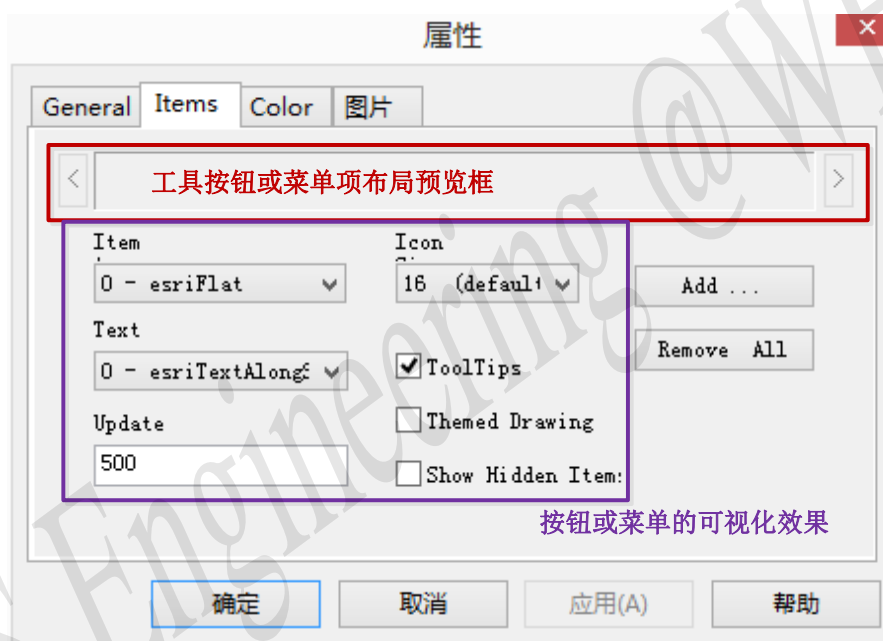


图 1.14. 属性对话框的 Items 属性页

`ToolBarControl` 将工具按钮和菜单进行了统一管理，都可以作为工具项排布在控件窗口中，设置的效果可以在预览框中模拟显示，开发者可以通过“Add”按钮打开工具集，选择需要出现在 `ToolBarControl` 中的项目，工具集对话框如图 1.15 所示，不同类型的工具集对应不同的属性页，每个属性页中都主要包含“Category”和“Commands”两个列表框，“Category”列表框显示了当前类型工具集中的各个工具组，选中某个工具组后，“Commands”列表框中就显示了该组中的各工具项。可以将通过双击的方法将整个工具组或者某个单独的工具添加到控件中，也可以用鼠标左键按住工具组或者工具拖拽到“Items”属性页的预览框中进行添加。如果要撤销某个工具，可以直接用鼠标将它拖出预览框，点击“Remove All”按钮将撤销所有的工具。操作过程如图 1.16 所示。

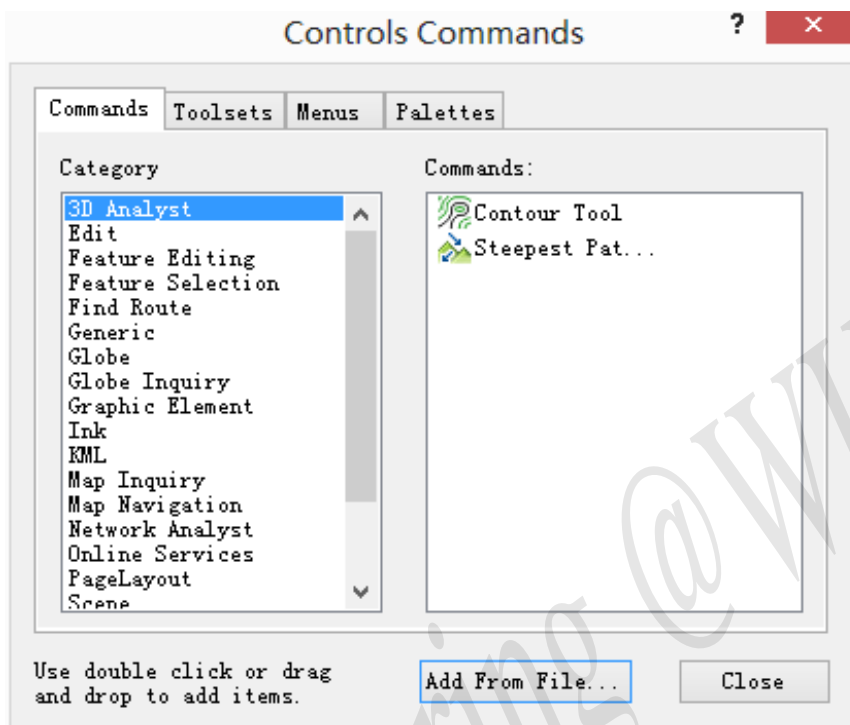


图 1.15. 工具集对话框

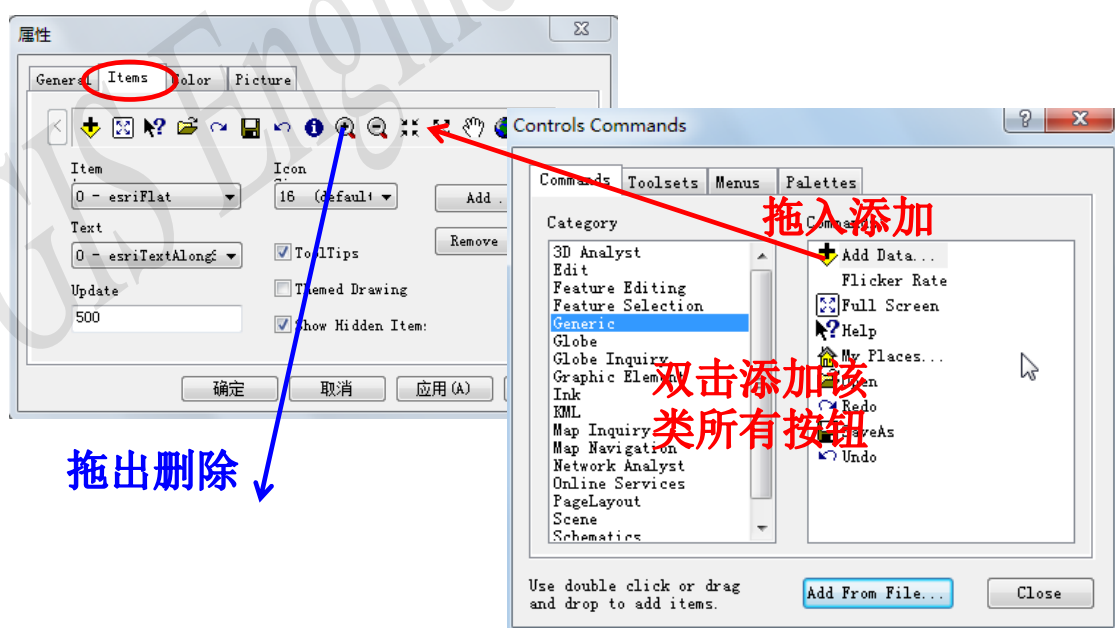


图 1.16. 添加和删除工具栏按钮的方法

(3) 图层信息列表的设置。TOCControl 控件实现了简单的图层信息列表功能，其属性设置对话框如图 1.17 所示，其中 General 属性页中的参数设置项基本上都与可视化效果和互动操作相关，读者可以自行尝试。但该属性页右上角的“Buddy”

需要设置为该控件的“伙伴控件”，在本例中它的“伙伴控件”与 ToolbarControl 一致，都是 axMapControl1。

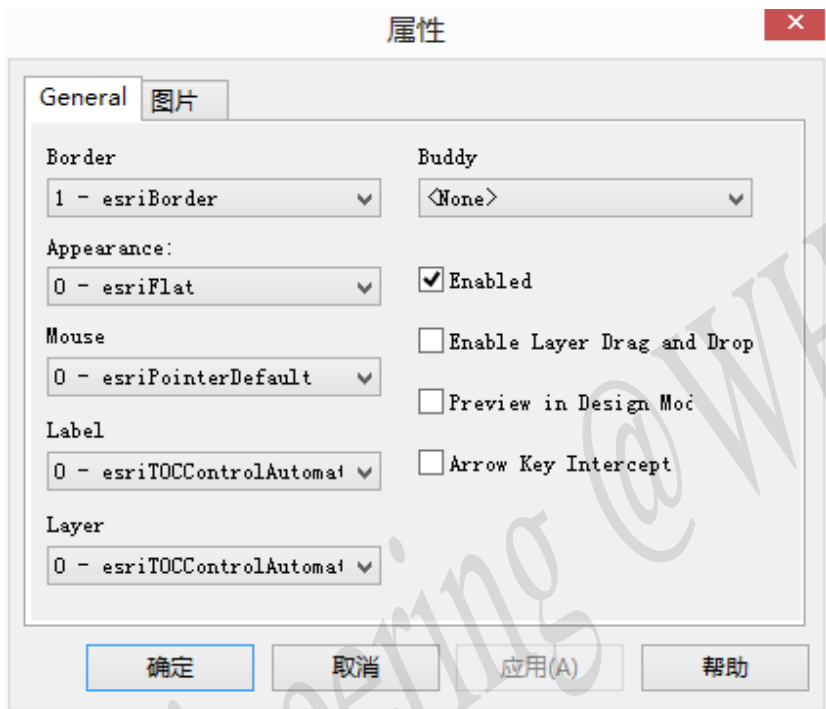


图 1.17. TOCControl 控件的属性对话框

(4) 地图主窗口的参数设置。地图主窗口由一个 MapControl 控件实现。该控件的属性对话框如图 1.18 所示，其中最主要的属性都在 General 属性页中进行设置，各项属性的含义也标示在下图中。大部分属性都与可视化效果和交互操作相关，读者可以充分地尝试各种设置参数以了解其意义和作用，如果勾选了“Preview in Design Mode”项，则大部分设置的效果可以直接在视图设计器的窗体中显示出来，否则就需要编译运行后显示。但是，有两类属性是与地图数据相关的，需要额外解释一下。“Map Document”框用来设置该 MapControl 中默认显示的地图数据，其中填入的是要相应 mxd 文件的路径名，如果该框保持空白或者文件路径错误，该 MapControl 初始化后将显示一个空白窗口。General 属性页的底部两个选项表示 MapControl 如何连接默认的地图数据，“Contain the map inside mapcontrol”表示一旦设置了 Mxd 文件的路径后，就将地图数据直接 copy 到应用程序数据段内，即使地图数据发生了变化、被删除了或者应用程序被移植到其他电脑上运行，都不会影响以及加载的地图数据；而“Link to the map document by filename”表示当应用程序运行时，该 MapControl 组件对象实例化后动态读取指定路径下的地图数据。



图 1.18. MapControl 的属性对话框及参数说明

(5) 地图鹰眼窗口的参数设置。鹰眼窗口也是一个 MapControl 控件，其可视化效果也可如图 1.18 所示设置，但是鹰眼承担了一些独特的功能，这些功能的实现更多的是依靠编程过程而不是属性设置过程，因此本次实验中暂不要求对地图鹰眼窗口进行参数设置。

(6) 正确设置好参数后运行程序，会显示一个类似图 1.19 所示的效果：

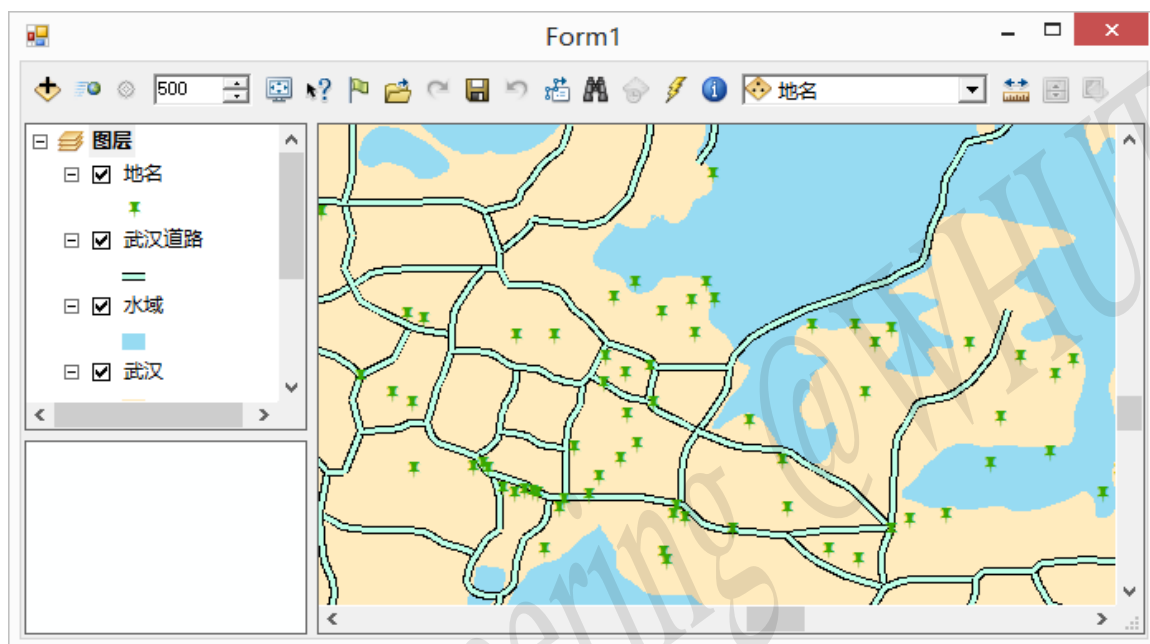


图 1.19. 程序运行效果图

#### 4. 查看 Visual Studio 中命令操作与代码联动的实验步骤

(1) 虽然本次实验的开发过程几乎不涉及代码编写，而主要是依靠命令操作完成的，但从程序实现的本质来说，任何效果最终都是靠程序代码来完成的，本次实验中的命令操作实际上是利用了 Visual Studio 环境为我们提供的便捷编程模式，而这些操作背后所涉及的代码编写由 Visual Studio 环境帮我们自动完成了，我们称之为命令操作与代码联动。但作为后续学习的基础，了解这些自动创建代码的相关信息是很有必要的，它们是在给定的开发环境下，最权威最合理的代码示例，通过对这些代码的分析和理解，往往可以达到举一反三、事半功倍的学习效果。

(2) 查看添加控件的代码。添加控件涉及到对容器窗口对象成员的修改，这部分代码主要体现在容器窗口类的布局代码源文件中，在本实验的例子中是“Form1.Designer.cs”文件。当自动创建窗体模式的应用程序项目后，Form1.Designer.cs 文件是图 1.20 显示的样子：



```

namespace WindowsApplication2
{
    partial class Form1
    {
        /// <summary>
        /// 必需的设计器变量。
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// 清理所有正在使用的资源。
        /// </summary>
        /// <param name="disposing">如果应释放托管资源，为 true；否则为 false。 </param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows 窗体设计器生成的代码
    }
}

```

图 1.20. Form1.Designer.cs 文件的初始状态

当在 Form1 中添加了地图主窗口 MapControl 控件后，这段代码变成了如图 1.21 显示样子。请注意红色框中的代码就是 Visual Studio 为我们自动添加的，这行代码显示了添加的 MapControl 控件实际上是成为了 Form1 类的一个成员变量，名字为 axMapControl1，类型为 ESRI.ArcGIS.Controls.AxMapControl，从类型定义中可以明确看出 AO 组件的“命名空间”定义方法，AxMapControl 是组件类名，ESRI.ArcGIS.Controls 是该类所在的命名空间，这个命名空间实际上就是组件类所属的组件库的名称。至于类名为什么是 AxMapControl 而不是 MapControl 是因为 AE 的 .Net 开发包针对 .Net 环境下对窗体控制的要求，对控件进行了进一步封装，并在类名前添加“Ax”前缀以示与原控件的区别。如果要简化类型定义时的命名空间定义，可以在当前源程序中使用 using 关键字将整个命名空间引入源程序的定义域，例如在 Form1.Designer.cs 文件的开头添加如图 1.22 中红色下划线所标注的代码，在后面的 axMapControl1 变量定义时就可以省略 ESRI.ArcGIS.Controls 部分：



```

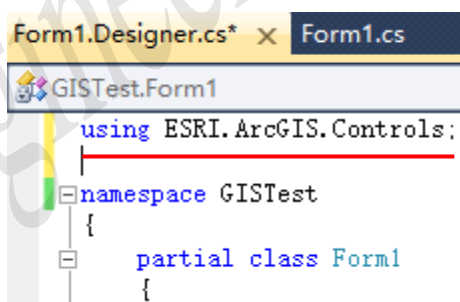
namespace WindowsApplication2
{
    partial class Form1
    {
        /// <summary>
        /// 必需的设计器变量。
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// 清理所有正在使用的资源。
        /// </summary>
        /// <param name="disposing">如果应释放托管资源，为 true；否则为 false。 </param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        Windows 窗体设计器生成的代码
        private ESRI.ArcGIS.Controls.AxMapControl axMapControl1;
    }
}

```

图 1.21. 添加地图主窗口后的代码变化



```

Form1.Designer.cs* x Form1.cs
GISTest.Form1
using ESRI.ArcGIS.Controls;
namespace GISTest
{
    partial class Form1
    {

```

图 1.22. 使用 using 关键字添加命名空间

## 六、思考题

- (1) 请列出本次实验操作完成后所发生的所有代码变化，并试着分析其意义。

## 实验二. ArcObjects 组件的事件处理机制

### 一、实验目的

通过学习 MapControl 控件的事件处理过程，了解事件处理的基本概念和调用结构；通过学习 Map 组件的事件处理过程，了解利用程序代码处理组件事件的基本方法。

### 二、实验仪器

常规配置微机，Windows 操作系统，Visual Studio2005 及以上版本，ArcGIS Engine9.2 及以上版本。本指导书将以 Visual Studio 2010 和 ArcGIS Engine10.0 为基本开发环境，在其他版本的 Visual Studio 和 ArcGIS Engine 中进行开发的过程仅有细微差别，读者可自行查找相关技术资料进行详细了解和处理。

### 三、实验要求

了解“事件驱动”模式的基本概念，了解事件处理的控制流程及其涉及的主要部分，了解事件处理函数的定义规则和作用，学会利用时序图表达事件处理过程，学会通过编写代码的方式自定义事件委托和事件处理函数。

### 四、实验内容

本次实验在上次实验所创建项目的基础上进行，整个实验围绕鹰眼窗口的地图同步功能的实现过程展开，分别介绍利用 MapControl 的事件处理和 Map 组件的事件处理两种解决方案，通过实验掌握 AO 组件事件处理的基本方法。主要包括：

1. 在 Visual Studio 中为窗体添加事件处理函数；
2. 利用 MapControl 控件的事件处理实现鹰眼窗口与地图主窗口的数据同步；
3. 控件事件处理的代码分析；
4. 利用 Map 组件的事件处理实现鹰眼窗口与地图主窗口的数据同步。

### 五、实验步骤

1. 在 Visual Studio 中为窗体添加事件处理函数。“事件驱动”是一种常用的交互式程序控制模式，“事件”通常代表程序对象的某些特殊状态，这些状态是其他交互对象或者外界交互设备所关注的，它们可以根据这些状态做出特定的响应，那么程序对象就需要一种机制能够在这些特殊状态发生时通知其他交互方，而其他交互方也需要能自由选择对应的响应行为，这种机制就是“事件处理”机制。

在良好定义的“事件处理”机制下，交互方只需针对它所感兴趣的程序对象定义它所感兴趣的特定事件，并按照规定实现其事件响应代码，就能顺利执行事件驱动过程。事件处理机制的基本流程是：首先明确控制方和被控制方，两者都是特定的程序对象，被控制方具有一系列自定义的特殊状态，当某个特殊状态出现时能够以“事件”的形式通知外界，称为“触发”一个事件。如果控制方需要了解被控制方的某些特殊状态并做出特定的响应行为，它需要首先向被控制方进行注册，以告知被控制方它所感兴趣的事件，这个过程称为“事件监听”，然后针对每一个事件定义一个专门的函数来表达对应的响应行为，这个函数称为“事件处理函数”，这样当被控制方发出相应的事件通知后，控制方就可以监听到这个事件，并自动调用相应的事件处理函数来执行相应的行为，这一过程称为“事件响应”。

C#环境下的所有窗体和控件都支持事件处理机制，其中最常用的处理模式是由容器窗体监听并处理它所包含的各种子窗体或子控件的事件，对于这种模式在 Visual Studio 中可以通过简单的操作进行事件监听并定义事件处理函数，当然事件处理函数的实现代码需要开发者自行编写。下面以一个常用的事件处理过程为例，介绍窗体事件处理的操作过程。Windows 窗体对象有一个表达其被操作系统所加载的特殊状态，当这个状态发生时，窗体会触发一个“Load”事件，通常在这个事件的处理函数中执行某些窗体成员的初始化操作，可以将 Load 事件处理过程中的控制方和被控制方都看作是这个窗体对象，在本实验案例中这个窗体对象是 Form1，在 Visual Studio 中可以如下操作：

在窗体设计器中选中 Form1 窗体，并打开其“属性”窗口，然后点击“事件”按钮（如图 2.1 红色方框所示），“属性”窗口中将列出该窗体对象所能触发的所有事件，找到“Load”事件（如图 2.1 红色椭圆框所示），并双击该事件，Visual Studio 将会在 Form1 的功能代码源文件（即 Form1.cs）中自动定义该事件的处理函数，如图 2.2 所示，在这个自动定义的函数中添加执行代码就可以实现事件响应。

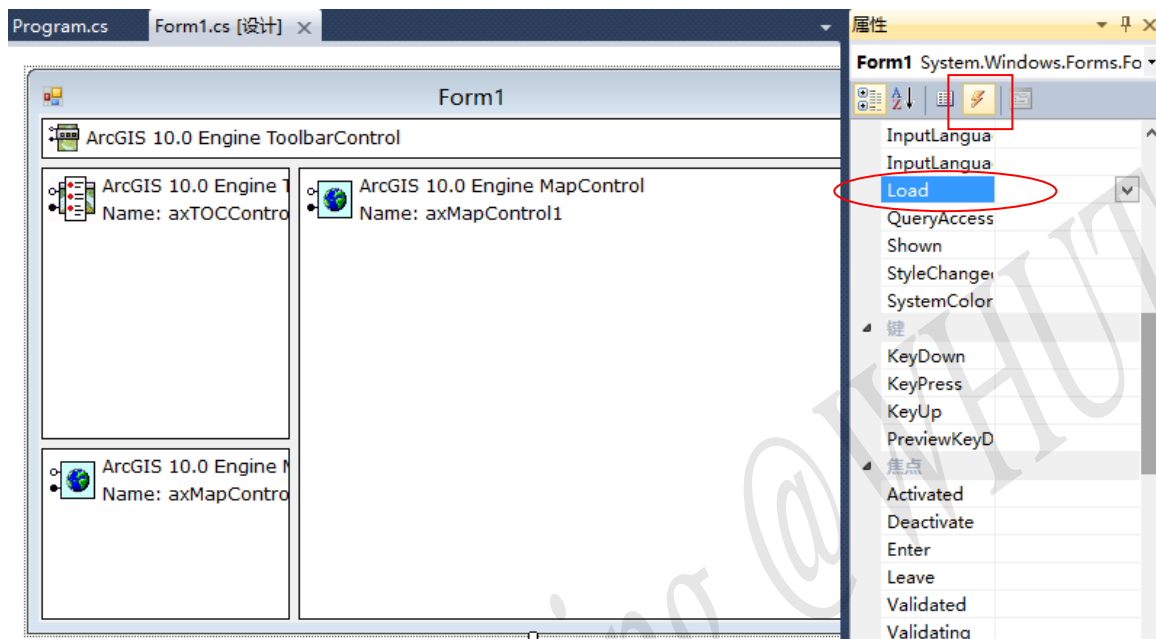


图 2.1. 添加 Load 事件处理

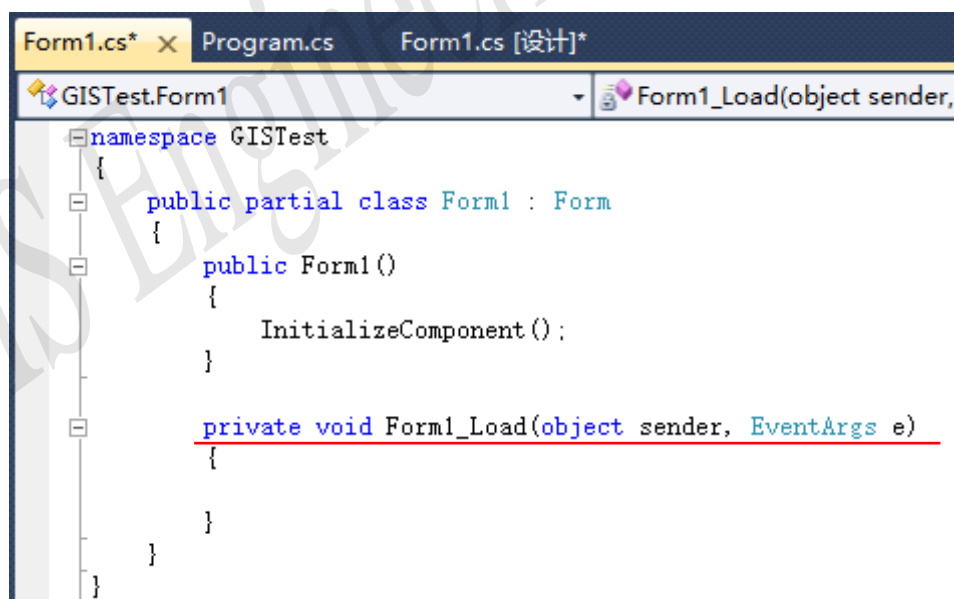


图 2.2. 自动定义的 Load 事件处理函数

2. 利用 MapControl 控件的事件处理实现鹰眼窗口与地图主窗口的数据同步。鹰眼窗口需要以“全图”的方式显示与地图主窗口一致的数据内容，但地图主窗口的数据可以运行过程中动态加载，此时鹰眼窗口要进行实时的数据同步，如果将主窗口看作被控制方，将鹰眼窗口看作控制方，则同步过程可以看作一种典型的事件处理过程，由控制方监听被控制方的“加载地图”事件，然后在事件响应中加载与被控制方相同的地图数据，该过程可以利用“时序图”的形式表达如下：

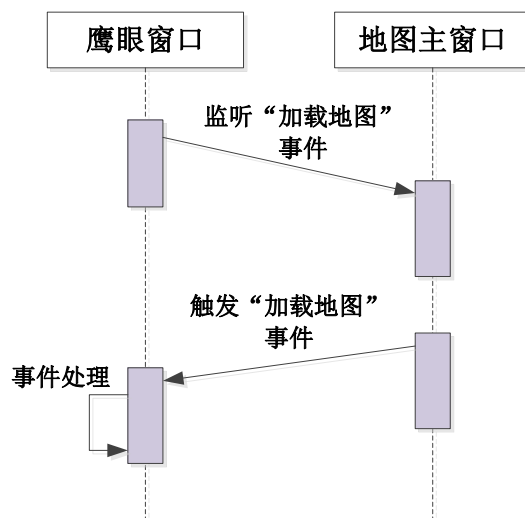


图 2.3. 鹰眼地图同步的事件处理过程

但是图 2.3 的设计存在一个问题，鹰眼窗口也是一个 MapControl 组件对象，它的源代码是不能被调用者修改的，因此就不能在这个对象中定义新的事件处理函数并编写事件响应代码，为了解决这个问题，可以在这个设计流程中添加一个我们可以修改其源代码的对象作为中转对象来监听并处理主窗口的事件，选择容器窗体 Form1 作为这个中转对象是很合适的，在这一思路下的事件处理过程如图 2.4 所示：

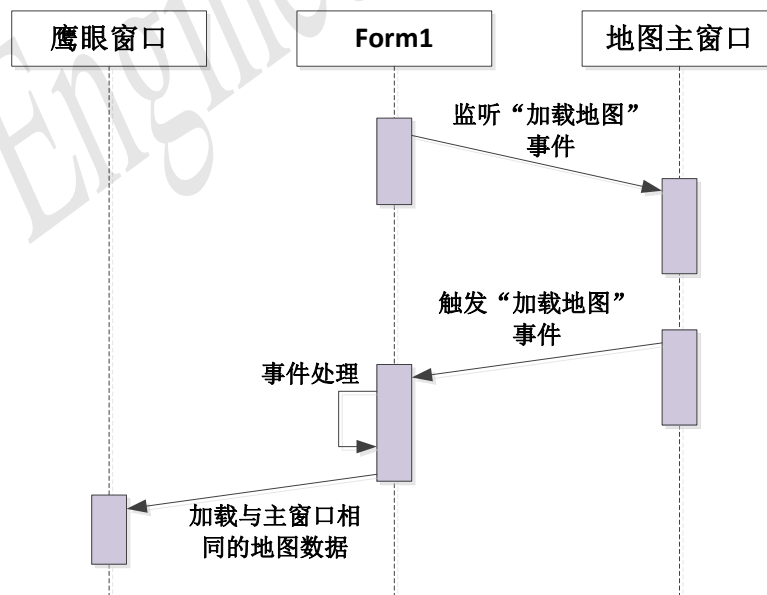


图 2.4. 利用 Form1 作为中转对象的事件处理过程

由于 Form1 是地图主窗口的容器窗体，因此在 Form1 中监听和处理地图主窗口的事件可以采用前述步骤完成。具体步骤如下所示：

- (1) 在 Form1 中监听并处理地图主窗口的“加载地图”事件。当 MapControl

中加载了新的 mxd 文件后，会触发一个 OnMapReplaced 事件，因此只需打开地图主窗口对应的 MapControl 对象的属性窗口，找到 OnMapReplaced 事件并双击就可以自动完成事件处理函数的定义，该函数定义在 Form1.cs 源文件中，如图 2.5、2.6 所示：

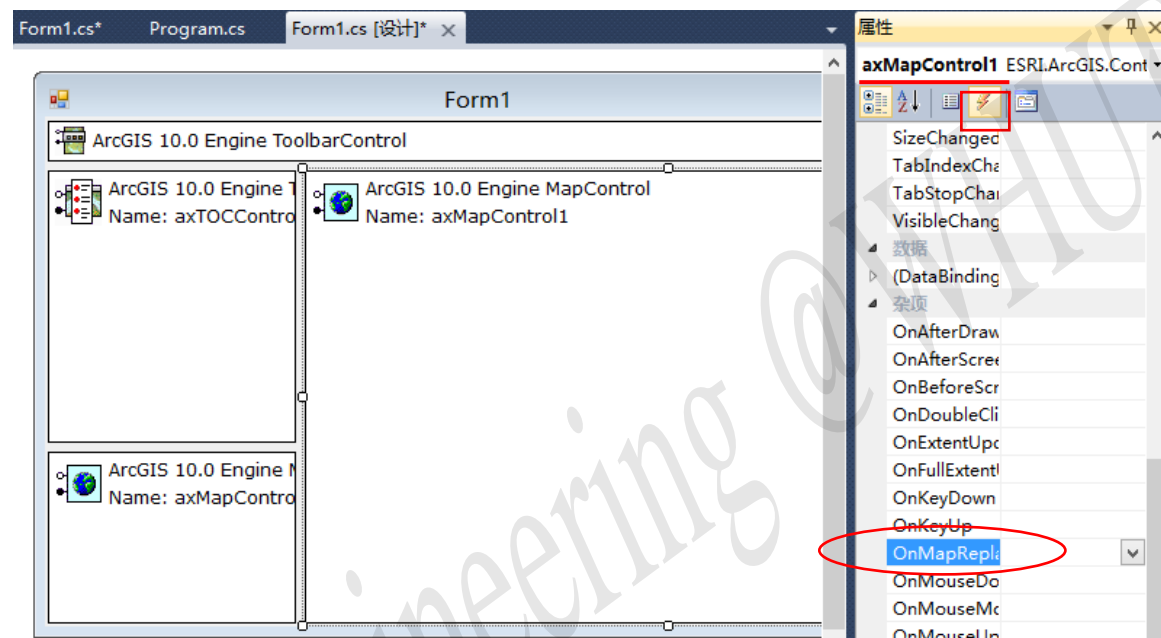


图 2.5. 找到地图主窗口的 OnMapReplaced 事件

```
private void axMapControl1_OnMapReplaced(object sender, ESRI.ArcGIS.Controls.IMapControlEvents2_OnMapReplacedEvent e)
{
}
}
```

图 2.6. Visual Studio 在 Form1 窗体中自动创建的事件处理函数

(2) 在事件处理函数中填写代码，使鹰眼窗口加载与地图主窗口相同的地图数据，可以参考代码示例 CODE2.1（假设地图主窗口对应的 MapControl 对象是 axMapControl1，鹰眼窗口对应的 MapControl 对象是 axMapControl2），其中需要注意的是用红色标出的部分，这一个 get\_Layer 方法在 MapControl 的帮助文档中是查不到的，因为它并不是该控件定义的原始方法，该控件定义了一个属性 Layer，这是一个集合属性，在地图中可以加载多个图层，因此存在一个 Layer 的集合，那这个 Layer 属性到底获取的是集合中的哪一个图层呢？必须通过附加的参数来说明，因此这类集合属性与普遍属性不同，在调用它们时需要附加参数，而在 C#语法中不允许调用时在成员变量后面添加参数，因此要将调用的语句改为函数的形式，对于这类需要添加参数的属性调用，在 C#内统一改为“get\_属性名”的函数形式，这就是“get\_Layer”方法的由来：

CODE 2.1. 在鹰眼窗口中加载与地图主窗口相同的地图数据

```
//清空鹰眼中的原有数据：
axMapControl2.ClearLayers();
//将主窗口中的图层按原有顺序添加到鹰眼控件中：
int n = axMapControl1.LayerCount;
for (int i = n - 1; i >= 0; i++)
{
    axMapControl2.AddLayer(axMapControl1.get_Layer(i), 0);
}
```

3. 控件事件处理的代码分析。读者应养成在实验中注意分析 Visual Studio 命令操作和代码联动的习惯，有利于快速学习编程方法。我们通过添加 OnMapReplaced 事件前后的代码变化分析事件处理过程的代码编写方法。

(1) 观察代码变化。添加 OnMapReplaced 事件的操作除了在 Form1.cs 中添加了事件处理函数外，还影响了 Form1.Designer.cs 源文件中的 InitializeComponent 方法，如果仔细阅读该方法的源代码可以通过注释了解到，该方法的主要任务是设置各种子控件的初始参数，我们定位到 axMapControl1 对象（地图主窗口）的初始化代码部分，在添加 OnMapReplaced 事件前该部分代码如图 2.7 所示：

```
private void InitializeComponent()
{
    System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(Form1));
    this.axMapControl1 = new ESRI.ArcGIS.Controls.AxMapControl();
    this.axTOCControl1 = new ESRI.ArcGIS.Controls.AxTOCControl();
    this.axToolbarControl1 = new ESRI.ArcGIS.Controls.AxToolbarControl();
    this.axMapControl2 = new ESRI.ArcGIS.Controls.AxMapControl();
    this.axLicenseControl1 = new ESRI.ArcGIS.Controls.AxLicenseControl();
    ((System.ComponentModel.ISupportInitialize)(this.axMapControl1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axTOCControl1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axToolbarControl1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axMapControl2)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axLicenseControl1)).BeginInit();
    this.SuspendLayout();
    //
    // axMapControl1
    //
    this.axMapControl1.Location = new System.Drawing.Point(179, 37);
    this.axMapControl1.Name = "axMapControl1";
    this.axMapControl1.OcxState = ((System.Windows.Forms.AxHost.State)(resources.GetObject("axMapControl1.OcxState")));
    this.axMapControl1.Size = new System.Drawing.Size(501, 311);
    this.axMapControl1.TabIndex = 0;
    //
```

图 2.7. 添加 OnMapReplaced 事件前的 Form1 窗体 InitializeComponent 方法部分代码

而添加 OnMapReplaced 事件后该部分代码如图 2.8 所示，与图 2.7 相比多了一行红色下划线标出的代码，这行代码的作用显然就是设置事件监听：

```

private void InitializeComponent()
{
    System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(Form1));
    this.axMapControl1 = new ESRI.ArcGIS.Controls.AxMapControl();
    this.axTOCControl1 = new ESRI.ArcGIS.Controls.AxTOCControl();
    this.axToolbarControl1 = new ESRI.ArcGIS.Controls.AxToolbarControl();
    this.axMapControl2 = new ESRI.ArcGIS.Controls.AxMapControl();
    this.axLicenseControl1 = new ESRI.ArcGIS.Controls.AxLicenseControl();
    ((System.ComponentModel.ISupportInitialize)(this.axMapControl1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axTOCControl1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axToolbarControl1)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axMapControl2)).BeginInit();
    ((System.ComponentModel.ISupportInitialize)(this.axLicenseControl1)).BeginInit();
    this.SuspendLayout();
    //
    // axMapControl1
    //
    this.axMapControl1.Location = new System.Drawing.Point(179, 37);
    this.axMapControl1.Name = "axMapControl1";
    this.axMapControl1.OcxState = ((System.Windows.Forms.AxHost.State)(resources.GetObject("axMapControl1.OcxState")));
    this.axMapControl1.Size = new System.Drawing.Size(501, 311);
    this.axMapControl1.TabIndex = 0;
    this.axMapControl1.OnMapReplaced += new ESRI.ArcGIS.Controls.IMapControlEvents2.Ax_OnMapReplacedEventHandler(this.axMapCo
    //

```

图 2.8. 添加 OnMapReplaced 事件后的 Form1 窗体 InitializeComponent 方法部分代码

(2) 分析事件监听代码。上述事件监听代码是有 Visual Studio 环境自动创建，语法规则，且具有鲜明的模式化特征，对其进行分析有助于更深刻理解和掌握创建事件监听的编程方法。该行代码的完整情况如下所示：

CODE 2.2. VS 自动创建的设置 OnMapReplaced 事件监听代码

```

this.axMapControl1.OnMapReplaced += new
ESRI.ArcGIS.Controls.IMapControlEvents2.Ax_OnMapReplacedEventHandler(
    this.axMapControl1_OnMapReplaced);

```

首先，该行代码从形式上看是一个利用“+=”操作符进行赋值的过程，这是 C# 语言定义事件监听的标准方法，C# 语言利用委托机制实现事件监听和处理函数回调，特定的事件通知将发送给符合特定规范的“事件委托类”对象，且一个事件可以与多个“事件委托类”对象相关联，从而实现事件通知的广播。而定义这种关联关系就是通过“+=”操作符，请注意上述代码的左边就是需要监听的事件，而右边是一个利用 new 操作符实例化的对象，这个对象就是符合 OnMapReplaced 事件监听规范的“事件委托类”对象，这个类的名称是用绿色文字表示的部分（类名前面的部分是其命名空间，我们可以在源文件开头使用 using 关键字来简化命名空间部分），而类的构造函数的参数正好是 Visual Studio 自动添加的那个事件处理函数名，这样就将事件监听与事件处理函数关联起来了。上述代码具有非常强烈的模式化特征，在定义其他的事件监听时可以仿造这个模式编写代码。

(3) 分析事件委托类。上述代码中的绿色文字部分表示了事件委托类，它的相关信息同样值得仔细分析，首先，类名本身也具有明显的模式化特征，整个类名



由“\_”分为三段，最后一段是“OnMapReplacedEventHandler”，字面意义就是“OnMapReplaced 事件处理器”，第一段是“IMapControlEvents2”，实际上是定义这个事件的接口名称，如果我们查阅 MapControl 的帮助文档就会在事件接口列表中看到这个接口名称，如图 2.9 所示。另外，如果查阅 IMapControlEvents2 的帮助文档可以看到该接口的命名空间是 ESRI.ArcGIS.Controls，与事件委托类的命名空间是一致的。而中间一段“Ax”的意义等同于 AxMapControl 类名中的“Ax”。因此当需要通过编程实现其他的事件监听时，可以模仿这个模式写出对应的事件委托类。

Event Interfaces	
Interfaces	
IMapControlEvents2 (default)	
ITOCBuddyEvents	

图 2.9. MapControl 帮助文档中的事件接口列表

除了分析类名之外，还可以进一步查看一下该事件委托类的详细定义，在代码文档中将鼠标移至委托类名上，点击右键，在弹出菜单中选择“转到定义”项（如图 2.10 所示）就可以查看该类的定义代码（如图 2.11 所示），需要注意的是，该代码并不是实际的定义源代码，而是 Visual Studio 环境根据元数据信息临时生成的模拟代码，但与源代码的内容是一致的。

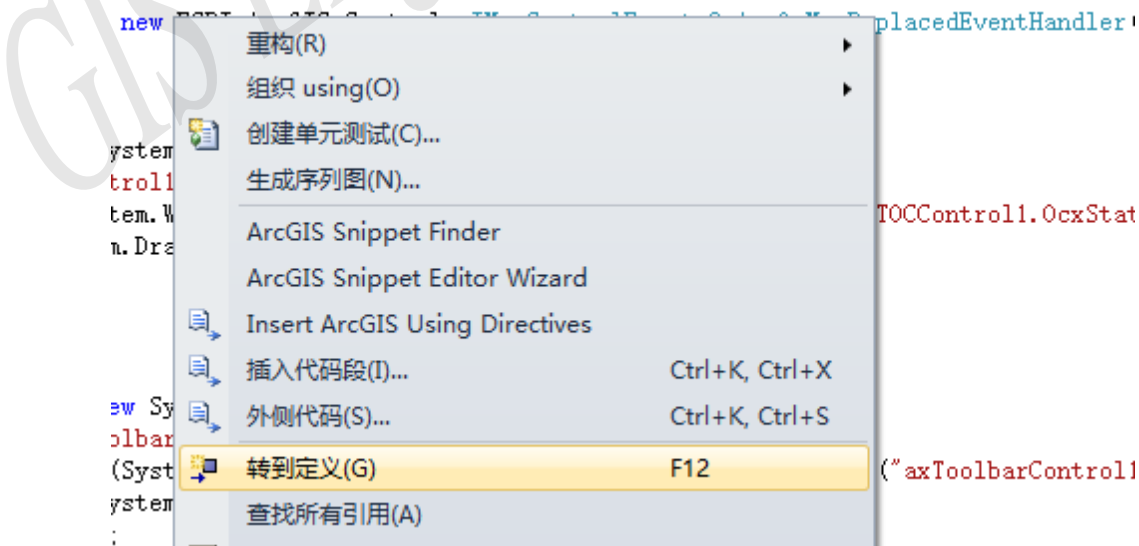


图 2.10. 查看事件委托类的定义代码

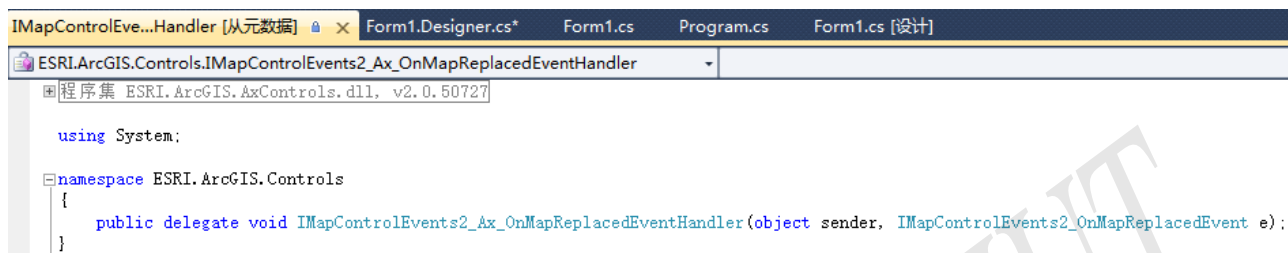


图 2.11. 事件委托类的定义代码

图 2.11 显示的代码是很特别的，因为它不是呈现出“类”的定义语法，而是一种函数定义的语法，这涉及到 C#委托机制的深层次规范，此处不做详细介绍，但是上述语法中函数的参数很值得注意，因为它与事件处理函数的参数是完全一致的（参见图 2.6），这说明事件处理函数的定义实际上也是有章可循的，虽然其函数名称可以自由定义，但是其参数形式必须依照事件委托类中的相关定义。上述的一系列分析将是后面实现 Map 组件事件处理过程的基础。

4. 利用 Map 组件的事件处理实现鹰眼窗口与地图主窗口的数据同步。如果测试一下前述利用 OnMapReplaced 事件实现的地图同步功能就会发现一个 bug，当不是加载 mxd 文件，而是加载一个单独的图层时，同步过程失效了。这说明加载单独的图层是并没有触发 MapControl 控件的 OnMapReplaced 事件，当然我们可以通过监听 MapControl 控件的其他相关事件来解决这一问题，MapControl 控件的主要事件都定义在 IMapControlEvents2 接口中，帮助文档中关于这一接口的事件列表如图 2.12 所示。如果加载了一个可显示的图层，就会引发 MapControl 控件的窗口重绘操作，在重绘前会触发 OnBeforeScreenDraw 事件，在重绘完成后会触发 OnAfterScreenDraw 事件，可以通过监听这两个事件中的某一个来间接的监听加载新图层的状态，并实现鹰眼同步。但是更保险、更直接的方案是直接监听加载新图层的事件，并进行鹰眼同步的响应，遗憾的是 MapControl 控件并不会触发这样的事件。

#### Members

	All <input type="button" value="v"/>	Description
←	<a href="#">OnAfterDraw</a>	Fires after the Map draws a specified view phase.
←	<a href="#">OnAfterScreenDraw</a>	Fires after the Map contained by the MapControl has finished drawing.
←	<a href="#">OnBeforeScreenDraw</a>	Fires before the Map contained by the MapControl starts to draw.
←	<a href="#">OnDoubleClick</a>	Fires when the user presses and releases the mouse button twice in quick succession.
←	<a href="#">OnExtentUpdated</a>	Fires after the extent (visible bounds) of the MapControl is changed.
←	<a href="#">OnFullExtentUpdated</a>	Fires after the full extent (bounds) of the MapControl has changed.
←	<a href="#">OnKeyDown</a>	Fires after a key is pressed on the keyboard.
←	<a href="#">OnKeyUp</a>	Fires after a pressed key is released.
←	<a href="#">OnMapReplaced</a>	Fires after the Map contained by the MapControl has been replaced.
←	<a href="#">OnMouseDown</a>	Fires when the user presses any mouse button while over the MapControl.
←	<a href="#">OnMouseMove</a>	Fires when the user moves the mouse over the MapControl.
←	<a href="#">OnMouseUp</a>	Fires when the user releases a mouse button while over the MapControl.
←	<a href="#">OnOleDrop</a>	Fires when an OLE drop action occurs on the MapControl.
←	<a href="#">OnSelectionChanged</a>	Fires when the current selection changes.
←	<a href="#">OnViewRefreshed</a>	Fires when the view is refreshed before drawing occurs.

图 2.12. IMapControlEvents2 接口中定义的所有事件

实际上 MapControl 控件只是一个地图数据的显示和操作界面，从图 2.12 中可以看出，它所触发的事件都与数据显示和用户交互有关，而真正实现数据管理功能的是该控件中所包含的 Map 组件对象。Map 组件是一个非常重要的 AO 组件，在 AE 二次开发中它是很多地图数据操作调用过程的起始点，在后续的实验中将会详细了解 Map 组件中的常用数据管理功能和使用方法，在本实验中主要用到该组件的加载图层事件。由于数据最终由 Map 组件来管理，因此与数据变化相关的事件往往由 Map 组件对象触发，Map 组件中的主要事件都定义在 IActiveViewEvents 接口中，该接口中定义的事件如图 2.13 所示，而图中用红色方框标示的三个以 Item 开头的事件当由 Map 组件对象触发时，就代表了对图层的相应操作（ItemAdded：添加了一个图层；ItemDeleted：删除了一个图层；ItemReordered：对一个图层进行了重排序），对这些事件进行监听就能直接掌握图层数据变化的各种状态。

## Members

	All ▼	Description
←	AfterDraw	Fired after the specified phase is drawn.
←	AfterItemDraw	Fired after an individual view item is drawn. Example: view page layout.
←	ContentsChanged	Fired when the contents of the view changes.
←	ContentsCleared	Fired when the contents of the view is cleared.
←	FocusMapChanged	Fired when a new map is made active.
←	ItemAdded	Fired when an item is added to the view.
←	ItemDeleted	Fired when an item is deleted from the view.
←	ItemReordered	Fired when a view item is reordered.
←	SelectionChanged	Call this function to fire the selection changed event.
←	SpatialReferenceChanged	Fired when the spatial reference is changed.
←	ViewRefreshed	Fired when view is refreshed before draw happens.

图 2.13. IActiveViewEvents 接口中定义的所有事件

但是 Map 组件并不是控件，无法利用 Visual Studio 中提供的命令操作实现事件处理，需要编写代码实现。结合前述代码分析的结果，以 ItemAdded 事件处理为例提出下列编写步骤：

(1) 引入新的程序集。本实验将操作 Map 组件及相关的事件接口 IActiveViewEvents，这两个类型都定义在 ESRI.ArcGIS.Carto 组件库中（因此使用它们时都要添加 ESRI.ArcGIS.Carto 命名空间），这个组件库可能并没有被自动添加到程序集引用列表中（可以在“解决方法管理器”的“引用”目录下进行查找），需要进行手动添加，主要操作方法是：

a) 在“解决方法管理器”的“引用”目录上点击鼠标右键，在弹出菜单中选择“添加引用”项目，如图 2.14 所示：

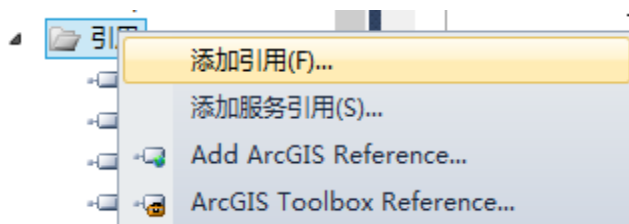


图 2.14. 添加引用菜单

b) 在弹出的“添加引用”对话框中选择“.NET”属性页，如图 2.15 所示：

c) 在属性页包含的列表中选择所需的组件库程序集进行添加，如图 2.16 所示：

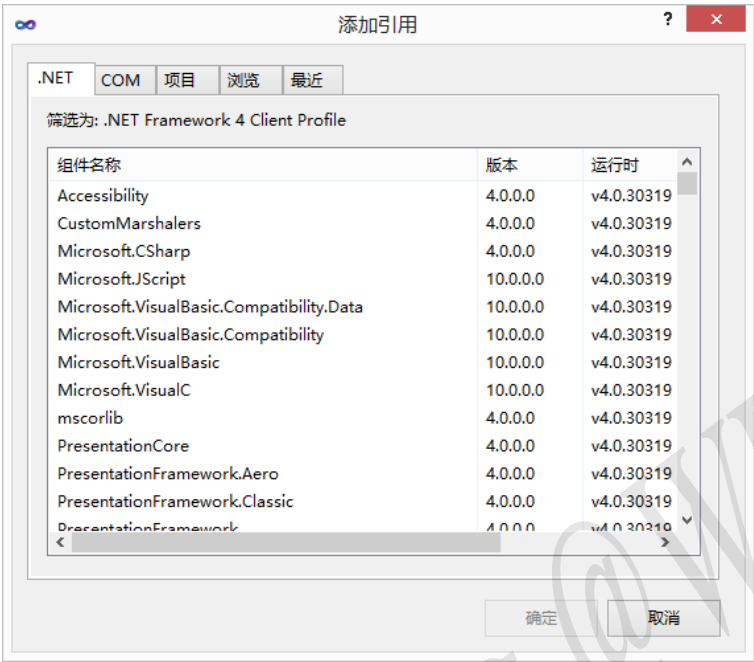


图 2.15. 选择添加“.NET”程序集引用

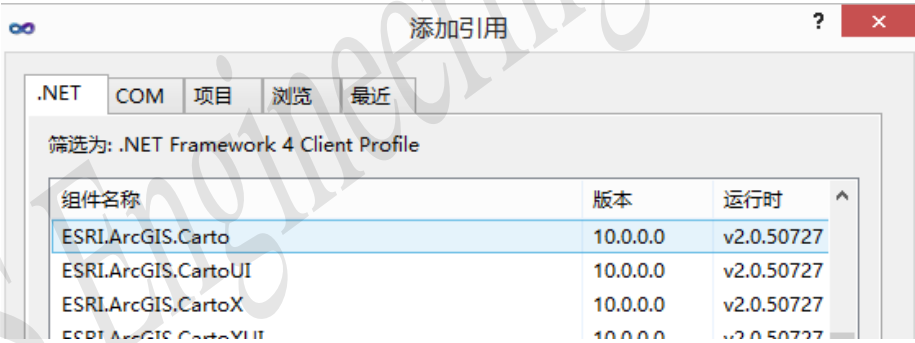


图 2.16. 选择并添加所需的组件库程序集

(2)定义事件监听。参考图 2.8，OnMapReplaced 事件的监听设置代码写在 Form1 窗体设计代码的 InitializeComponent 方法中，我们也可以同样将 ItemAdded 事件的监听设置代码写在这里。由于 MapControl 控件以成员变量的形式包含了 Map 组件对象，而且 IActiveViewEvents 接口定义在 ESRI.ArcGIS.Carto 命名空间中，参考前述分析，仅仅通过模仿图 2.8 中的代码就可以写出 ItemAdded 事件的监听设置代码如下：

CODE 2.3. 模仿 OnMapReplaced 事件编写的 ItemAdded 事件监听设置代码

this.axMapControl1.Map.ItemAdded += new  
ESRI.ArcGIS.Carto.IActiveViewEvents\_ItemAddedEventHandler(this.OnItemAdded);

其中 OnItemAdded 应该是事件处理函数的名称，可以自由命名，但在后面定

义事件处理函数时应保持一致。但如果仅仅这样完全模仿会出现编译错误，因为“axMapControl1.Map.ItemAdded”的写法是错误的，这是由于.Net 环境中替代 MapControl 控件类的 AxMapControl 控件类包含了 OnMapReplaced 事件的定义，因此可以采用“axMapControl1.OnMapReplaced”的写法，但该类中的 Map 组件成员是以 IMap 接口类型定义的，而在 IMap 接口中并没有 ItemAdded 的定义，因此不能采用“axMapControl1.Map.ItemAdded”的写法。解决办法是将 Map 组件对象进行接口转换，转换到 IActiveViewEvents 接口，然后再获取其 ItemAdded 事件，因此正确的代码应该是：

CODE 2.4. 正确的 ItemAdded 事件监听设置代码

```
(this.axMapControl1.Map as IActiveViewEvents_Event).ItemAdded += new  
ESRI.ArcGIS.Carto.IActiveViewEvents_ItemAddedEventHandler(this.OnItemAdded);
```

上述代码有两点值得注意，一是表示接口转换的 as 关键字，另一个是转换成的接口名称（as 后的绿色部分）并不是 IActiveViewEvents 而是 IActiveViewEvents\_Event，即在实际的事件接口后增加了“\_Event”后缀，这也是一条规律，由于.Net 环境对基于 COM 规范的组件需要进行一些修改封装，因此会变更很多名称定义，例如实验一中碰到的 AxMapControl，此处的事件接口后面添加“\_Event”后缀又是一例，AO 帮助文档中显示的所有事件接口当在 C# 中进行调用时都需添加“\_Event”后缀！

如果将上述代码写在 InitializeComponent 方法中可能会引起 Visual Studio 环境的异常，在 Visual Studio2010 中查看 Form1 的窗体设计器时会出现如图 2.17 所示的警告提示，也就是说 Visual Studio 不希望我们随便修改 InitializeComponent 方法（虽然添加的代码没有错），解决方法是可将上述事件监听的设置代码移到 Form1 窗体的 Load 事件处理函数中（参照“实验内容 1”中的操作）。

另外，在 Visual Studio2010 中如果调试程序，可能会在执行到 CODE2.4 所示代码位置时出现如图 2.18 所示的错误信息并退出程序，此时很可能的原因是没有对 ESRI.ArcGIS.Carto 程序集执行“取消嵌入互操作”的操作，解决方法参见实验一的图 2.10。





图 2.17. 警告信息



图 2.18. 执行 ItemAdded 事件监听时出现的错误提示

(3) 定义事件处理函数。根据前述分析, 事件处理函数的名称应与监听设置时命名的一致, 而处理函数的参数形式可以通过事件委托类的定义来查询, 这里的事件委托类是 `IActiveViewEvents_ItemAddedEventHandler`, 查询其定义代码可以得到如图 2.19 的结果:

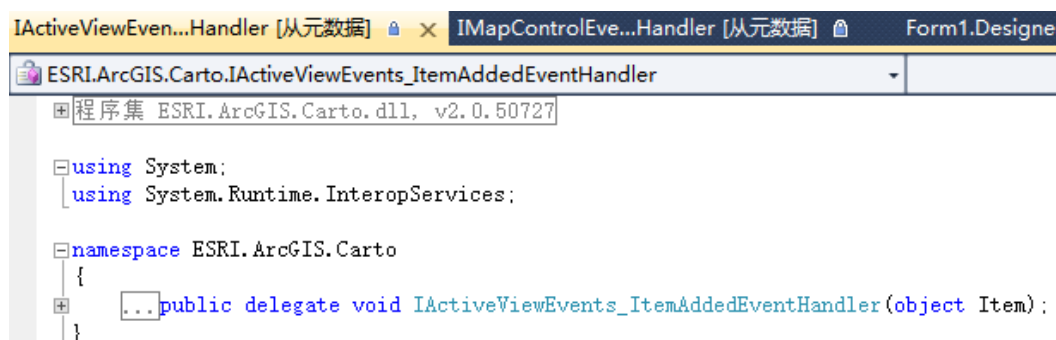


图 2.19. `IActiveViewEvents_ItemAddedEventHandler` 的定义代码

因此在 Form1.cs 中定义的事件处理函数应该是如图 2.20 所示的形式，至于函数中的实现代码请读者自行编写，只提示两点：1) 函数的参数 Item 实际上就是在地图主窗口中新添加的那个图层；2) MapControl 控件的 AddLayer 方法定义如图 2.21 所示。

```
private void OnItemAdded(object Item)
{
    |
}
```

图 2.20. 事件处理函数的定义

```
[C#]public void AddLayer (
    ILayerLayer,
    int toIndex);
```

图 2.21. AddLayer 方法的定义

## 六、思考题

1. 既然 ItemAdded 表示添加一个新图层的事件，为什么没有命名为更直接的 LayerAdded？
2. 请实现当删除图层后的鹰眼地图同步功能。
3. 实验完成后在鹰眼同步功能上还有什么 bug？是什么原因？应如何解决？



## 实验三. Map 组件中的图形元素操作

### 一、实验目的

通过对图形元素的操作，了解 Map 组件在数据管理上的图层化结构，掌握 AO 组件显示空间数据的基本模式：几何图形+显示符号。

### 二、实验仪器

常规配置微机，Windows 操作系统，Visual Studio2005 及以上版本，ArcGIS Engine9.2 及以上版本。本指导书将以 Visual Studio 2010 和 ArcGIS Engine10.0 为基本开发环境，在其他版本的 Visual Studio 和 ArcGIS Engine 中进行开发的过程仅有细微差别，读者可自行查找相关技术资料进行详细了解和处理。

### 三、实验要求

熟悉 AO 控件和组件的事件处理方法，能够比较熟练查询帮助文档，掌握基本的组件调用 C#语法。本次实验的编程过程比较简单，但重在理解 AO 组件显示空间数据的基本模式，掌握几何体操作和显示符号操作，为后续学习矢量要素渲染打基础。

### 四、实验内容

本次实验的主要内容是在前次实验的基础上实现在鹰眼窗口中实时显示地图主窗口显示范围标志，并能在鹰眼中利用鼠标控制主窗口显示范围的功能，效果如图 3.1 所示，分为以下两个方面：

1. 利用图形元素在鹰眼窗口中绘制表示地图主窗口显示范围的矩形框；
2. 利用事件处理方法在鹰眼中跟踪和控制地图主窗口显示范围变化。

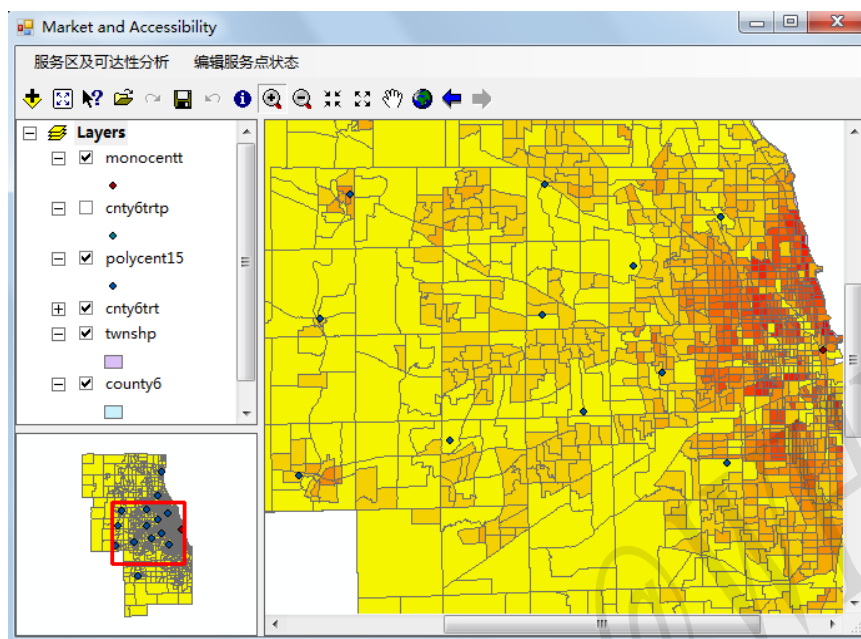


图 3.1. 实验效果示意图

## 五、实验步骤

1. 利用图形元素在鹰眼窗口中实时绘制表示地图主窗口显示范围的矩形框。  
MapControl 控件实现的 IMapControlDefault 接口中定义了一个 Extent 属性，如果在 C# 中使用的是 AxMapControl 控件，可以之间通过控件对象方法获取这个属性。该属性表示当前视窗中显示的地图范围，通过查阅帮助文档可知这个属性是一个 IEnvelope 接口类型的组件对象。在 AO 组件库中只有一个组件实现了 IEnvelope 接口，即 Envelope 组件，它表示了一种特殊的矩形，这种矩形的边框与显示窗口的边框是平行的，常常用来表达其他空间图形的最小外包矩形（MER），如图 3.2 所示，我们可以获取地图主窗口的 Extent 属性并将其矩形绘制在鹰眼窗口中，这样就实现了在鹰眼中标志地图显示范围的效果。

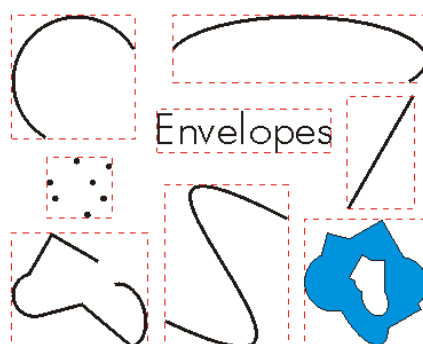


图 3.2. Envelope 组件的图形效果

但是，在 MapControl 中显示一个几何图形并不是一个像拿笔在纸上画图一般简单而直接的过程，实现几何图形的显示需要符合 MapControl 中的矢量数据可视化模式。总的来说，这一可视化模式在数据组织上采用了图层化结构，而在

显示效果上采用了“几何体+显示符号”组合结构。

(1) 在“图形元素”图层上绘制临时图形。前面实验中介绍过，MapControl 控件只是一个数据的显示和操控界面，真实的功能主要由它所包含的 Map 组件对象实现的，因此 MapControl 的可视化数据组织模式就是由 Map 组件所定义的。Map 组件采用了图层化的数据组织结构，各图层的可视化图像分层叠加在一起合成最终的显示结果，要在 MapControl 中显示图形，首要解决的问题是确定将图形数据放在 Map 组件中的哪个图层上，例如当我们每次加载一个新的 Shapefile 文件就会在 Map 中创建一个新的图层来表达和管理其中的空间数据。Map 组件可以加载并管理不同类型的地理信息数据图层，例如矢量要素、栅格数据、TIN 等。无论添加多少个图层，Map 组件都会在最顶部预留一个特殊的图层，用来显示一些地图上的辅助标识，例如注记、图框或其他临时图形标志。这些标识主要作为图形化的标记，没有丰富的属性信息，也不与特定的地理对象相联系，严格的说不属于地理信息数据，因此操作起来也相对简单。AO 将它们定义为“图形元素”，并设计了一批组件来管理和操作各种图形元素，这批组件的共同点是都实现了 IElement 接口。本次实验的主要技术路线就是将主窗口显示范围用图形元素的形式显示在鹰眼窗口中。

图形元素图层的 management 功能主要是定义在 IGraphicsContainer 接口中，其主要成员如图 3.3 所示，Map 组件实现了这个接口，利用这些方法，可以制定出在鹰眼窗口中显示主窗口范围矩形框的主要代码框架，可以参考代码示例 CODE3.1。

Members		
	All	Description
←	AddElement	Add a new graphic element to the layer.
←	AddElements	Add new graphic elements to the layer.
←	BringForward	Move the specified elements one step closer to the top of the stack of elements.
←	BringToFront	Make the specified elements draw in front of all other elements.
←	DeleteAllElements	Delete all the elements.
←	DeleteElement	Delete the given element.
←	FindFrame	Find the frame that contains the specified object.
←	GetElementOrder	Private order object. Used to undo ordering operations.
←	LocateElements	Returns the elements at the given coordinate.
←	LocateElementsByEnvelope	Returns the elements inside the given envelope.
←	MoveElementFromGroup	Move the element from the group to the container.
←	MoveElementToGroup	Move the element from the container to the group.
←	Next	Returns the next graphic in the container.
←	PutElementOrder	Private order object. Used to undo ordering operations.
←	Reset	Reset internal cursor so that Next returns the first element.
←	SendBackward	Move the specified elements one step closer to the bottom of the stack of elements.
←	SendToBack	Make the specified elements draw behind all other elements.
←	UpdateElement	The graphic element's properties have changed.

图 3.3. IGraphicsContainer 接口成员列表

### CODE 3.1. 鹰眼中用矩形框展示主窗口显示范围

```
// 获取地图主窗口的显示范围
IEnvelope Ext = axMapControl1.Extent;
// 由于 Ext 是一个 Envelope 组件对象，不是图形元素，
// 因此不能直接将 Ext 添加到图形元素图层中，
// 需要根据 Ext 的几何图形创建对应的图形元素组件对象。
// 下面的代码表示根据主窗口显示范围 Ext 创建矩形框图形元素，
// 创建过程后文详细描述。
// IElement ExtElement = .....
// 将鹰眼窗口控件包含的 Map 组件对象转换到 IGraphicsContainer 接口
IGraphicsContainer Graphics = axMapControl2.Map as IGraphicsContainer;
// 删除所有鹰眼窗口控件上的所有图形元素（假定仅仅需要显示范围框）
// 因为显示范围会动态变化，因此每次显示时要将旧的矩形框删掉
Graphics.DeleteAllElements();
// IGraphicsContainer 接口定义了一个图形元素列表来管理所有的图形元素，
// 利用 AddElement 方法将指定的元素添加到列表的指定位置上，
// 位置用索引值表示，索引值从 0 开始，
// 由于本例中只有一个图形元素，因此简单的将其添加到列表的第一个。
Graphics.AddElement(ExtElement, 0);
// Refresh 方法用来刷新控件窗口中的内容，以便让新添加的图形元素立即显示。
axMapControl2.Refresh(esriViewDrawPhase.esriViewGraphics, Type.Missing,
Type.Missing);
```

（2）正确显示 Extent 矩形框。接下来的关键问题就是如何利用表示主窗口范围的 Ext 变量创建图形元素。首先思考一个问题：既然图形元素是用来显示几何图形的，那为什么又不能直接将几何图形添加到地图中进行显示呢？这个问题的答案涉及到 AO 中处理图形显示效果的基本模式。实际上，从严格的数学意义上说，几何图形是不可见的，因为它只具有空间位置信息，没有光谱信息，是不可能被视觉所观测到的，而当我们绘制一个几何图形时，实际上是利用了诸如颜色、线型、填充、标志等辅助的光谱信息，因此 AO 在处理图形显示效果时也对几何图形和光谱信息进行了严格区分，必须将两者配合起来才能实现图形的正确显示，同时，通过不同的光谱信息设置，能够以不同的效果显示同一份的空间数据。

AO 中设计了两大类组件分别处理几何图形和用来显示几何图形的光谱信息，前者统称为“几何体组件”，后者统称为“显示符号组件”。所有的几何体组件都

实现了 IGeometry 接口，而所有的显示符号组件都实现了 ISymbol 接口。Envelope 组件实现了 IGeometry 接口，因此它是一个几何体组件，它的作用是用来表达矩形。根据 Envelope 组件对象来创建图形元素的主要过程分为两步，首先创建对应的图形元素组件对象，然后设置几何体及相应的显示符号。

Envelope 组件对象是一个矩形框，而在 AO 组件库中定义了一种专门对应于矩形的组件——RectangleElement 组件，这是一个 CoClass 类型的组件，可以直接采用 new 的方法进行实例化，在 C#环境中实例化一个 RectangleElement 组件对象的方法参考代码示例 CODE3.2.

CODE 3.2. 实例化 RectangleElement 组件对象
IElement ExtElement = new RectangleElementClass();

此处需要注意的是组件名称后面的 Class 后缀，这是在.NET 环境下调用 AO 中 CoClass 类型组件的语言规范之一，由于.NET 环境对 COM 组件的特殊处理，在任何 CoClass 组件类的名称后都要加上 Class 后缀，例如有一个 AO 组件叫做 FeatureClass，那么在 C#环境中应该改为 FeatureClassClass。

另外，需要提醒的是，在 Visual Studio2010 环境下编译上述代码时可能会出现“CS1752: 无法嵌入互操作类型“ESRI.ArcGIS.Carto.RectangleElementClass”。请改用适用的接口。”的编译错误，解决方案也是取消 ESRI.ArcGIS.Carto 引用的嵌入互操作，具体方法参见实验一的图 1.10。

接下来看看图形元素的公共接口 IElement，在它的成员列表中（如图 3.4）中定义了一个 Geometry 属性，就是用来设置图形元素对应的几何体，那么这个列表中哪个属性或方法是用来设置显示符号的呢？答案是“没有”，因为利用 AO 可以显示非常丰富的图形效果，因此不同类型图形元素的显示符号差异化较大，AO 组件库将显示符号的设置放在各个不同的图形元素组件中进行个性化处理。

Members	
	All
←	Activate
←	Deactivate
←	Draw
■	Geometry
←	HitTest
■	Locked
←	QueryBounds
←	QueryOutline
■	SelectionTracker

图 3.4. IElement 接口的成员列表

本实验不对各种图形元素的显示符号设置技巧进行深入全面的讲解，仅仅通过矩形框显示问题来体会显示符号设置的基本思路。我们创建了 RectangleElement 组件对象来表达矩形框图形元素，在这个组件中设置显示符号的部分定义在其实现的 IFillShapeElement 接口中，这个接口很常用，从名称上看它是一种具有可填充形状的图形元素，因此它主要是用来显示平面区域的。这个接口仅有一个 Symbol 属性（如图 3.5），从名称上看就知道是用来设置显示符号的。

Members	
	All
■	Symbol

图 3.5. IFillShapeElement 接口的成员列表

但是进一步查看帮助文档可知，这个 Symbol 属性的类型不是 ISymbol 而是 IFillSymbol，这一点验证了前面提到的“显示符号类型应与图形元素类型相关”的说法，可填充形状应具有填充符号。那么应该如何设置 IFillSymbol 类型的属性呢？可进一步查看 IFillSymbol 接口的帮助文档，从而了解到该接口仅有两个成员，分别是 Color 属性和 Outline 属性（如图 3.6），

Members		Description
■	Color	Fill color.
■	Outline	Line symbol of fill outline.

图 3.6. IFillSymbol 接口成员

再进一步深入查询,Color 属性是一个 IColor 类型的对象,用于设置填充颜色,Outline 是一个 ILineStyle 类型的对象,用于设置边界线的显示符号,因此 IFillSymbol 类型的对象实际上是一种组合符号,包含了区域内部的填充符号和外围的边界线显示符号。另一方面,我们知道,在区域内部进行填充时,除了填充颜色,往往还需要定义填充图案,例如斜线填充、阴影填充、点集填充等,这又该如何设置呢? 填充图案不能通过 IFillSymbol 接口来设置,AO 中有多个组件实现了 IFillSymbol 接口(如图 3.7),它们代表了各种具体的填充形式,关于填充的细节设置由这些特定组件所实现的其他接口来完成,此处不做详细介绍。如果仅仅不需要填充图案,仅仅用画刷涂满同一种颜色,可以使用最简单的 SimpleFillSymbol 组件,在我们的实验中选择了这一组件来表达矩形框的显示效果。同理,作为边框属性类型的 ILineStyle 接口也是一个线型符号的通用接口,多种线型符号组件都实现了这一接口,在具体应用时应根据需要先创建一个对应的线型符号组件对象,然后进行相应的设置。在我们的实验中选择最简单的 SimpleLineStyle 组件对象来表达矩形框的边框显示效果。具体的矩形框图形元素实现代码可以参考代码示例 CODE3.3.:

CODE 3.3. 设置红色矩形框的显示符号

```
// 变量 ExtElement 及 Ext 参照 CODE3.1 及 3.2 中的定义
ExtElement.Geometry = Ext;
// 实例化一个 RgbColor 组件对象用于表示颜色
IRgbColor OutlineClr = new RgbColorClass();
// 定义为红色 (Red)
OutlineClr.Blue = 0;
OutlineClr.Red = 255;
OutlineClr.Green = 0;
// 实例化一个 SimpleLineStyle 组件对象用于表示矩形边框的显示符号
ILineSymbol Outline = new SimpleLineStyleClass();
// 边框颜色为红色,线宽为 2,其他线型选择默认设置
Outline.Color = OutlineClr as IColor;
Outline.Width = 2;
// 实例化一个 SimpleFillSymbol 组件对象用于表示矩形的完整显示符号,
// 在本实验的例子中只要求矩形以边框的形式显示,内部没有填充,
// 但由于矩形图形元素要求设置内部填充效果,
// 本实验的方案是将其设置为完全透明。
IFillSymbol ExtFill = new SimpleFillSymbolClass();
```



```
// 矩形显示符号的边框采用前面创建的 SimpleLineSymbol 组件对象
ExtFill.Outline = Outline;

// 将前面定义的红色颜色组件设置为透明色，然后将其作为矩形的填充色。
// Transparency 属性值从 0~255 表示从完全透明至完全不透明。
OutlineClr.Transparency= 0;
ExtFill.Color = OutlineClr as IColor;

// 将矩形图形元素的显示符号设置为前面创建的 SimpleFillSymbol 组件对象。
(ExtElement as IFillShapeElement).Symbol = ExtFill;
```

Classes that implement IFillSymbol	
Classes	Description
<a href="#">ColorRampSymbol (esriCarto)</a>	ESRI ColorRampSymbol for raster rendering.
<a href="#">ColorSymbol (esriCarto)</a>	ESRI ColorSymbol for raster rendering.
<a href="#">DotDensityFillSymbol</a>	Defines a dot density fill symbol, a data driven symbol commonly used with the dot density renderer.
<a href="#">GradientFillSymbol</a>	A fill symbol composed from a ramp of colors.
<a href="#">LineFillSymbol</a>	A fill symbol comprised of any of the supported line symbols.
<a href="#">MarkerFillSymbol</a>	A fill symbol comprised of any of the supported marker symbols.
<a href="#">MoleFillSymbol (esriDefenseSolutions)</a>	Mole Fill Symbol Class.
<a href="#">MultiLayerFillSymbol</a>	A fill symbol that contains one or more layers.
<a href="#">PictureFillSymbol</a>	A fill symbol based on either a BMP or an EMF picture.
<a href="#">RasterRGBSymbol (esriCarto)</a>	ESRI RasterRGBSymbol for raster rendering.
<a href="#">SimpleFillSymbol</a>	A fill symbol comprised from a predefined set of styles.
<a href="#">TextureFillSymbol (esri3DAnalyst)</a>	Texture Fill Symbol component.

图 3.7. 用于表达填充模式的所有显示符号组件

2. 利用事件处理方法在鹰眼中跟踪和控制地图主窗口显示范围变化。将上述代码示例 CODE3.1-3.3 综合起来就可以实现用红色矩形框在鹰眼中展示地图主窗口的显示范围，但这只是一次显示的效果，作为一种常规的鹰眼跟踪效果，更重要的是根据主窗口的变化实时显示矩形框，同时经常也需要反过来利用在鹰眼中绘制矩形框的方法控制主窗口的显示范围。

（1）在鹰眼中跟踪主窗口显示范围的变化。这是一个典型的事件处理过程，我们需要的找到主窗口显示范围变化后所触发的事件，当 MapControl 的显示范围发生变化后会触发一个 OnExtentUpdated，这是一个非常简单事件处理过程，实现方法此处略过。

（2）在鹰眼中控制主窗口的显示范围。根据实际应用可以定义出多种在鹰眼中控制主窗口显示范围的操作方案，本次实验介绍最常用也最简单的方案：通过在鹰眼中用鼠标绘制矩形框来将主窗口的显示范围。之所以是最简单的方案，是因为在 MapControl 用鼠标绘制矩形是非常简单的事，这个控件提供了一系列的 Track\*\*\* 方法（如图 3.8）用于在控件窗口内用鼠标绘制图形，其中的 TrackRectangle 方法可以根据鼠标左键按下时的起始点和鼠标左键弹起时的终止点作为矩形的对角顶点创建一个临时的矩形，该方法在矩形创建完后返回一个



IEnvelope 类型的对象表示这个矩形，由于 MapControl 的 Extent 属性是一个可读写属性，我们只需将这个新创建的矩形设置为主窗口的 Extent，并同时在鹰眼中用红色矩形框显示出来就可以实现鹰眼的反向控制了。

在何时调用 TrackRectangle 方法比较合适？作为鼠标操作过程，应该在鼠标左键按下时立即调用该方法，然后 MapControl 控件就会自动跟踪鼠标移动和左键弹起的操作，完成矩形创建。在 MapControl 控件内部按下鼠标左键会触发一个 OnMouseDown 事件，因此这这也是一个简单的事件处理过程，此处略过详细代码，只是提醒读者，这个事件应是鹰眼对应的 MapControl 控件所触发的。

←	TrackCircle	Rubber-bands a circle on the MapControl.
←	TrackLine	Rubber-bands a polyline on the MapControl.
←	TrackPolygon	Rubber-bands a polygon on the MapControl.
←	TrackRectangle	Rubber-bands a rectangle on the MapControl.

图 3.8. MapControl 控件提供的 Track\*\*\*方法

六、思考题

- 1.按照本实验的解决方案，当初始加载地图后并不会在鹰眼中显示范围框，只有当在主窗口执行了缩放、平移等操作后才会出现矩形框，如何能改变这种情况？
- 2. 按照本实验的解决方案，当加载新地图后，原有的范围框不会消失，并且显示的是错误的范围，如何解决这种情况？
- 3. 本实验的解决方案还有没有其他 bug？应如何解决？

## 实验四. 利用 Geodatabase API 读取 shapefiles 数据

### 一、实验目的

通过 shapefiles 数据的读取操作，认识 Geodatabase 数据模型下矢量数据的基本管理结构 and 应用特征，理解 Geodatabase 数据模型的基本框架，掌握 AO 组件中的 Geodatabase API 基本调用规则。

### 二、实验仪器

常规配置微机，Windows 操作系统，Visual Studio2005 及以上版本，ArcGIS Engine9.2 及以上版本。本指导书将以 Visual Studio 2010 和 ArcGIS Engine10.0 为基本开发环境，在其他版本的 Visual Studio 和 ArcGIS Engine 中进行开发的过程仅有细微差别，读者可自行查找相关技术资料进行详细了解和处理。

### 三、实验要求

了解 ArcGIS 中矢量要素的“空间+属性”数据组织方式，了解 shapefiles 数据的基本结构，掌握关系型数据模型的基本概念。

### 四、实验内容

除去面向对象的一些特征外，Geodatabase 的概念框架完全植根于关系型 DBMS 的相关理论，在关系型 DBMS 的结构中，数据存储的基本结构是：数据库->数据集的集合->数据集->数据记录，其中的“->”表达包含关系。数据集一般是以数据表的形式表达的，由一系列字段构成。上述概念在 AO 的 Geodatabase API 中都有对照的接口，数据库对应于 IWorkspace，数据集的集合对应于 IDatasets，数据集对应于 IDataset，而字段对应于 IField，在矢量要素环境下数据记录对应于 IFeature，有不同的组件实现了上述接口，也有的组件同时实现了上述接口中的多个接口。本实验将通过 Shapefiles 数据的读取操作，分别介绍相关组件和接口的调用方法。

1. 利用 Geodatabase API 添加 shapefiles 图层。
2. 读取矢量要素的字段信息。

### 五、实验步骤

1. 实验目标。本次实验在前次实验的项目上进行扩展，实现的新功能包括：
  - (1) 当鼠标右键点击 TOCControl 对象顶部的 Map 项时，显示带有“添加 shape 图层”项的右键菜单（如图 4.1）；选择该项将显示一个“打开文件”对话框，并由用户选择一个\*.shp 文件（如图 4.2）；选择“打开”按钮后，将选定文件的数

据作为新的矢量要素图层添加到地图主窗口中（如图 4.3）；

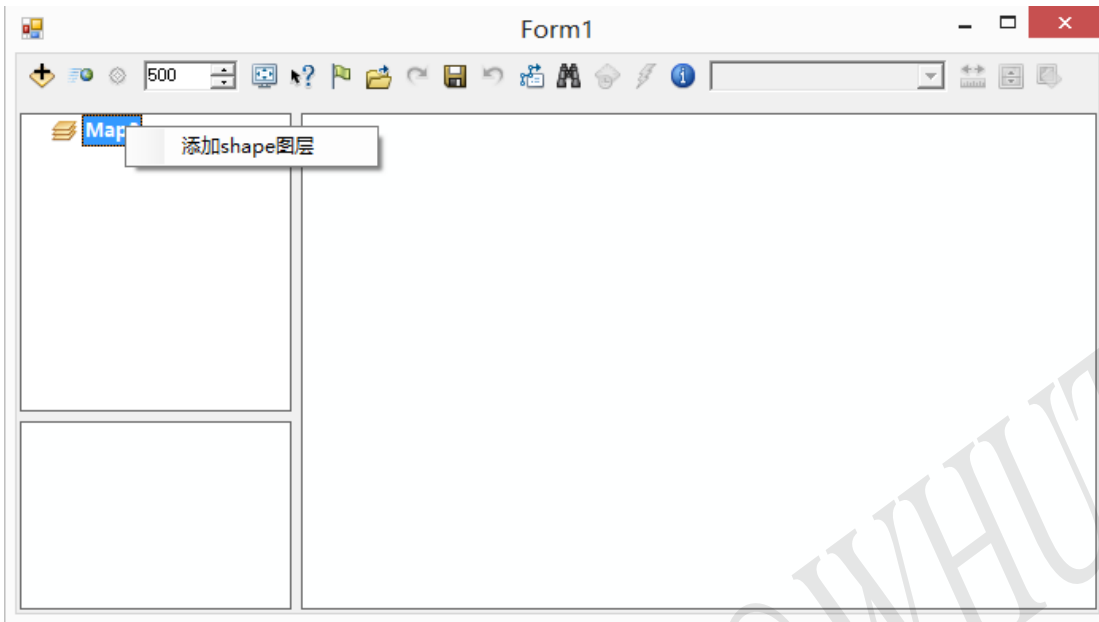


图 4.1. Map 项上的右键菜单效果

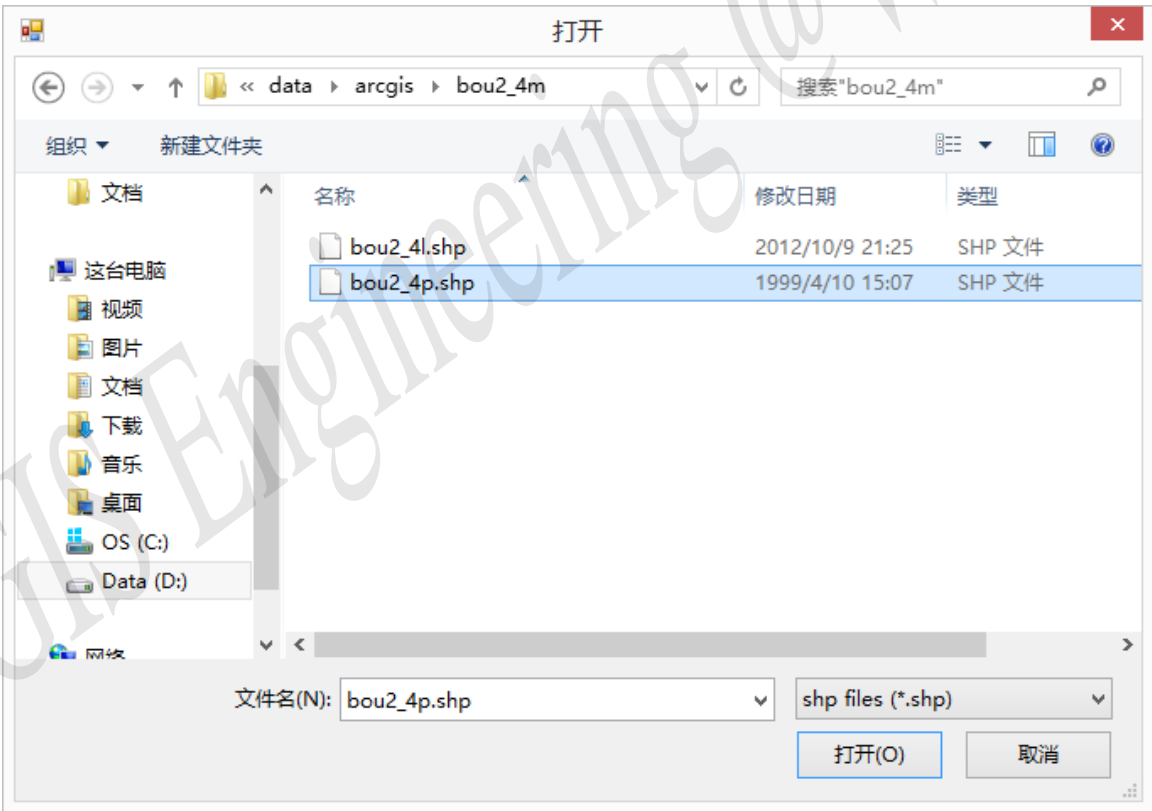


图 4.2. 选择 shp 文件的效果

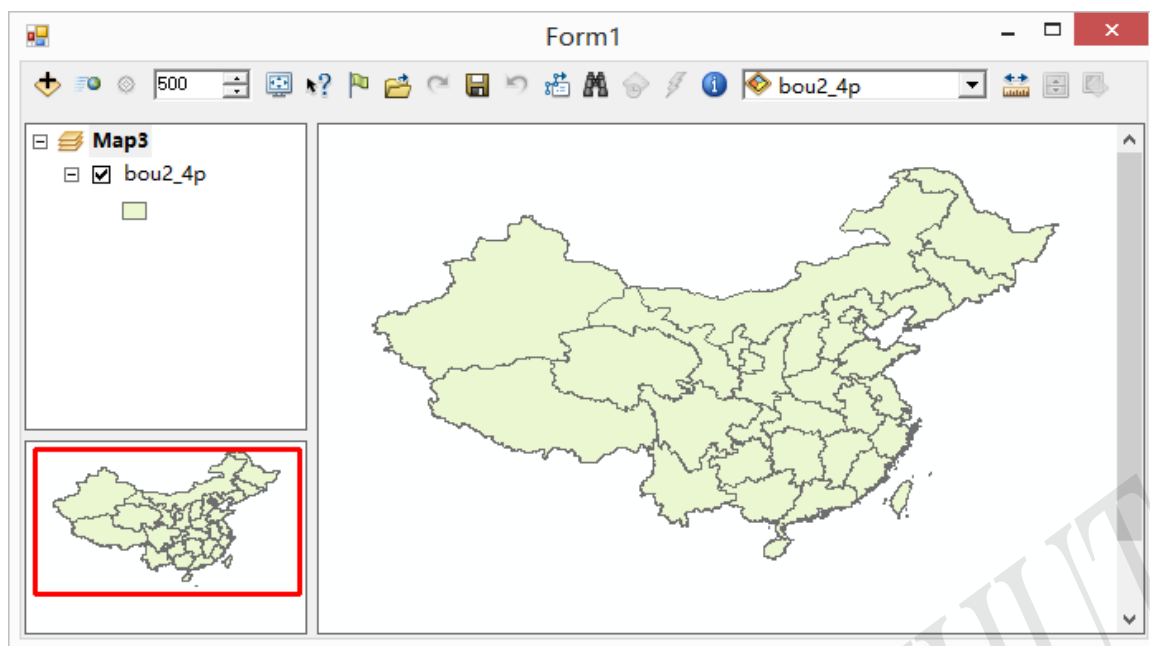


图 4.3. 添加图层后的效果

(2) 当鼠标右键点击 TOCControl 对象中的图层项时，显示带有“打开属性表”项的右键菜单（如图 4.4）；选择该项将显示一个新的“图层属性表”对话框，对话框中有一个 MapControl 窗口显示目标图层的空间信息，并作为空间查询的操作界面，同时表格的形式显示点击的图层中的所有矢量要素信息，表格的下部安排了操作属性数据的一些控件。本次实验将仅列出所有字段（如图 4.5），其他的属性值获取、查询和编辑功能在后续实验中进行介绍。

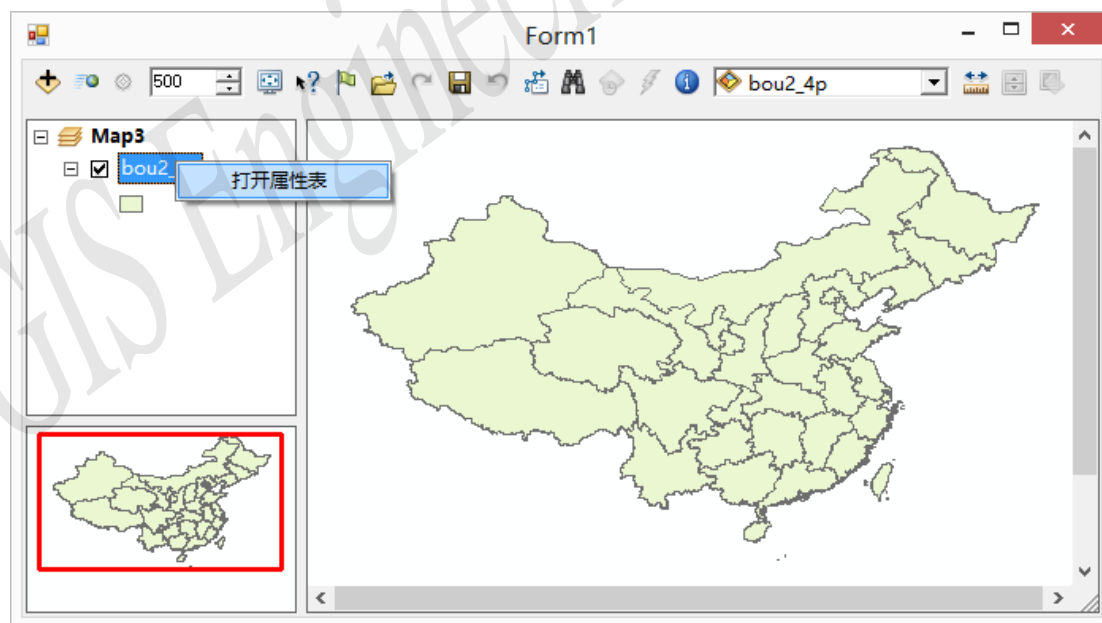


图 4.4. 图层项上的右键菜单效果

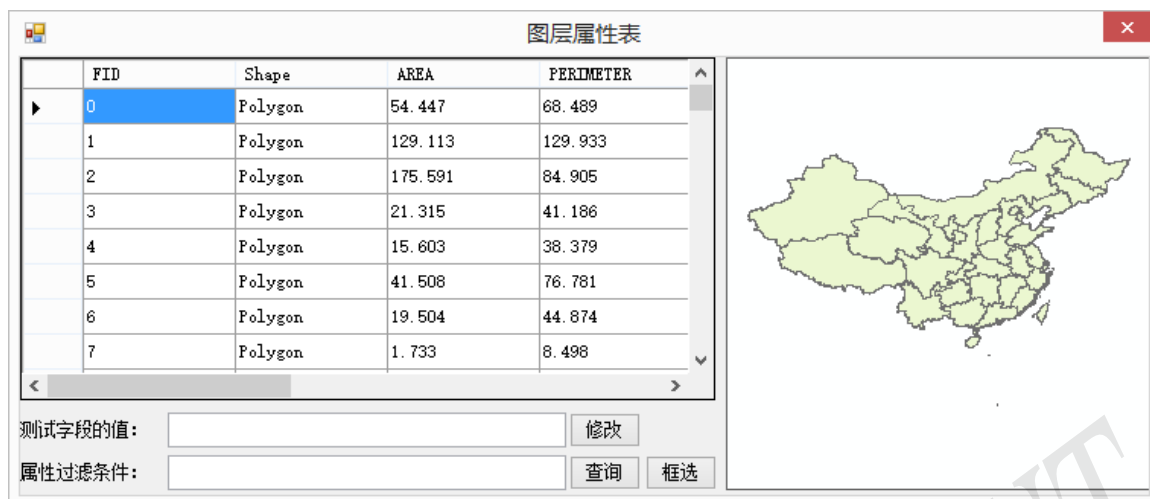


图 4.5. “图层属性表”对话框效果

2. 相关 Windows 菜单和窗体设计工作。本次实验需要在前次实验项目基础上添加新的 Windows 交互界面，并完成相应的事件处理过程。

(1) 添加“图层属性表”对话框。首先在实验项目内添加一个新的 Windows 窗体，操作过程如图 4.6-4.7，打开窗体设计器，将窗体的 Name 属性修改为“LayerAttrib”，将窗体的 Text 属性修改为“图层属性表”（如图 4.8）。然后向窗体中拖入一个 DataGridView 控件，并调整为合适的尺寸（如图 4.9）。

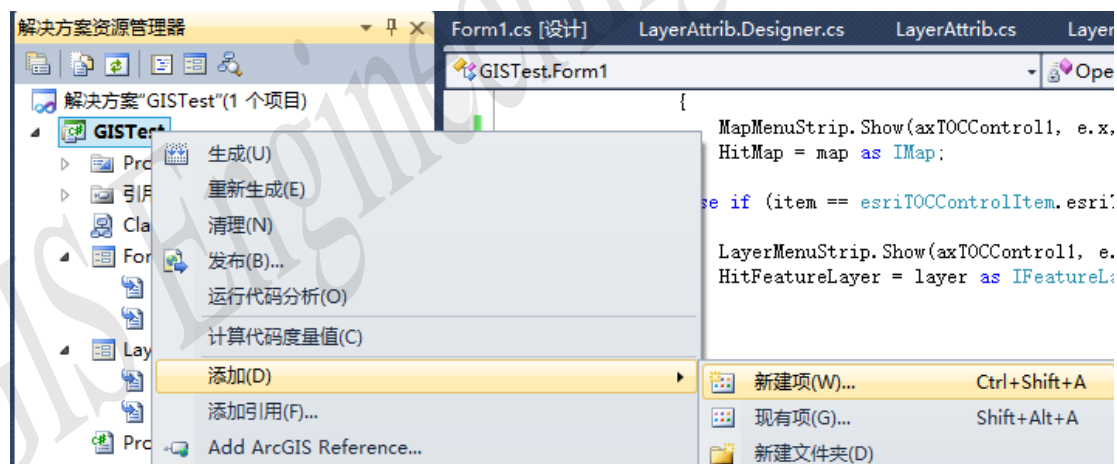


图 4.6. 在实验项目下添加新项

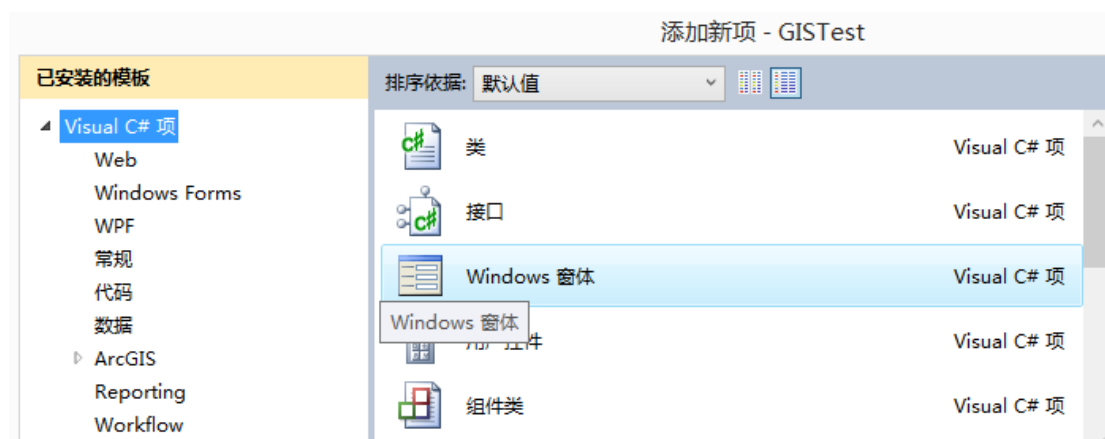


图 4.7. 选择添加 Windows 窗体

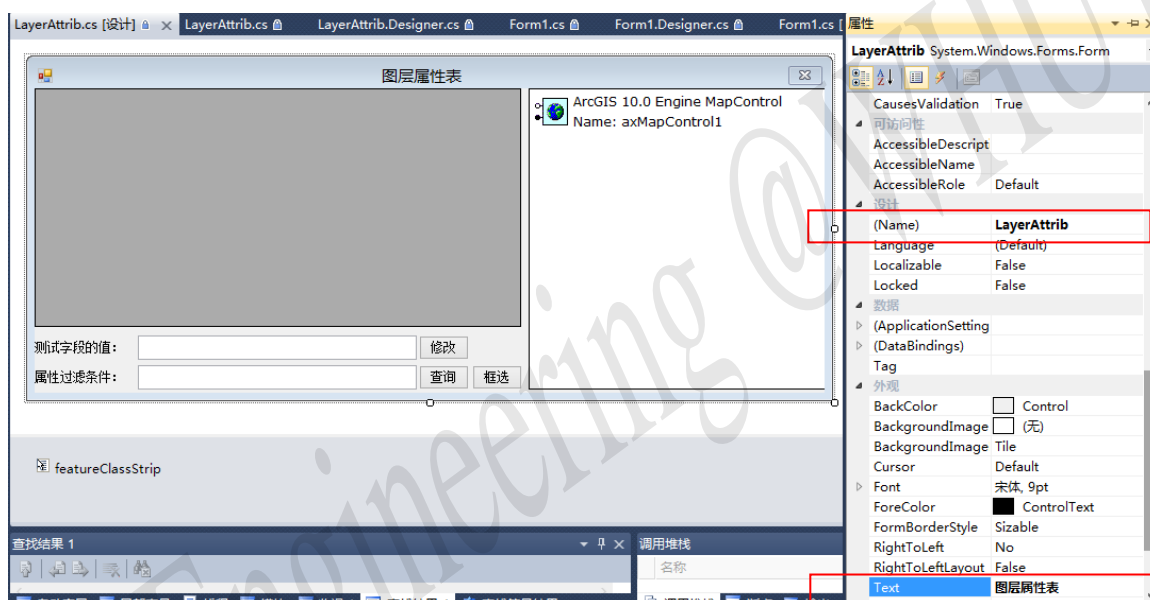


图 4.8. 修改新窗体的 Name 和 Text 属性

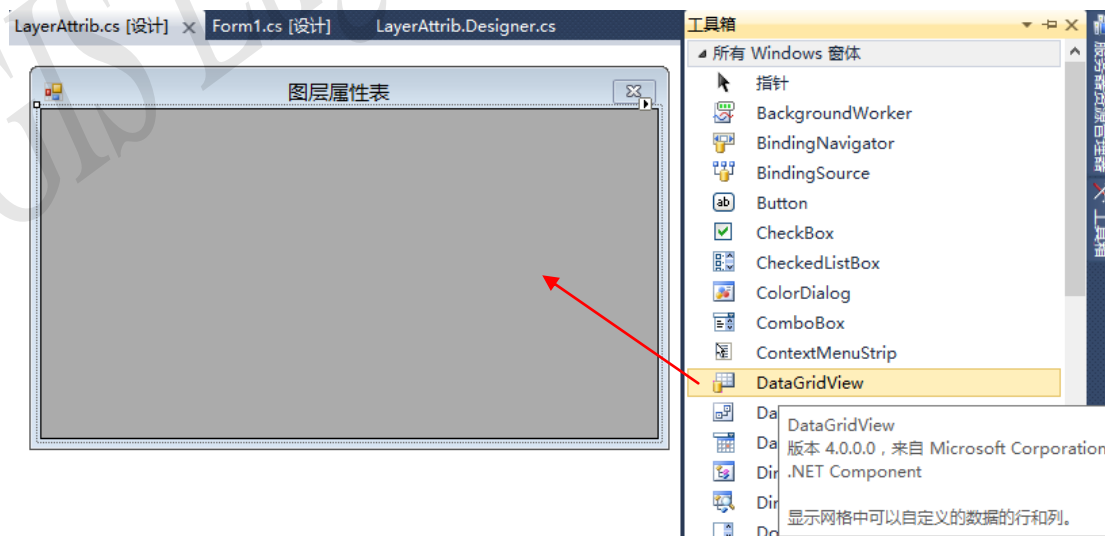


图 4.9. 添加 DataGridView 控件

添加对话框之后，在 LayerAttrib.cs 源文件中添加一个新的方法 ShowFeatureLayerAttrib 用来显示矢量要素图层的属性表（如图 4.10，注意使用 using 语句引入必要的 AO 库命名空间）

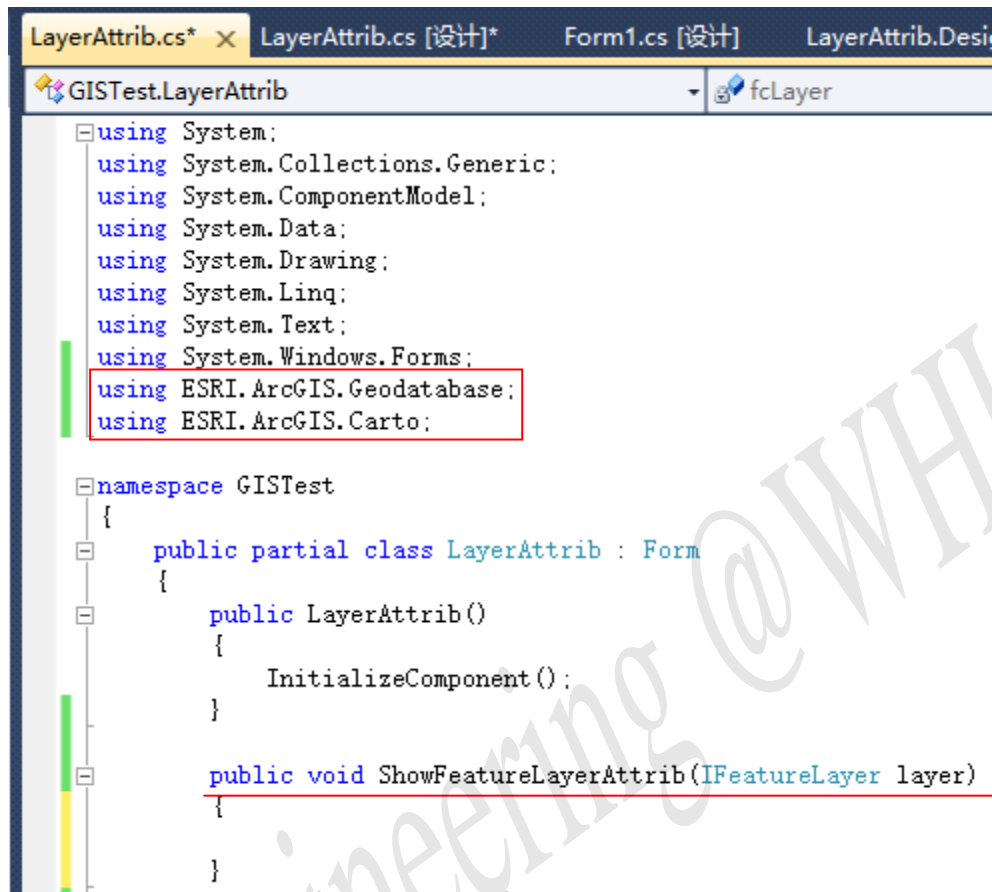


图 4.10. 在 LayerAttrib 窗体源文件中添加新的方法

(2) 添加右键菜单。首先在“工具箱”中找到 ContextMenuStrip 控件，并拖入 Form1 的窗体设计器中，将其 Name 属性修改为“MapMenuStrip”，加入一个“添加 Shape 图层”子项（如图 4.11.），然后仿造上述步骤拖入一个 Name 为“LayerMenuStrip”的 ContextMenuStrip 控件，加入一个“打开属性表”的子项。

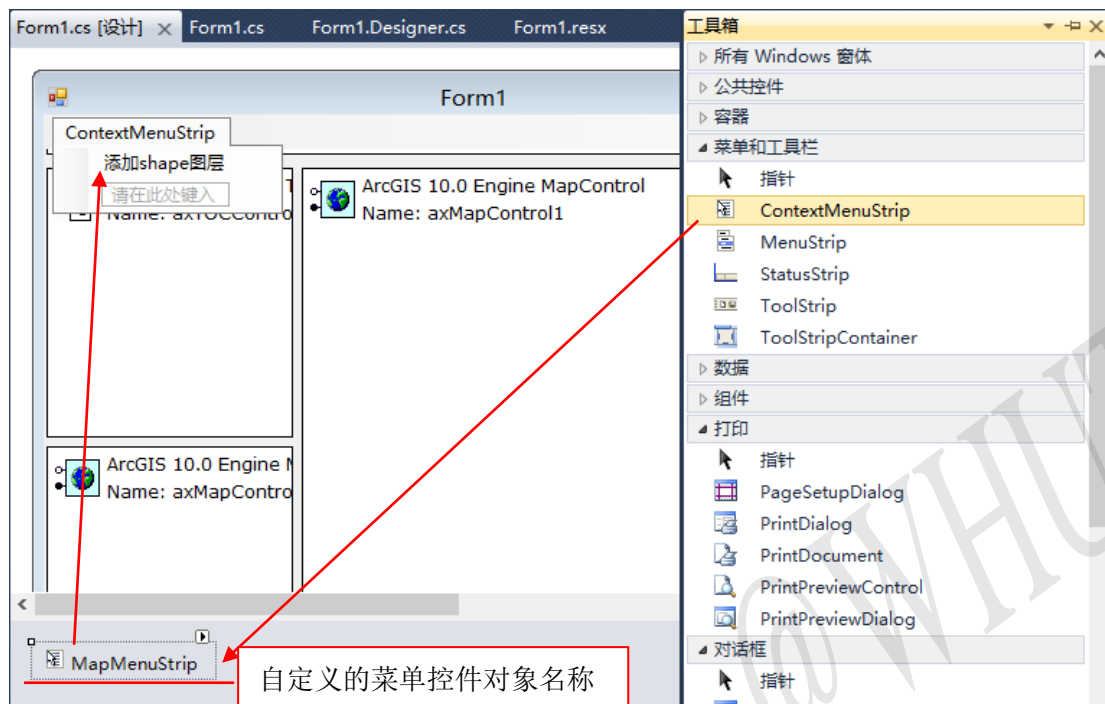


图 4.11. 添加 Map 项的右键菜单

由于上述右键菜单要在 TOCControl 控件中点击鼠标右键时显示，因此是一个标准的事件处理过程。当鼠标在 TOCControl 对象内部点击时会触发 OnMouseDown 事件（按下时）和 OnMouseUp 事件（弹起时），只需在任一个事件处理函数中实现菜单的显示就可以了。而其中的难点在于判断鼠标右键点击的是 TOCControl 中的哪个项目，这一过程可由 TOCControl 控件的 HitTest 方法实现。

首先，在 Form1.Designer.cs 源文件的成员定义部分添加两个新的成员变量，用于保存在 TOCControl 中点中的项目（如图 4.12，HitMap 表示点中的地图项目，HitLayer 表示点中的图层），这两个变量的作用是在鼠标点击事件处理函数和其他的 Form1 类方法间传递数据，然后在 Form1.cs 源文件的构造函数中对上述成员变量进行初始化（如图 4.13）





图 4.12. 在 Form1 中添加新的成员变量

```

public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        HitMap = null;
        HitLayer = null;
    }
}

```

图 4.13. 成员变量的初始化

参考代码示例 CODE4.1 实现判断鼠标右键点中的项目，并弹出相应菜单的功能，该示例是 OnMouseDown 事件的处理函数。

CODE4.1. 在 TOCCControl 对象中判断鼠标点中的项目并弹出右键菜单

```

private void axTOCCControl1_OnMouseDown(object sender,
                                         ITOCCControlEvents_OnMouseDownEvent e)
{
    if (e.button == 2)
    {
        esriTOCCControlItem item = esriTOCCControlItem.esriTOCCControlItemNone;
        IBasicMap map = null;
        ILayer layer = null;
        System.Object other = null;
        System.Object index = null;
        // HitTest 方法用于按照鼠标指针的位置进行鼠标点击测试，

```

```

// 该方法的前两个参数表示鼠标位置的 x 和 y 坐标，
// 接下来的三个参数是 ref 类型，可以作为返回值，分别表达了：
// item 参数：点中的项目类型；
// map 参数：点中的 Map 对象；
// layer 参数：点中的 Layer 对象。
// 最后两个参数用的较少，不做详细介绍。
// 其中利用 item 参数就可以判断点中的是什么项目，
// 而之后的 map 和 layer 参数按照 item 的结果返回对应类型的接口值。
axTOCControl1.HitTest(e.x, e.y, ref item, ref map, ref layer, ref other,
                      ref index);

// item 参数是一个 esriTOCControlItem 类型变量，
// 该类型是一个枚举型，列举了 TOCControl 中可以显示的多种项目，
// 具体的意义可以查阅 AO 帮助文档，
// 其中 esriTOCControlItemMap 类型表示点中了一个 Map 项目；
// esriTOCControlItemLayer 类型表示点中了一个图层项目
if (item == esriTOCControlItem.esriTOCControlItemMap)
{
    MapMenuStrip.Show(axTOCControl1, e.x, e.y);
    // 如果点中了一个 Map 项目，
    // map 参数将返回伙伴控件（地图主窗口 MapControl）中的
    // 相应 Map 对象（以 IBasicMap 接口类型），
    HitMap = map as IMap;
}
else if (item == esriTOCControlItem.esriTOCControlItemLayer)
{
    LayerMenuStrip.Show(axTOCControl1, e.x, e.y);
    // 如果点中了一个图层项目，
    // layer 参数将返回伙伴控件（地图主窗口 MapControl）中的
    // 相应图层对象（以 ILayer 接口类型），
    HitLayer = layer;
}
}
}

```

### 3. 利用 Geodatabase API 添加 shapefiles 图层。这一部分内容介绍实现添加图层

功能的基本。本内容的实现过程可以分为以下几步：

(1) 添加菜单项的点击事件处理函数。CODE4.1.只能实现右键菜单的显示，但要执行相应的菜单项功能，还需要进行菜单项的点击事件处理（如图 4.14），CODE4.2.表达了选择“添加 Shape 图层”菜单项后显示打开文件对话框进行\*.shp 文件选择的代码，其中事件处理函数名称与实验者自定义的菜单项对象名相关，在实验中可能与 CODE4.2 不同。选择文件之后的数据读取操作将在后面的步骤中详述。

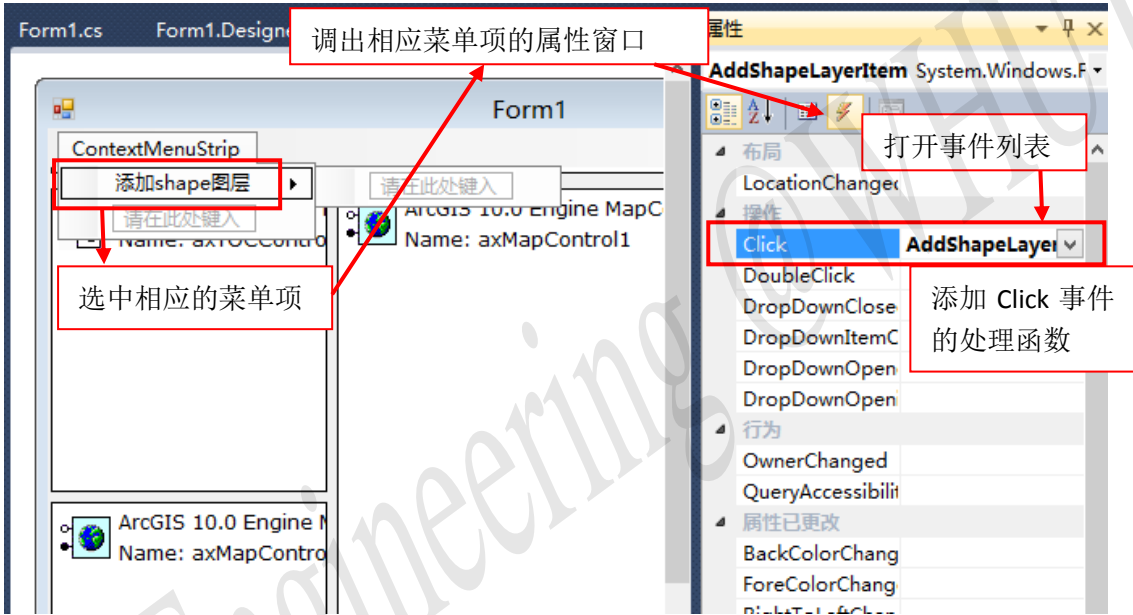


图 4.14. 添加菜单项的点击事件处理

CODE4.2. “添加 Shape 图层”菜单项的事件处理函数

```
private void AddShapeLayerItem_Click(object sender, EventArgs e)
{
    // OpenFileDialog 类用于创建“打开文件”对话框，它是 C#中的标准控件。
    OpenFileDialog OpenShapeDlg = new OpenFileDialog();
    // 设置打开文件的类型过滤规则为：仅打开*.shp 文件。
    OpenShapeDlg.Filter = "shp files (*.shp)|*.shp";
    // 其他参数设置。
    OpenShapeDlg.FilterIndex = 1;
    OpenShapeDlg.RestoreDirectory = true;
    // 显示“打开文件”对话框，
    // 返回值为 DialogResult.OK 表示点击了“确定”按钮。
    if (OpenShapeDlg.ShowDialog() == DialogResult.OK)
```

```
{  
    // 读取 shapefile 数据的代码，参见 CODE4.3  
}  
}
```

(3) 添加必要的程序集引用。本实验将用到的 Geodatabase API 中的各种接口和组件主要定义在 ESRI.ArcGIS.Geodatabase 库中，而与 shapefiles 文件相关的 Workspace 组件定义在 ESRI.ArcGIS.DataSourceFile 库中，这两个库的程序集引用需实验者手动添加至开发项目的“引用”集合中（如图 4.15），并且在 Visual Studio2010 中注意将这两个程序集的“嵌入互操作类型”设置为 False。

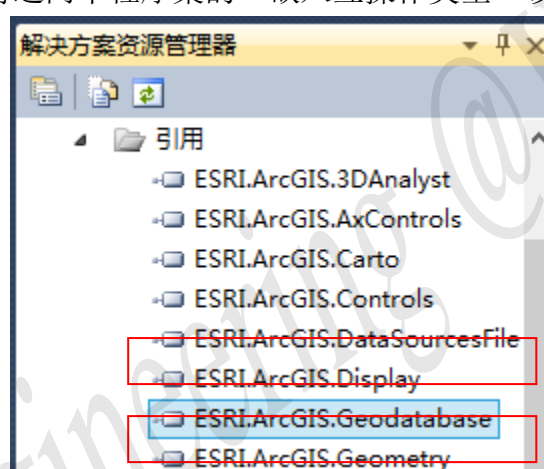


图 4.15. 添加必要的程序集引用

(4) 打开 Workspace 的操作。在 Geodatabase API 中 IWorkspace 接口定义了各项数据操作的入口，通过创建一个带 IWorkspace 接口的组件对象来实现对特定数据库的连接。但是各种 Workspace 组件都不是 CoClass 类型，也就是说不能用 new 操作符来创建对象，为了提高接口与数据库类型的分离性，提高接口方法处理各类型数据存储的能力，Geodatabase API 采用了“类工厂”的模式来创建 Workspace 组件对象，Workspace 组件的类工厂是一种实现了 IWorkspaceFactory 接口的 CoClass 型组件，不同类型的数据存储对应于不同的 WorkspaceFactory 组件，详细信息可以在帮助文档的 IWorkspaceFactory 接口部分查阅。IWorkspaceFactory 接口有两个方法主要用来创建 Workspace 组件对象(如图 4.16)，其中 Open 方法用于需要通过连接属性来打开的各种数据库系统，例如各种远程访问的数据库管理系统（SqlServer，Oracle 等），OpenFromFile 方法用于以本地文件形式存储的数据文件或者数据库文件，例如 Personal GDB 数据库管理系统（Access 等）或者早期的文件数据系统（Shapefile、Coverage 等）。

←	Open	Opens the workspace specified by the connection properties.
←	OpenFromFile	Opens the workspace specified by the given file name.

图 4.16. IWorkspaceFactory 接口中用于创建 Workspace 组件对象的方法

对应于 Shapefile 文件的类工厂是 ShapefileWorkspaceFactory 组件，也就是说要连接 Shapefile 文件的数据库，必须通过利用 ShapefileWorkspaceFactory 组件对象创建 Workspace 组件对象的方式来实现。由于 Shapefile 文件的数据库是以文件形式存储在本地磁盘中的，因此可以利用 IWorkspaceFactory 接口定义的 OpenFromFile 方法来创建一个连接了指定 Shapefile 数据库的 Workspace 组件对象。OpenFromFile 方法的定义如图 4.17 所示，其中 hWnd 表示连接对话框的父窗口句柄，一般可以设置为 0，关键的是 fileName 参数，它表示数据库文件的磁盘存储路径名。严格的说，Shapefile 并不是以 DBMS 的形式组织管理的，不存在单独的数据库文件，它的数据是利用操作系统的本地磁盘管理系统来组织管理的，在 Windows 环境下，就是采用目录的方式，因此一个 Shapefile 文件数据库就对应于一个包含 Shapefile 文件的目录，对于 Shapefile 数据来说，OpenFromFile 方法中要求的数据库文件路径名就是包含指定数据的目录名称。例如，在如下图 4.18 所示的路径下打开名为 emptaz 的 shape 文件，则数据库的名称应该是“D:\myprojects\data\arcgis\commuting”（注意，OpenFromFile 中需要使用绝对路径名，相对路径名是无效的）。

```
[C#]public IWorkspace OpenFromFile (
    string fileName,
    IntPtr hWnd);
```

图 4.17. OpenFromFile 方法的定义

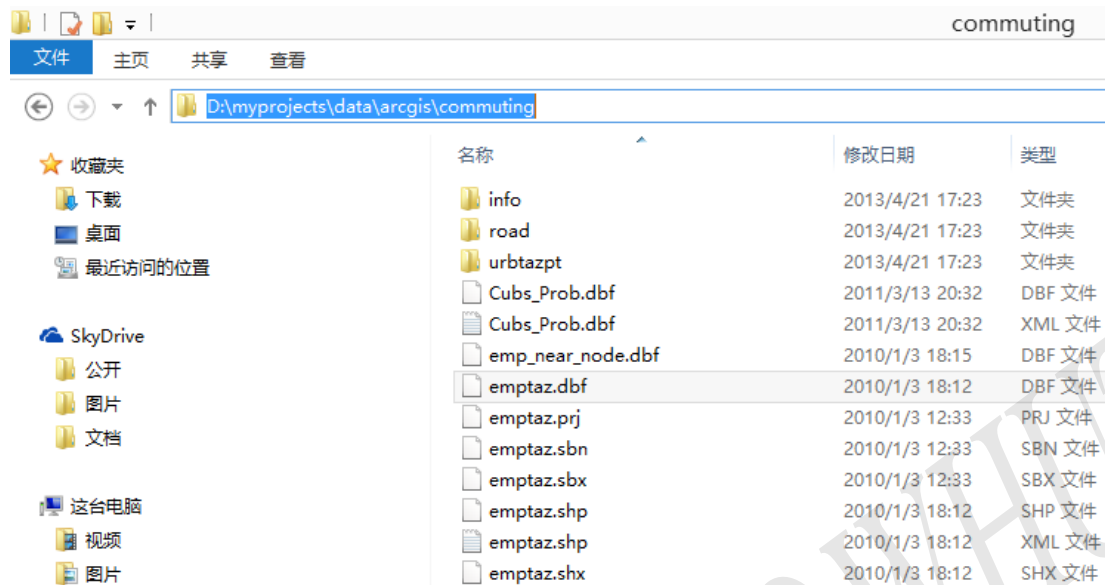


图 4.18. shapefiles 数据的目录结构示例

接上文中的 CODE4.2，利用 OpenFileDialog 对话框可以获取选定文件的完整路径名，利用.NET 中 System.IO 库的 Path 类定义的相关方法可以解析这个完整路径名得到目录名称和文件名称，然后就可以调用 OpenFromFile 方法打开 Shapefile 数据库，这部分代码可以参见 CODE4.3。

#### CODE 4.3 打开 OpenFileDialog 选择的\*.shp 文件所在的 Shapefile 数据库

```
// 变量 OpenShapeDlg 的定义见 CODE4.2.
// OpenFileDialog 类的 FileName 属性存储了选中文件的完整路径名，
// 利用 System.IO 库中 Path 类的 GetDirectoryName 方法可以解析出目录名。
string strWorkspace = System.IO.Path.GetDirectoryName(OpenShapeDlg.FileName);
// 利用 System.IO 库中 Path 类的 GetFileNameWithoutExtension 方法可以解析出
// 不包含扩展名的文件名称（不包含路径名），
// 该文件名用于后续的加载数据操作，可以作为图层名称使用。
string strLayerName =
    System.IO.Path.GetFileNameWithoutExtension(OpenShapeDlg.FileName);
// 实例化一个 ShapefileWorkspaceFactory 组件对象，
// 该组件定义在 ESRI.ArcGIS.DataSourcesFile 组件库中，
// 如要直接使用组件类名进行实例化，请在相应的代码文件中添加 using 操作。
IWorkspaceFactory wsFactory = new ShapefileWorkspaceFactory();
// 调用 OpenFromFile 通过目录名方法打开 Shapefile 数据库。
IWorkspace shpWorkspace = wsFactory.OpenFromFile(strWorkspace, 0);
```

```
If (shpWorkspace != null)
{
    // 此处添加读取数据文件的代码，参见 CODE4.4
}
```

(5) 加载矢量要素图层的操作。以 **Workspace** 的形式打开数据文件目录后，其中的数据文件就可以看作是数据集 (**Dataset**)，在 **Workspace** 中将创建多个数据集组件对象来读取相应的数据文件，根据数据文件的不同类型，这些组件对象可能属于不同的组件类，但都实现了 **IDataset** 接口。**Shapefile** 文件所表达的数据集存储了矢量要素 (**Feature**) 信息，由矢量要素构成的数据集又称为“要素类” (**FeatureClass**)，要素类对应的数据集组件除了实现 **IDataset** 接口外，还实现了 **IFeatureClass** 接口。

原则上，我们希望能根据数据文件的名称获取 **Workspace** 中对应的数据集组件对象，但 **IWorkspace** 接口并没有直接提供这一方法，因为在 **Geodatabase** 数据模型中，**Workspace** 被定义为“数据集的集合的集合”，数据库中的数据集先按照类型被组合为多种“数据集的集合” (**Datasets**)，然后这些个 **Datasets** 再组合为 **Workspace**，从而一个 **Workspace** 中能包含多种类型的数据集。因此，在这一模型下，要获取一个指定的数据集组件对象，需要先获取指定类型的“数据集的集合” (**Datasets**)，这是一个实现了 **IEnumDataset** 接口的组件对象，然后在这个对象中查找指定的数据集组件对象。**IEnumDataset** 接口很简单，仅提供了两个方法：**Next** 和 **Reset**，它可以采用轮询的方式通过集合元素定位器依次获取集合中的每一个元素，其中 **Reset** 方法表示将定位器重置到集合的起始位置，**Next** 方法有两个作用，一是返回定位器当前指向的集合元素，二是同时将定位器后移一位。**AO** 库中有很多命名为 **IEnum\*\*\*** 的接口，都采用了类似的轮询操作模式来获取各种组件对象集合中的元素，在后续实验中会碰到一些其他的例子。

另一方面，矢量要素类组件对象并不能直接作为图层添加到 **Map** 组件对象中，只有实现了 **ILayer** 接口的图层组件对象才能作为 **Map** 的图层，包含矢量要素数据的图层组件对象称为矢量要素图层 (**FeatureLayer**)，它除了实现 **ILayer** 接口外，还实现了 **IFeatureLayer** 接口，该接口定义了一个 **FeatureClass** 属性，通过将这个属性赋值为相应的矢量要素类组件对象，就能将矢量要素图层与指定的矢量要素数据集关联起来。

根据上述说明，可以参考 **CODE4.4** 来将指定的 **Shapefile** 数据文件加载为矢量要素图层。

```
CODE 4.4 将指定的 Shapefile 文件加载为矢量要素图层
// 变量 shpWorkspace 的定义见 CODE4.3.
```



```

// 获取数据集类型为“矢量要素类”的 Datasets。
// IWorkspace 中定义了一个带参数的 Datasets 属性，
// 在 C#中读取该属性的值应转换为函数的形式，使用 get_Datasets 函数，
// 该属性的参数指明了要获取的 Datasets 的类型，
// 这是一个 esriDatasetType 类型的枚举变量，
// 本例中 esriDatasetType.esriDTFeatureClass 表示“矢量要素类”类型。
IEnumerable shpDatasets =
    shpWorkspace.get_Datasets(esriDatasetType.esriDTFeatureClass);
// 获取的 Datasets 是一个实现了 IEnumerable 接口的组件对象，
// 首先利用 Reset 方法将元素定位器重置到起始位置。
shpDataset.Reset();
// 调用 Next 方法获取其中的第一个数据集对象，并将定位器后移一位
IDataset shpDataset = shpDatasets.Next();
// 循环访问 Datasets 中的每一个数据集对象，shpDataset 为 null 表示循环结束。
while (shpDataset != null)
{
    // 变量 strLayerName 的定义见 CODE4.3.
    // 数据集的 Name 属性表示了相应数据文件的名称，
    // 通过名称比较来查找指定数据文件对应的数据集对象。
    if (shpDataset.Name == strLayerName)
    {
        // 首先实例化一个 FeatureLayer 组件对象，作为添加进 Map 的图层。
        IFeatureLayer newLayer = new FeatureLayerClass();
        //将矢量要素图层对象的 FeatureClass 属性设置为对应的数据集对象。
        // 由于 Datasets 是“矢量要素类”类型，
        // 因此其中的所有数据集对象都是矢量要素类数据集，
        // 都能转换为 IFeatureClass 类型。
        newLayer.FeatureClass = shpDataset as IFeatureClass;
        // 将图层的名称设置为数据文件的名称，
        // 这个名称将显示在 TOCControl 对应的图层项中。
        newLayer.Name = strLayerName;
        // 将新建的矢量要素图层添加到地图主窗口中。
        axMapControl1.Map.AddLayer(newLayer);
        // 同时将新建的矢量要素图层添加到鹰眼中，实现数据同步。
        axMapControl2.Map.AddLayer(newLayer);
    }
}

```

```

        break;
    }
    // IEnumDataset 接口的轮询操作模式要求依次获取每一个集合元素，
    // 因此要依次调用 Next 方法。
    shpDataset = shpDatasets.Next();
}

```

4. 读取矢量要素的字段信息。在 Geodatabase 模型中，无论矢量要素数据集的原始存储方式是什么，都会被统一表达为关系型数据表的形式，矢量要素的空间信息和属性信息都表达为数据表的字段，因此我们在“打开属性表”对话框中采用 DataGridView 控件显示矢量要素图层的属性表，表的字段设置通过读取矢量要素类（FeatureClass）的字段信息得到，数据记录的值的读取方法在后续的实验中介绍。这一内容通过以下步骤完成：

（1）添加菜单项的点击事件处理函数。CODE4.5.表达了选择“打开属性表”菜单项后显示“打开属性表”对话框的代码，其中事件处理函数名称与实验者自定义的菜单项对象名相关，在实验中可能与 CODE4.5 不同。对话框中的数据表显示由 LayerAttrib 类的 ShowFeatureLayerAttrib 方法实现（参见图 4.10），实现过程后文详述。

CODE4.5 “打开属性表”菜单项的 Click 事件处理函数

```

private void OpenLayerAttribItem_Click(object sender, EventArgs e)
{
    // 创建“打开属性表”对话框的实例
    LayerAttrib AttribData = new LayerAttrib();
    // 显示对话框
    AttribData.Show();
    // 在对话框的 DataGridView 控件内显示选中图层的属性字段信息
    AttribData.ShowFeatureLayerAttrib(HitLayer as IFeatureLayer);
    // 将保存的选中图层变量置为 null，避免以后的操作中出现混乱
    HitLayer = null;
}

```

（2）读取矢量要素数据集的字段信息。LayerAttrib 类的 ShowFeatureLayerAttrib 方法实现了根据传入的图层参数显示图层中属性字段的功能，其中传入的图层参数定义为 IFeatureLayer 类型，表示仅针对矢量要素图层进行显示。根据 Geodatabase 模型，矢量要素数据集称为“矢量要素类”（FeatureClass），它可看

作一个关系型数据表，其属性由称为“字段”（Field），因此表可以看作是字段的集合（Fields）。在 Geodatabase API 中定义了对字段集合和字段进行操作的接口，分别为 IFields 和 IField。通过 IFeatureClass 接口定义的 Fields 属性（该属性是一个 IFields 类型的组件对象变量），可以获取对应的矢量要素类中的字段集合，然后通过 IFields 接口定义的 Field 属性（该属性是一个通过索引参数返回 IField 类型组件对象的变量）查找字段信息，具体代码可参见 CODE4.6。

CODE4.6 显示矢量要素图层中属性字段。

```
public void ShowFeatureLayerAttrib(IFeatureLayer layer)
{
    if (layer != null)
    {
        // 将目标图层显示在对话框的 MapControl 控件窗口中
        axMapControl1.ClearLayers();
        axMapControl1.AddLayer(layer);
        // 创建一个 C# 的 DataTable 对象用于显示图层要素的属性表
        DataTable FeatureTable = new DataTable(layer.Name);
        // 获取矢量要素图层包含的要素类
        IFeatureClass fcLayer = layer.FeatureClass;
        // 获取要素类中包含的属性字段总数
        int nFieldCount = fcLayer.Fields.FieldCount;
        // 分别按照每一个属性字段信息在 DataTable 对象中创建相应的列
        for (int i = 0; i < nFieldCount; i++)
        {
            // 创建一个新的列（用于添加到 DataTable 对象中）
            DataColumn Field = new DataColumn();
            // 列名设置为属性字段的名称
            // 因为 AO 的 IFields 接口的 Field 属性是一个带参数的集合属性
            // 通过该属性访问某个字段时要采用 get_Field 的函数形式
            // 函数的参数是字段的索引值（从 0 开始）
            Field.ColumnName = fcLayer.Fields.get_Field(i).Name;
            // 根据属性字段值的类型设置 DataTable 对象中相应列的值类型
            // IField 接口定义的 Type 属性表示对应字段的值类型
            // Type 属性是一个 esriFieldType 枚举类型的变量
            // esriFieldType 定义了 Geodatabase 中可以处理的所有值类型
```

```

        switch (fcLayer.Fields.get_Field(i).Type)
        {
            // 利用 switch...case...结构将新列的值类型
            // 与要素类属性字段的值类型一一对应
            case esriFieldType.esriFieldTypeOID:
Field.DataType = System.Type.GetType("System.Int32");
                break;
            case esriFieldType.esriFieldTypeGeometry:
                Field.DataType = System.Type.GetType("System.String");
                break;
            case esriFieldType.esriFieldTypeInteger:
                Field.DataType = System.Type.GetType("System.Int32");
                break;
            case esriFieldType.esriFieldTypeSingle:
                Field.DataType = System.Type.GetType("System.Int32");
                break;
            case esriFieldType.esriFieldTypeSmallInteger:
Field.DataType = System.Type.GetType("System.Int32");
                break;
            case esriFieldType.esriFieldTypeString:
                Field.DataType = System.Type.GetType("System.String");
                break;
            case esriFieldType.esriFieldTypeDouble:
                Field.DataType = System.Type.GetType("System.Double");
                break;
        }
        // 将创建的新列加入到 DataTable 对象中
FeatureTable.Columns.Add(Field);
    }

    // 将 DataTable 对象作为 DataGridView 控件的数据源
    dataGridView1.DataSource = FeatureTable;
    // ShowFeatures 函数在 DataGridView 控件中显示所有要素的属性数据
    // 该函数的实现方法在后续实验中进行介绍，参见 CODE5.1.
    //ShowFeatures(layer, null, false);
}

```

```
// 如果传入的 layer 参数为 null，则清空 DataGridView 控件中的所有数据
else
{
    dataGridView1.Columns.Clear();
}
}
```

## 六、思考题

1. 请在图层右键菜单中添加一个“删除图层”的菜单项，并实现相应的功能。
2. 请解释 `esriFieldType` 中定义的每一种值类型的含义。
3. CODE4.4 中利用在鹰眼窗口中添加相同的图层的方式实现与主窗口的数据同步，在前面的实验中我们采用了 `ItemAdded` 事件处理的方案实现添加图层时的数据同步，请测试一下，在本例中能否在该事件处理函数中进行地图同步？如不能，是什么原因？如果要同步过程统一放在 `ItemAdded` 事件处理函数中，应该如何修改实验项目？

## 实验五. 矢量要素的查询和编辑

### 一、实验目的

认识 Geodatabase 数据模型下矢量要素空间和属性查询的统一模式，掌握基于数据表结构的矢量要素属性值读取和修改操作。

### 二、实验仪器

常规配置微机，Windows 操作系统，Visual Studio2005 及以上版本，ArcGIS Engine9.2 及以上版本。本指导书将以 Visual Studio 2010 和 ArcGIS Engine10.0 为基本开发环境，在其他版本的 Visual Studio 和 ArcGIS Engine 中进行开发的过程仅有细微差别，读者可自行查找相关技术资料进行详细了解和处理。

### 三、实验要求

了解 Geodatabase 数据模型的基本结构，掌握 Geodatabase API 中与矢量要素相关的组件和接口的基本调用方法。

### 四、实验内容

本次实验主要围绕“打开属性表”对话框进行操作，实验完成后将实现的功能如下：

1. 选择矢量要素图层并弹出“打开属性表”对话框后，DataGridView 控件中将显示所有要素的所有字段信息（如图 5.1）。

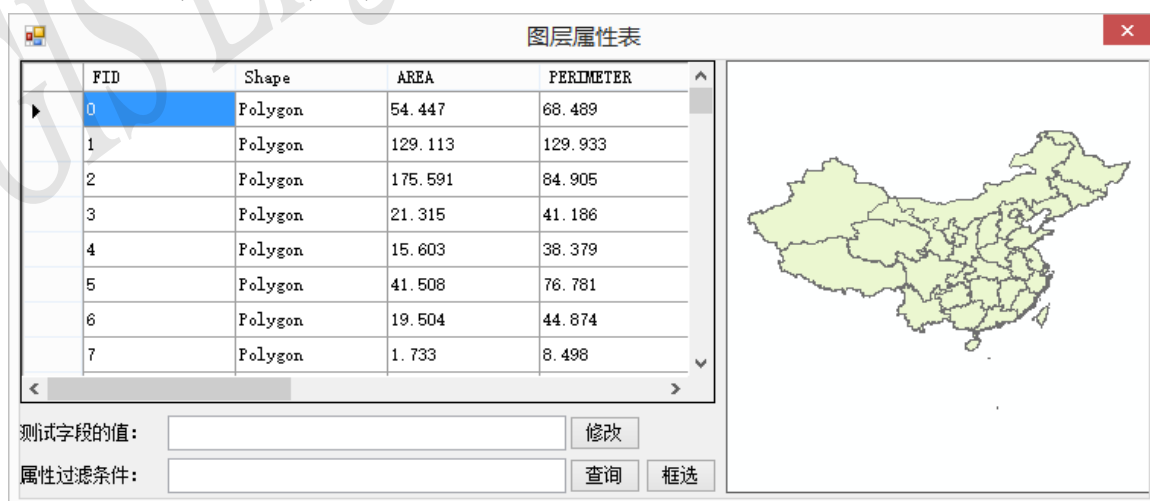


图 5.1. 矢量要素信息的完整显示

2. 在“打开属性表”对话框中添加属性值查询修改功能。对话框中“属性过滤条件”后的文本框中可以采用 SQL 语言的语法输入属性过滤条件（如果不输入任

何信息，则表示全选所有要素)，点击“查询”按钮后 DataGridView 控件中显示符合查询条件的矢量要素信息，同时右边的 MapControl 控件中将高亮显示这些矢量要素（如图 5.2）。

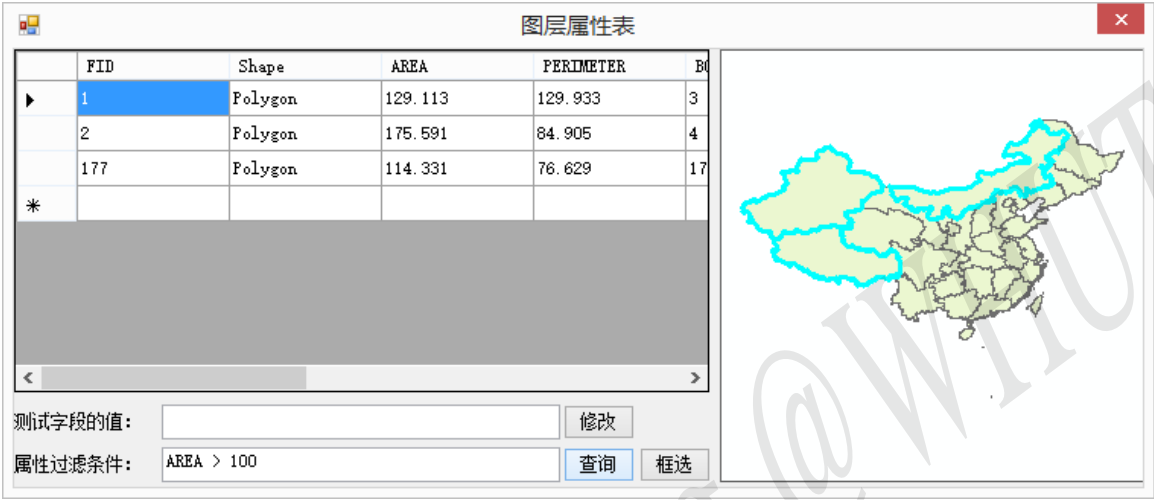


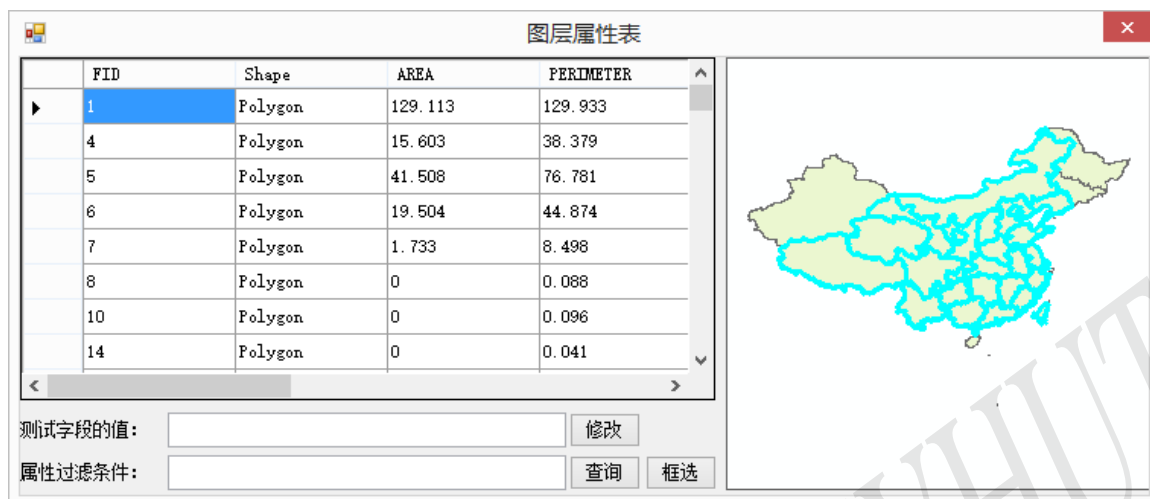
图 5.2. 属性查询功能

3. 在“打开属性表”对话框中添加空间查询功能。点击“框选”按钮后，能在 MapControl 控件窗口中执行一次鼠标拖出矩形框进行矢量要素空间查询的功能，MapControl 控件窗口中将高亮显示与鼠标绘制的矩形框相交的所有矢量要素，同时在 DataGridView 控件中也将显示这些要素的属性信息(如图 5.3-5.4)。另外，在空间查询过程中，“属性过滤条件”文本框中的设置也是有效的，也就是说可以执行“属性+空间”的联合查询。



图 5.3. 鼠标拖出矩形框进行空间查询





5.4. 空间查询的结果显示

4. 在“打开属性表”对话框中实现添加测试字段并修改属性值的功能。在 **DataGridView** 控件中点击鼠标右键能弹出具有“添加测试字段”项和“删除测试字段”项的菜单，选择“添加测试字段”项能在目标图层属性表的最后添加一个名为“StringTest”的字符串类型测试字段，用于进行属性值修改的实验，选择“删除测试字段”项能将“StringTest”字段删除（如图 5.5-5.7）。在“属性过滤条件”文本框中填写相应的过滤设置，然后在“测试字段的值”文本框中填写字段值，再点击“修改”按钮就能将符合过滤条件的矢量要素的“StringTest”字段设置为相应的值，同时符合条件的矢量要素会在 **DataGridView** 和 **MapControl** 中进行相应的显示（如图 5.8）。

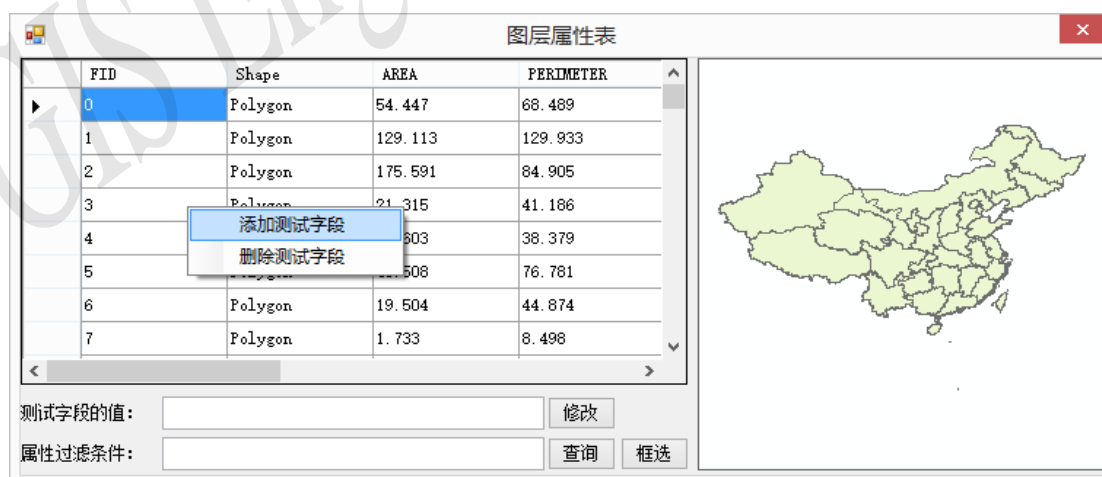


图 5.5. 字段操作的右键菜单

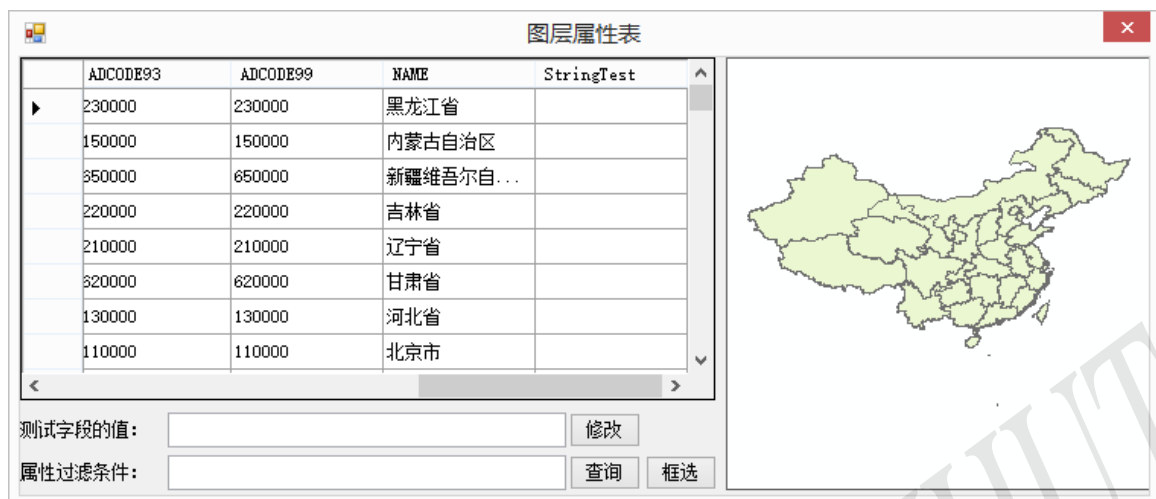


图 5.6. 添加测试字段的的结果

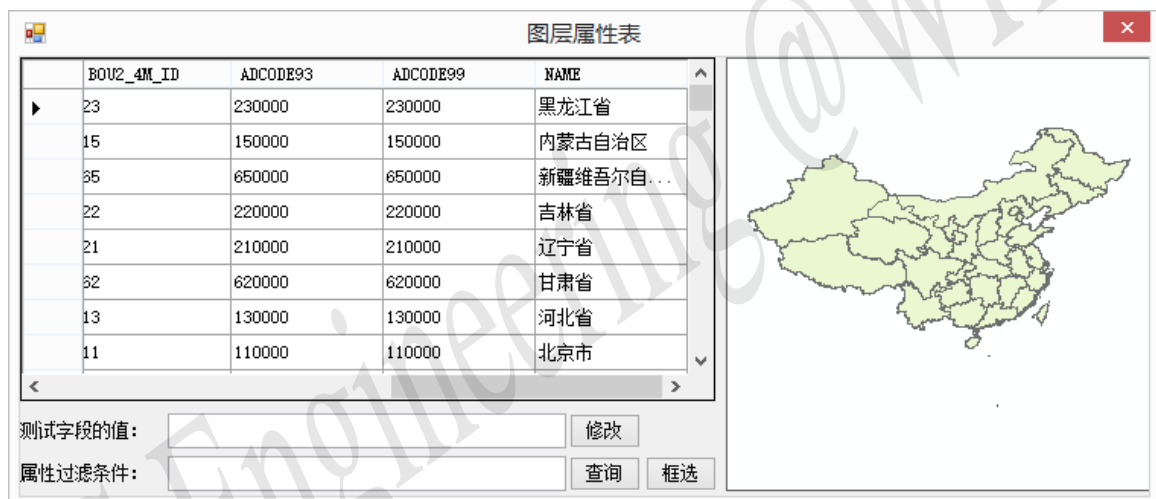


图 5.7. 删除测试字段的的结果



图 5.8. 修改测试字段属性值的的结果

五、实验步骤

1. 在“窗体设计器”中进行“图层属性表”对话框的控件布局设置，这部分内容为常规操作，此处略去。为了后文描述代码的需要，本文案例中各主要控件的对象名称（即控件的 Name 属性）按图 5.9 标注所示定义，读者可以按照自己的喜好定义不同的对象名称，但要注意在参考后文代码示例时也进行相应的修改。

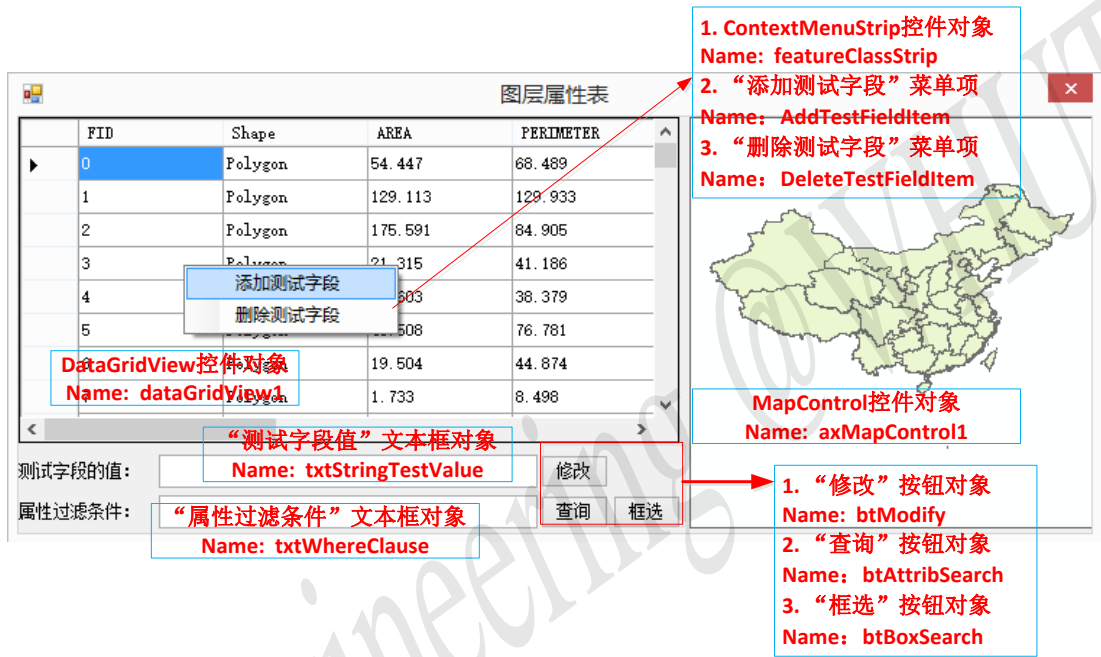


图 5.9. 主要控件的对象名称设置

2. 获取所有要素的属性信息并在 DataGridView 控件中显示。在实验 4 的 CODE4.6 中描述了显示要素全部属性字段的方法，在该方法的注释中说明了 ShowFeatures 方法用于显示所有要素的属性值，本次实验将介绍 ShowFeatures 方法的实现过程。本案例中的 ShowFeatures 方法是一个 LayerAttrib 窗体中的自定义方法，它的主要功能是在 DataGridView 控件和 MapControl 控件中显示按照指定查询条件搜索得到的所有矢量要素的属性和空间信息，其参数定义如表 5.1 所示

表 5.1 ShowFeatures 方法的参数定义

参数	意义
IFeatureLayer featurelayer	属性操作的目标图层
IQueryFilter condition	指定的查询条件
bool drawshape	指示是否需要在 MapControl 窗口中高亮显示矢量要素

ShowFeatures 方法在本次实验中是一个核心方法，调用时将 condition 参数设置为 null 就表示获取全部要素，而将 condition 参数设置为其他有意义的查询条

件就能实现按属性查询要素。因此本次实验案例的基本思路是统一利用要素查询过程实现在矢量要素图层中获取要素的功能。

3. 根据指定的属性查询条件查询矢量要素并高亮显示的方法。由于 Geodatabase 模型采用了统一的关系型数据模型框架，因此要素查询可以看作在数据表中查询数据的过程。GIS 中常用的查询模式是根据属性值查询（即“属性查询”）或者根据空间关系查询（即“空间查询”），AO 组件库提供了 IQueryFilter 接口用来设置属性查询条件，并从该接口派生了一个 ISpatialFilter 接口用来设置空间查询条件。查询条件的设置方法在后文中详细讲解，此处主要介绍 ShowFeatures 方法的实现过程，在该方法中查询条件是一个 IQueryFilter 类型的传入参数，我们将学习在给定查询条件的情况下，如何获取查询结果并正确显示。

在关系型数据模型概念下，在数据表中执行一次查询操作将得到一个查询结果集，其中可能包含 0 条或多条数据记录，然后通过“游标”来逐个访问结果集中的各条记录。IFeatureCursor 接口定义了 Search 方法用于根据指定查询条件搜索矢量要素，该方法的定义如图 5.10 所示，其中 filter 参数表示查询条件，Recycling 参数表示了结果集的存储方式，如果需要对结果集中返回的矢量要素进行更进一步的修改，应将该参数设置为 false，如果仅需逐条读取属性值，可以将其设置为 true 以提高效率。在数据量不大的常规环境中建议设置为 false 以避免出现某些特殊的问题。

```
[C#]public IFeatureCursor Search (
    IQueryFilter filter,
    bool Recycling);
```

图 5.10. Search 方法的定义

Search 方法的返回值是一个 IFeatureCursor 类型的变量，实现了 IFeatureCursor 接口的 AO 组件对象表示矢量要素的“游标”，通过该接口定义的方法（如图 5.11）可以实现矢量要素查询结果集中各条记录（矢量要素）的访问和修改等操作。

Members

	All ▾	Description
←	DeleteFeature	Delete the existing Feature in the database corresponding to the current position of the cursor.
■	Fields	The fields Collection for this cursor.
←	FindField	The index of the field with the specified name.
←	Flush	Flush any outstanding buffered writes to the database.
←	InsertFeature	Insert a new Feature into the database using the property values in the input buffer. The ID of the new Feature is returned.
←	NextFeature	Advance the position of the cursor by one and return the Feature object at that position.
←	UpdateFeature	Update the existing Feature in the database corresponding to the current position of the cursor.

图 5.11. IFeatureCursor 接口的成员列表

其中 `NextFeature` 是获取查询结果的主要方法，它的作用是返回游标当前指向的矢量要素，并将游标前移一位，返回 `null` 值表示游标执行结果集的介绍位。需要注意的是 `IFeatureCursor` 对象是一种单向游标，即它用于从结果集开始位置依次获取矢量要素，但游标位置不可回退，也不可重置为开始位置（该接口没有定义 `Reset` 方法），因此如果要多次访问结果集，需要先将要素读出并保存到其他数组中，再多次访问该数组，而不可多次访问 `IFeatureCursor` 对象。

在 `Geodatabase` 模型中，一个矢量要素就是一条包含了空间信息和属性信息的记录。对矢量要素的相关操作方法主要定义在 `IFeature` 接口中（接口成员列表如图 5.12），其中 `Value` 属性用来读取和设置相应字段的值，调用该属性需要以字段的索引号作为参数，因此在 `C#` 中要转化为 `get_Value` 或者 `set_Value` 的函数形式。从图 5.1 中可以看出，属性表的前两个字段是 `FID` 和 `Shape`，这并不是本例所特有的，它们是 `shapefile` 型矢量要素属性表的通用成员，其中 `FID` 存储了矢量要素的编号，对应于 `Geodatabase` 模型中的 `OID`，`Shape` 存储了矢量要素的几何形状信息，这两个通用成员除了通过 `IFeature` 接口的 `Value` 属性访问外，还可以通过 `OID` 和 `Shape` 两个属性访问。

**Members**













	All	Description
	Class	The Object Class for the row.
	Delete	Deletes the row.
	Extent	The extent of the feature.
	FeatureType	The type of the feature.
	Fields	The fields Collection for this row buffer.
	HasOID	Indicates if the row has an OID.
	OID	The OID for the row.
	Shape	A reference to the default shape for the feature.
	ShapeCopy	A cloned copy of the default shape for the feature.
	Store	Stores the row.
	Table	The Table for the row.
	Value	The value of the field with the specified index.

图 5.12. IFeature 接口的成员列表

在 `C#` 中利用 `get_Value` 方法获取的矢量要素属性值是以 `object` 类型的变量返回的，要在 `DataGridView` 中进行显示必须转换为相应的实际数据类型，利用 `IFeature` 接口的 `Fields` 属性能依次获取每个字段的值类型信息，然后根据值类型

进行相应的转换，转换过程可以参考 CODE4.6 中的相应环节。此处需要特别注意的是 Shape 属性，由于它表示了几何图形信息，在非 SDE 的数据存储模型中它不能用标准的关系型数据表结构来存储，因此往往将对应的几何体组件以二进制数据的形式直接存储在属性表的相应字段中，或者以其他形式存储在专门的空间数据文件中（如 shapefile 文件格式）。鉴于这种多样化的存储方式，为了提高程序代码的可移植性，建议实验者通过 IFeature 接口的 Shape 属性来访问空间数据，而非 get\_Value 方法。

“高亮”显示指的是在地图背景上采用特殊效果对目标几何体进行鲜明绘制的过程，原则上我们可以采用后续实验中将介绍的渲染方法来自由设置高亮显示的效果，但在一般情况下通常采用 Map 组件提供的一种标准效果（如图 5.2）。矢量要素图层对象中包含一个要素选择集，Map 组件能自动对选择集中的要素进行标准效果的高亮显示。在矢量要素图层组件所实现的众多接口中，包含一个 IFeatureSelection 接口，该接口定义了操作要素选择集的各种方法和属性（如图 5.13 所示）。通过 Add 方法将本图层中需要高亮显示的矢量要素加入到选择集中，就能实现自动高亮显示。

另一方面，Map 组件高亮显示选择集的过程是在地图重绘的步骤中完成的，如果需要立即显示高亮效果，可以通过 MapControl 控件的 Refresh 方法实时启动地图重绘。但要正确显示选择集的结果，Map 组件要求执行进行两次地图刷新操作，一次在选择集数据变化之前，另一次在选择集变化之后。

Members		
	All ▾	Description
←	Add	Adds a feature to the selection set.
■-■	BufferDistance	Buffer distance used for the selection.
←	Clear	Clears the selection.
■-■	CombinationMethod	Combination method for the selection.
←	SelectFeatures	Selects features based upon the specied criteria and combination method.
←	SelectionChanged	Fires the features layer update event. Required when SelectionSet changes.
■-□	SelectionColor	Selection color. (used when SetSelectionSymbol = FALSE).
■-□	SelectionSet	The selected set of features.
■-□	SelectionSymbol	Selection symbol.
■-■	SetSelectionSymbol	Indicates if the selected set of features is drawn using the SelectionSymbol.

图 5.13. IFeatureSelection 接口的成员列表

ShowFeatures 方法的完整代码可以参考 CODE5.1，它提供了在属性表中显示选择结果集的属性值并在地图窗口高亮显示空间形态的功能。

CODE5.1 ShowFeatures 方法
<pre>public void ShowFeatures(IFeatureLayer featurelayer, IQueryFilter condition, bool</pre>



```

drawshape)
{
    if (featurelayer != null)
    {
        // 获取矢量要素图层关联的要素类
        IFeatureClass featureclass = featurelayer.FeatureClass;
        //定义一个 IFeatureCursor 变量用于获取查询结果
        IFeatureCursor cursor = null;
        // 此处使用的 try...catch...语法用于捕获异常
        // 异常处理常用的运行错误处理模式
        // 在属性查询中如果查询条件语句不规范，Search 方法会产生异常
        try
        {
            cursor = featureclass.Search(condition, false);
        }
        catch (Exception)
        {
            // 如果 Search 方法产生了异常，将查询结果集游标置为 null
            cursor = null;
        }
        if (cursor != null)
        {
            // 第一次地图重绘请求
            // 注意 Refresh 的第一个参数
            axMapControl1.Refresh(esriViewDrawPhase.esriViewGeoSelection,
                                   Type.Missing, Type.Missing);
            IFeatureSelection fselection = featurelayer as IFeatureSelection;
            //清空原先的选择集
            fselection.Clear();
            // 获取 DataGridView 控件中的数据表
            DataTable FeatureTable = (DataTable)dataGridView1.DataSource;
            // 清空表中原有的行
            FeatureTable.Rows.Clear();
            // 通过游标获取要素查询结果集中的第一个要素
            IFeature feature = cursor.NextFeature();

```



```

// 循环获取每一个要素，直到游标到达结果集的结束位
while (feature != null)
{
    // 如果需要高亮显示查询结果，将查询到的要素加入选择集
    if (drawshape)
    fselection.Add(feature);
    // 创建一个新的行在 DataGridView 的数据表中添加数据
    DataRow FeatureRec = FeatureTable.NewRow();
    int nFieldCount = feature.Fields.FieldCount;
    //依次读取矢量要素的每一个字段值
    for (int i = 0; i < nFieldCount; i++)
    {
        // 下列的 switch...case...结构用于根据字段值类型
        // 转换矢量要素的各个属性值
        // 并将值写入新建数据行的对应列中。
        // 以下代码仅列出了常用的和特殊的几种值类型，
        // 其他值类型实验者可参考本案例自行扩充
        switch (feature.Fields.get_Field(i).Type)
        {
            //注意：第一个 FID 字段的值类型为 OID 类型
            case esriFieldType.esriFieldTypeOID:
                FeatureRec[feature.Fields.get_Field(i).Name] =
                    Convert.ToInt32(feature.get_Value(i));
                break;
            //注意：Shape 字段的值类型为 Geometry 类型，
            // 但应通过 Shape 属性来获取其值。
            // 但 Shape 属性是一个 AO 组件对象，
            // 使用 C#的 ToString 方法不会得到相应的几何体类型名，
            // 此处使用了一个 GeometryTypeToString 函数，
            // 来生成几何体类型的字符串，
            // 该函数的实现方法非常简单，本书中不给出具体代码
            case esriFieldType.esriFieldTypeGeometry:
                FeatureRec[feature.Fields.get_Field(i).Name] =
                    GeometryTypeToString(feature.Shape.GeometryType);
                break;

```

```

        case esriFieldType.esriFieldTypeInteger:
FeatureRec[feature.Fields.get_Field(i).Name] =
                                Convert.ToInt32(feature.get_Value(i));

                                break;

        case esriFieldType.esriFieldTypeString:
FeatureRec[feature.Fields.get_Field(i).Name] =
                                feature.get_Value(i).ToString();

                                break;

        case esriFieldType.esriFieldTypeDouble:
        FeatureRec[feature.Fields.get_Field(i).Name] =
                                Convert.ToDouble(feature.get_Value(i));

                                break;

        }

    }

    //将填充了数据的新行加入到 DataGridView 的数据表中
    FeatureTable.Rows.Add(FeatureRec);
    // 获取下一个要素
    feature = cursor.NextFeature();
}
//第二次调用地图重绘，执行高亮显示。
// 注意 Refresh 函数的第一个参数。
if (drawshape)
    axMapControl1.Refresh(esriViewDrawPhase.esriViewGeoSelection,
                            Type.Missing, Type.Missing);
}
}
}

```

4. 设置属性查询条件。在 AO 环境下，任何符合 Geodatabase 规范的数据集都可以利用 IWorkspace 接口中提供的 ExecuteSQL 方法执行 SQL 语句，而数据查询对应于 select 语句。例如在如图 5.1 所示的表格中查询 AREA 字段值大于 100 的记录，用 SQL 语句可以表示如下：

Select \* from ... where AREA > 100;

其中\*表示返回的结果集包含所有字段，如果仅包含一部分字段，可以用字段名称列表代替符号\*，from 关键字后面是指定数据表的名称，而查询条件是通

过 where 关键字后的内容来表达的,这部分称为“where 字句”或者“where clause”。

但是,在矢量要素类中通常使用 IFeatureClass 接口的 Search 方法进行查询,该方法需要利用 QueryFilter 组件来表达查询条件(如图 5.10),该组件所实现的主要接口是 IQueryFilter,该接口定义了属性查询条件的设置方法(如图 5.14)。

Members		
	<div>A11 ▾</div>	Description
←	AddField	Appends a single field name to the list of sub-fields.
▢	OutputSpatialReference	The spatial reference in which to output geometry for a given field.
▢	SubFields	The comma delimited list of field names for the filter.
▢	WhereClause	The where clause for the filter.

图 5.14. IQueryFilter 接口的成员列表

其中 AddField 方法对应于 select 语句的结果集字段列表部分,利用该方法可以逐个添加需要在结果集中返回的字段,并可以通过 SubFields 属性查看字段列表的设置结果。如果创建一个 QueryFilter 组件对象后不进行任何 AddField 调用,或者将 SubField 属性清空,则表示“select \*”。该接口中最重要的属性是 WhereClause,它表达了属性查询条件,对应于 SQL 的 where 字句中除去“where”关键字的部分,在如图 5.1 所示的表格中查询 AREA 字段值大于 100 的记录时,WhereClause 的值应设置为“AREA > 100”。在本实验案例中,我们在对话框中添加了一个“属性过滤条件”文本框用于填写 WhereClause,点击“查询”按钮后,能根据用户填写的条件执行属性查询,结果集中将保留所有字段。

添加“查询”按钮的点击事件处理函数,并完成属性查询代码,可参见 CODE5.2。

CODE5.2 “查询”按钮的点击事件处理函数

```
private void btAttribSearch_Click(object sender, EventArgs e)
{
    // 在本例中,“图层属性表”对话框的 MapControl 控件中只显示一个图层
    // 该图层就是用于数据操作的目标图层。
    IFeatureLayer fLayer = axMapControl1.get_Layer(0) as IFeatureLayer;
    if (fLayer != null)
    {
        //创建一个 QueryFilter 组件对象
        IQueryFilter filter = new QueryFilterClass();
        // 设置 WhereClause
        filter.WhereClause = txtWhereClause.Text;
        // 调用 ShowFeatures 方法进行查询和显示。
        // 需要注意的是,用户自己填写的查询条件可能不符合 SQL 规范,
```

```
// 这会引起 Search 方法产生异常，
// 因此在 ShowFeatures 方法中使用的 try...catch...语法进行异常处理。
ShowFeatures(fLayer, filter, true);

}

}
```

5. 设置空间查询条件。在矢量要素类中采用 **Search** 方法进行查询的一个好处是，它提供了一种统一方案来处理属性查询和空间查询操作。空间查询是 GIS 区别于其他数据库管理系统的主要特征之一，它提供根据空间拓扑关系进行数据查询的功能。AO 库提供了 **SpatialFilter** 组件用于表达空间查询条件，该组件的主要接口是 **ISpatialFilter**（如图 5.15），而该接口继承于 **IQueryFilter**，因此它同时提供了属性查询条件和空间查询条件的设置功能，并能实现“属性+空间”的联合查询。

Members

	All	Description
←	AddField	Appends a single field name to the list of sub-fields.
■	FilterOwnsGeometry	Indicates whether the filter owns the query geometry.
■□	Geometry	The query geometry used to filter results.
—□	GeometryEx	The query geometry used to filter results.
■	GeometryField	The name of the Geometry field to which the filter applies.
■□	OutputSpatialReference	The spatial reference in which to output geometry for a given field.
■	SearchOrder	The search order used by the filter.
■	SpatialRel	The spatial relationship checked by the filter.
■	SpatialRelDescription	The array elements which describe the spatial relation between the query geometry and the requested geometries. There are 9 chars in this string which can be either 'F', 'T' or '*'; e.g., TT*FFT*** represents CONTAIN.
■	SubFields	The comma delimited list of field names for the filter.
■	WhereClause	The where clause for the filter.

图 5.15. ISpatialFilter 接口的成员列表

与 **IQueryFilter** 相比，该接口主要通过一些特殊的属性来表达空间关系的相关内容，如表 5.2 所示。

表 5.2 ISpatialFilter 接口的特殊方法和属性

方法或属性名称	说明
Geometry	最重要的属性之一，用于设置进行空间查询的参考几何体组件对象，需要注意的是在 AO 中只有实现了 IRelationalOperator 接口的几何体组件对象才能作为空间查询的参考几何体。
SpatialRel	最重要的属性之一，用于设置空间关系

	条件,它是一个 <code>esriSpatialRelEnum</code> 类型的枚举变量,定义了 9 种空间关系条件可供选择 (如图 5.16)。
GeometryField	用于设置要素类中存储空间信息的字段名称,默认是 “Shape”。
SearchOrder	用于设置联合查询的过滤顺序,该属性仅对 ArcSDE 型数据库有意义。
SpatialRelDescription	利用该属性可以进行多种空间关系的组合设置,其规范比较复杂,也不常用,此处不做详细介绍。

esriSpatialRelEnum Constants		
Queryable Spatial Relationships.		
Constant	Value	Description
<code>esriSpatialRelUndefined</code>	0	No Defined Spatial Relationship.
<code>esriSpatialRelIntersects</code>	1	Query Geometry Intersects Target Geometry.
<code>esriSpatialRelEnvelopeIntersects</code>	2	Envelope of Query Geometry Intersects Envelope of Target Geometry.
<code>esriSpatialRelIndexIntersects</code>	3	Query Geometry Intersects Index entry for Target Geometry (Primary Index Filter).
<code>esriSpatialRelTouches</code>	4	Query Geometry Touches Target Geometry.
<code>esriSpatialRelOverlaps</code>	5	Query Geometry Overlaps Target Geometry.
<code>esriSpatialRelCrosses</code>	6	Query Geometry Crosses Target Geometry.
<code>esriSpatialRelWithin</code>	7	Query Geometry is Within Target Geometry.
<code>esriSpatialRelContains</code>	8	Query Geometry Contains Target Geometry.
<code>esriSpatialRelRelation</code>	9	Query geometry IBE(Interior-Boundary-Exterior) relationship with target geometry.

图 5.16. 可供空间查询的 9 中空间关系

点击 “框选” 按钮后可以执行一次矩形框空间查询,但由于这涉及到鼠标在 MapControl 控件中的操作,因此完成空间查询需要用到两次事件处理过程,先是处理 “框选” 按钮的点击事件,设置程序进入 “框选” 状态 (参见 CODE5.3),然后处理 MapControl 控件的鼠标点击事件进入框选操作 (参见 CODE5.4)。

CODE5.3 “框选” 按钮的点击事件函数
<pre>private void btBoxSearch_Click(object sender, EventArgs e) {</pre>

```

// 该函数非常简单，仅标志进入“框选”状态。
// nSpatialSearchMode 是在 LayerAttrib 窗体类中自定义的整型成员变量，
// 自定义“= 1”表示“框选”状态
nSpatialSearchMode = 1;
}

```

#### CODE5.4 MapControl 控件的 OnMouseDown 事件处理函数

```

private void axMapControl1_OnMouseDown(object sender,
    ESRI.ArcGIS.Controls.IMapControlEvents2_OnMouseDownEvent e)
{
    // 为了程序的拓展性，我们选择 switch...case...结构判断操作模式
    switch (nSpatialSearchMode)
    {
    case 1: // 矩形框选模式
        // 通过鼠标拖动创建矩形框
        IEnvelope box = axMapControl1.TrackRectangle();
        // 获取空间查询的目标图层
        IFeatureLayer fLayer = axMapControl1.get_Layer(0) as IFeatureLayer;
        if (fLayer != null)
        {
            // 创建一个 SpatialFilter 组件对象
            ISpatialFilter filter = new SpatialFilterClass();
            // 设置属性过滤条件
            filter.WhereClause = txtWhereClause.Text;
            // 设置空间查询的参考几何体为鼠标拖出的矩形框
            filter.Geometry = box;
            // 设置空间关系（本例中是相交关系）
            filter.SpatialRel = esriSpatialRelEnum.esriSpatialRelIntersects;
            // 利用 ShowFeatures 方法查询并显示矢量要素
            ShowFeatures(fLayer, filter, true);
        }
        // 退出“框选”模式
        nSpatialSearchMode = -1;
        break;
    }
}

```

```
}
```

6. 在要素类中添加和删除字段。通过 `IFeatureClass` 接口的 `AddField` 和 `DeleteField` 方法可以实现要素类中字段的添加和删除,但是这两个操作设计到对数据表结构的修改,为了避免操作错误,需要在修改过程中阻止并发的其他数据访问操作,这可以通过数据集的加锁操作实现。矢量要素类组件对象实现了 `ISchemaLock` 接口,利用该接口的 `ChangeSchemaLock` 方法能对数据集进行加锁和解锁操作。删除字段的操作相对简单,可参见 **CODE5.5**。

**CODE5.5** 删除字段,假设字段名称用 `strFieldName` 变量表示,目标要素类用 `fclass` 变量表示

```
// 先查找是否存在指定名称的字段
int nField = fclass.FindField(strFieldName);
if (nField != -1)
{
    // 利用 ISchemaLock 接口进行要素类的加锁操作
    ISchemaLock slock = fclass as ISchemaLock;
    slock.ChangeSchemaLock(esriSchemaLock.esriExclusiveSchemaLock);
    // 获取要删除的字段
    IField delfield= fclass.Fields.get_Field(nField);
    // 删除该字段
    fclass.DeleteField(delfield);
    // 要素类的解锁操作
    slock.ChangeSchemaLock(esriSchemaLock.esriSharedSchemaLock);
}
```

而添加字段的操作相对复杂一些,因为其中涉及到字段对象的创建和设置工作, `AO` 中提供了 `Field` 组件表示 `Geodatabase` 数据集中的字段,它实现了 `IField` 接口,我们在实验中多次使用该接口读取字段属性,但该接口不能用于设置字段属性。新创建字段组件对象后,应使用 `IFieldEdit` 接口来设置其各项属性,并且在 `C#` 中进行设置时,属性名称后应添加 “\_2” 的后缀,如 **CODE5.6** 所示。

**CODE5.6** 添加字段,假设目标要素类用 `fclass` 变量表述,添加字段的名称用 `strFieldName` 表示,字段的值类型为字符串类型

```
// 首先查找是否存在同名字段
int nField = fclass.FindField(strFieldName);
// 如果不存在同名字段才能添加
```

```

if (nField == -1)
{
    // 利用 ISchemaLock 接口进行要素类的加锁操作
    ISchemaLock slock = fclass as ISchemaLock;
    slock.ChangeSchemaLock(esriSchemaLock.esriExclusiveSchemaLock);
    // 创建一个 Field 组件对象
    IField stringfield = new FieldClass();
    // 获取其 IFieldEdit 接口
    IFieldEdit stringfieldedit = stringfield as IFieldEdit;
    // 设置相应的字段属性，请注意属性名后面应添加“_2”后缀。
    // 注意，在 shapefile 格式下，字段名不能超过 10 个字符。
    stringfieldedit.Name_2 = strFieldName; // 字段名
    stringfieldedit.AliasName_2 = strFieldName; // 字段别名
    stringfieldedit.Type_2 = esriFieldType.esriFieldTypeString; // 字段类型
    // 如还需设置其他属性，请模仿上述代码。
    // 添加字段
    fclass.AddField(stringfield);
    // 要素类的解锁操作
    slock.ChangeSchemaLock(esriSchemaLock.esriSharedSchemaLock);
}

```

根据 CODE5.5-5.6，请实验者自行实现图 5.5-5.7 所示右键菜单的功能。

7. 修改矢量要素的属性值。在 SQL 语言中，修改数据可使用 Update，而该语句也利用 where 字句进行记录过滤，因此修改数据的操作可以看作“数据查询+数据更改”的组合操作。IFeatureClass 接口提供了 Update 方法用于矢量要素类的数据修改操作，该方法的与 Search 方法具有完全一致的参数结构和返回值类型，实际上它仅完成了“数据查询”部分，只不过它所返回的游标指向的结果集是可更改数据的。要完成完整的数据修改操作，还需要通过游标获取每一条矢量要素，并设置需要更改的 Value 属性值。在本实验案例中，我们添加了一个“StringTest”字段用于测试修改属性值的操作，修改时的过滤条件仍然来自“属性过滤条件”文本框，而“StringTest”的值来自“测试字段的值”文本框，图 5.8 功能的实现可参见 CODE5.7。

CODE5.7 修改测试字段“StringTest”的属性值，假设目标要素类用 fclass 变量表示。

```

// 首先查找是否存在该字段，并获取该字段的索引值

```



```

int nField = fclass.FindField("StringTest");
if (nField != -1)
{
    IFeatureCursor updatecursor = null;
    // 设置过滤条件
    IQueryFilter updatefilter = new QueryFilterClass();
    updatefilter.WhereClause = txtWhereClause.Text;
    // 采用 try...catch...语句捕获数据过滤时的异常
    try
    {
        // 利用 update 方法获取数据修改游标
        updatecursor = fclass.Update(updatefilter, false);
    }
    catch (Exception)
    {
        updatecursor = null;
    }
    if (updatecursor != null)
    {
        // 依次轮询结果集中的每一个矢量要素
        IFeature feature = updatecursor.NextFeature();
        while (feature != null)
        {
            // 将指定字段的值设置为用户填写的字符串
            feature.set_Value(nField, txtStringTestValue.Text);
            // set_Value 后实际上并没有真正改变数据集中的值
            // 修改属性值后必须调用游标的 UpdateFeature 方法进行标记,
            // 然后在数据操作结束后会由数据引擎进行统一的更新
            updatecursor.UpdateFeature(feature);
            feature = updatecursor.NextFeature();
        }
    }
}

```

## 六、思考题

1. 请通过操作测试明确图 5.14 中前 8 种空间关系的意义及查询效果。

## 实验六. 矢量要素的渲染

### 一、实验目的

了解 AO 组件中矢量要素的显示机制，掌握基本的渲染方法。

### 二、实验仪器

常规配置微机，Windows 操作系统，Visual Studio2005 及以上版本，ArcGIS Engine9.2 及以上版本。本指导书将以 Visual Studio 2010 和 ArcGIS Engine10.0 为基本开发环境，在其他版本的 Visual Studio 和 ArcGIS Engine 中进行开发的过程仅有细微差别，读者可自行查找相关技术资料进行详细了解和处理。

### 三、实验要求

了解矢量要素的数据组成，了解 AO 组件中的几何图形显示机制，了解对常规渲染方式的效果及作用。

### 四、实验内容

与图形元素类似，矢量要素包含了几何体的空间数据，但要在地图上显示，必须结合相应的显示符号。但与图形元素不同的是，矢量要素的显示符号往往是其某种属性信息的形象化表现(例如专题图)，为矢量图层中的各个要素独立的设置显示符号是不现实也是不合理的。因此，AO 中通过一系列特定的组件，实现了根据指定属性字段信息来为每一个矢量要素自动生成相匹配的显示符号的显示机制，这称为矢量要素的“渲染”。本实验主要通过“同一值渲染”和“分级渲染”学习矢量要素渲染的基本思路和常用组件调用方法。本次实验在上次实验完成系统的基础上进行，仍然在“打开属性表”对话框中实现针对字段的矢量要素渲染功能，当鼠标右键点击对话框中 DataGridView 控件表格的列头是，弹出渲染操作菜单，如图 6.1 所示：

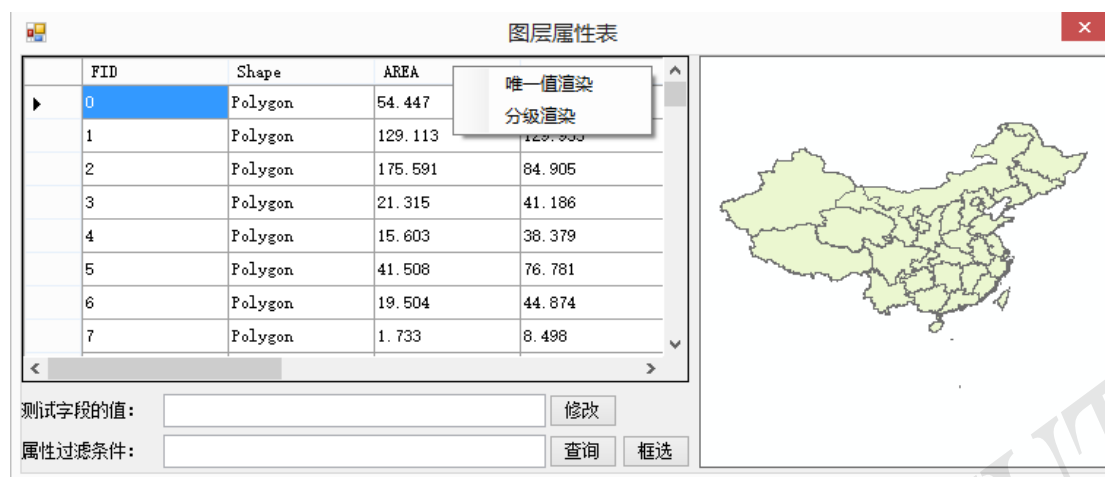


图 6.1. 渲染操作

1. 同一值渲染的实现。选择“同一值渲染”菜单项，将针对指定字段进行图层的同一值渲染（图 6.2 显示针对图中 NAME 字段进行同一值渲染的效果）：

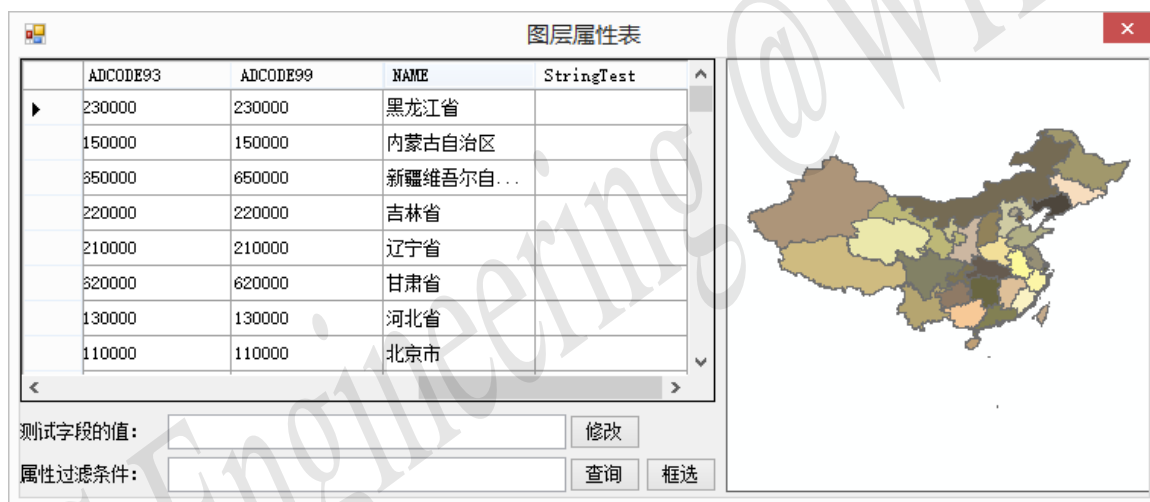


图 6.2. 同一值渲染的参考效果

2. 分级渲染的实现。选择“分级渲染”菜单项，将针对指定字段进行图层的分级渲染（图 6.3 显示针对图中 AREA 字段进行分级渲染的效果）：

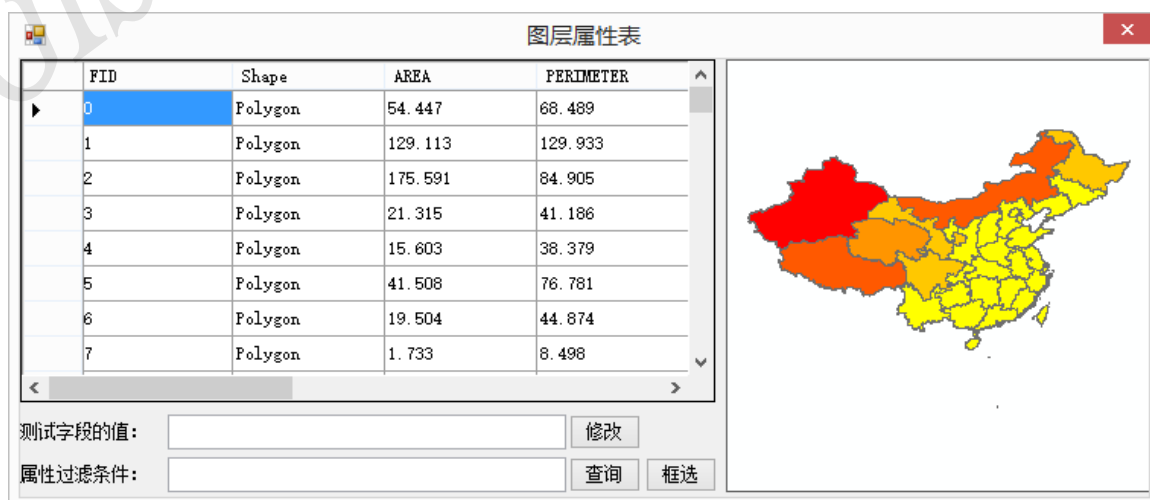


图 6.3. 分级渲染的参考效果

## 五、实验步骤

1. 创建右键菜单并定义函数。在 **LayerAttrib** 类（“打开属性表”对话框的窗体类）的设计器中添加一个新的 **ContextMenuStrip** 控件，对象名设置为 **rendererStrip**，并添加“同一值渲染”和“分类渲染”两个菜单项，菜单项的对象名分别设置为 **uniqueValueItem** 和 **classBreaksItem**，上述控件的对象名可以自由设定，但要注意在参考后文的代码示例时进行相应的修改。为了表达渲染与字段的相关性，我们设定 **rendererStrip** 菜单在点击表格的列头部位时才弹出，而在点击其他部位时仍然按前次实验弹出字段编辑的菜单，因此在 **DataGridView** 控件的 **MouseDown** 事件处理函数中应在前次实验的基础上做如 **CODE6.1** 所示的修改，以实现针对不同点击位置创建不同菜单的功能。

**CODE6.1** DataGridView 控件 MouseDown 事件处理函数的修改

```
private void dataGridView1_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Right)
    {
        // 主要是利用 HitTest 函数进行点击位置的判断
        DataGridView.HitTestInfo hit = dataGridView1.HitTest(e.X, e.Y);
        //如果点击位置是列头，则显示渲染操作菜单
        if (hit.Type == DataGridViewHitTestType.ColumnHeader)
        {
            rendererStrip.Show(dataGridView1, e.X, e.Y);
            // strRenderField 是 LayerAttrib 类中新添加的 string 类型成员变量
            // 作用的保存点击位置的列名信息（也就是对应的要素字段名）
            strRenderField = dataGridView1.Columns[hit.ColumnIndex].Name;
        }
        // 如果是其他位置，显示字段操作菜单
        else
        {
            featureClassStrip.Show(dataGridView1, e.X, e.Y);
        }
    }
}
```

为了实现菜单功能，还需分别添加两个菜单项的点击事件处理函数，它们的实现代码可以参考 CODE6.2 和 CODE6.3。

CODE6.2 “同一值渲染”菜单项的点击事件处理函数

```
private void uniqueValueItem_Click(object sender, EventArgs e)
{
    IFeatureLayer fLayer = axMapControl1.get_Layer(0) as IFeatureLayer;
    RenderByUniqueValue(fLayer, strRenderField);
}
```

CODE6.3 “分级渲染”菜单项的点击事件处理函数

```
private void classBreaksItem_Click(object sender, EventArgs e)
{
    IFeatureLayer fLayer = axMapControl1.get_Layer(0) as IFeatureLayer;
    RenderByClassBreaks(fLayer, strRenderField, 5);
}
```

上述代码中 `RenderByUniqueValue` 和 `RenderByClassBreaks` 是 `LayerAttrib` 类中新添加的两个成员函数，用于实现同一值渲染和分级渲染的功能，它们都需要目标图层和目标字段名作为参数，另外分级渲染函数 `RenderByClassBreaks` 还需要一个表示级数的 `int` 型参数，表示级数可由用户自定义设置，在本例中为了简化操作，统一设置为“5”（参看 CODE6.3）。

2. 创建同一值渲染的基本步骤。同一值渲染实现了根据指定的一个或多个（最多三个）字段的值进行分类显示的渲染方法，在指定字段上值相同的矢量要素被分成同一类，用同一种符号进行显示。在 ArcMap 中将“图层属性”中“显示符号”页的显示类型设置为“Categories->Unique Values”，并进行相应的参数设置（如图 6.4），就能用同一值渲染方式进行要素显示。

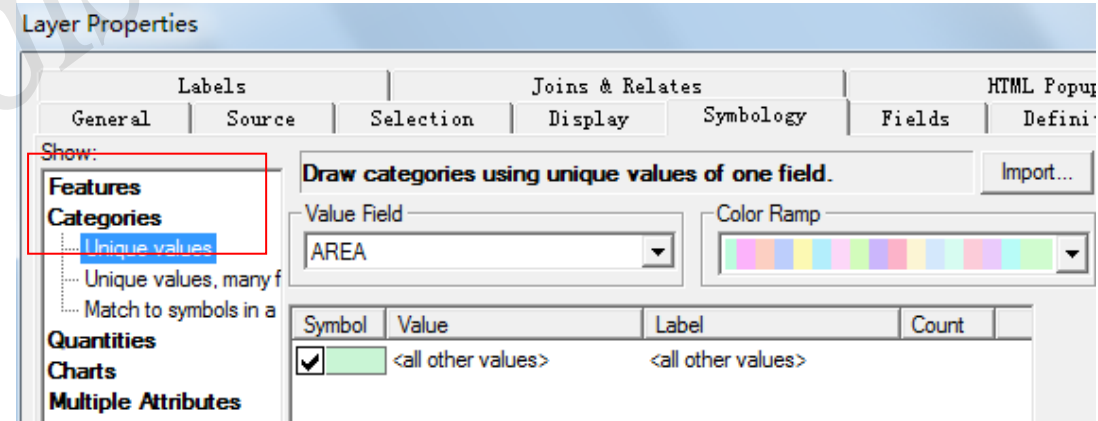


图 6.4. ArcMap 中设置同一值渲染的操作

从 ArcMap 的操作过程中我们也能大致了解实现同一值渲染功能所需的主要参数及其意义。AO 为我们提供了一组用于实现矢量要素渲染功能的组件，它们都实现了 IFeatureRenderer 接口（定义于 ESRI.ArcGIS.Carto 组件库）。其中 UniqueValueRenderer 组件用于同一值渲染，主要定义方式是针对指定字段的每一种值设置一个对应的现实符号，而要让图层按照自定义的 UniqueValueRenderer 对象渲染要素，需要利用 FeatureLayer 组件对象所实现的 IGeoFeatureLayer 接口，在该接口中定义了 Renderer 属性，将该属性设置为自定义的 UniqueValueRenderer 对象，就能实现自定义的同一值渲染。其基本实现框架可以参考 CODE6.4.

CODE6.4 RenderByUniqueValue 函数的基本实现框架

```
private void RenderByUniqueValue(IFeatureLayer fLayer, string strFieldName)
{
    // 由于同一值渲染要针对字段中的每一种值都设定显示符号，
    // 因此首先应得出全部要素在该字段的属性值中的所有相异值。
    // GetUniqueValues 是自定义的用于获取所有相异值的函数，参见 CODE6.5
    // 该函数返回由所有相异值组成的字符串数组，
    // 如果指定名称的字段不存在，则返回 null。
    // 要素在该字段上的值类型不是字符串类型，则将其值转换为字符串，
    // 之所以要将各种类型的字段值统一转换为字符串类型，
    // 是因为 UniqueValueRenderer 组件在构建字段值与显示符号的对应关系时，
    // 要求采用字符串形式的字段值。
    string[] strUniqueValues = GetUniqueValues(fLayer, strFieldName);
    if (strUniqueValues != null)
    {
        // 创建一个 UniqueValueRenderer 组件对象
        IUniqueValueRenderer uniquerenderer = new UniqueValueRendererClass();
        // FieldCount 属性用于设置渲染所针对的字段数量，
        // 通常针对一个字段进行渲染，也可针对多个字段进行联合判断，
        // 最多可以联合 3 个字段。
        uniquerenderer.FieldCount = 1;
        // Field 属性用于设置目标字段的字段名，
        // 由于可以有多个目标字段，因此 Field 属性是利用索引参数进行访问的
        // 在调用时应转换为对应的 get_Field 或 set_Field 函数形式
        uniquerenderer.set_Field(0, strFieldName);
        // 以下代码是为了设置目标字段类型（通过 FieldType 属性），
```

```

// 这是 UniqueValueRenderer 组件比较特殊的地方，
// 前面提到 UniqueValueRenderer 组件在构建字段值与显示符号的对应关
// 系时，要求采用字符串类型的字段值，
// 那么组件对象在进行自动渲染的过程中，就需要根据实际的字段值类型
// 来判断是否要将要素字段值转换为字符串。
// 设置目标字段类型主要是为了区分字段值类型是否为字符串，
// 因此该类型属性是一个 bool 型变量。
IFeatureClass fclass = fLayer.FeatureClass;
int nField = fclass.FindField(strFieldName);
IField field = fclass.Fields.get_Field(nField);
// 根据实际的字段值类型设置 UniqueValueRenderer 组件对象的字段类型
if (field.Type == esriFieldType.esriFieldTypeString)
{
    // 如果实际字段值类型是字符串型，
    // 则 UniqueValueRenderer 组件对象的字段类型设置为 true。
    // 由于可以有多个目标字段，
    // 因此 FieldType 属性也是利用索引参数进行访问的，
    // 在调用时应转换为对应的 get_FieldType 或 set_FieldType 函数形式
    uniquerenderer.set_FieldType(0, true);
}
else
{
    // 如果实际字段值类型不是字符串型，
    // 则 UniqueValueRenderer 组件对象的字段类型设置为 false。
    uniquerenderer.set_FieldType(0, false);
}
// 此处是构建字段值与显示符号的对应关系的代码，参见 CODE6.6-6.7
// .....
// 需要利用 IGeoFeatureLayer 接口设置自定义渲染方式。
IGeoFeatureLayer geoLayer = fLayer as IGeoFeatureLayer;
if (geoLayer != null)
{
    // 设置为自定义的渲染组件对象
    geoLayer.Renderer = uniquerenderer as IFeatureRenderer;
    IActiveView activeview = axMapControl1.ActiveView;

```

```

        // ContentsChanged 方法是为了通知 TOC 控件更新显示内容，
        // TOC 控件将在图层项下显示渲染信息。
        activeview.ContentsChanged();
    // 刷新显示窗口，立即显示新的渲染效果，
    // 注意 esriViewDrawPhase 参数
    activeview.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);
    }
}
}

```

#### CODE6.5 获取指定字段上所有相异值的函数代码示例

// 该函数主要思路是先获取该字段上全部要素的属性值，然后再剔除重复值。  
 // 由于不涉及新的 AO 组件操作，此处不进行详细分析，请读者自行分析。

```

string[] GetUniqueValues(IFeatureLayer fLayer, string strFieldName)
{
    if (fLayer != null)
    {
        List<string> strFieldValues = new List<string>();
        IFeatureClass fclass = fLayer.FeatureClass;
        int nField = fclass.FindField(strFieldName);
        if (nField != -1)
        {
            IFeatureCursor fcursor = fclass.Search(null, false);
            IFeature feature = fcursor.NextFeature();
            while (feature != null)
            {
                strFieldValues.Add(feature.get_Value(nField).ToString());
                feature = fcursor.NextFeature();
            }
            strFieldValues.Sort();
            for (int i = 1; i < strFieldValues.Count; i++)
            {
                if (strFieldValues[i] == strFieldValues[i - 1])
                {
                    strFieldValues.RemoveAt(i);
                }
            }
        }
    }
}

```



```

        i--;
    }
}

return strFieldValues.ToArray();
}
}
return null;
}

```

回到对 CODE6.4 的分析中，UniqueValueRenderer 组件对象构建字段值与显示符号的对应关系主要利用 IUniqueValueRenderer 接口中定义的 AddValue 方法，该方法的定义见图 6.5，

```

[C#]public void AddValue (
    string Value,
    string Heading,
    ISymbol Symbol) ;

```

图 6.5. AddValue 方法的定义

其中 Value 参数就是转换为 string 类型后的字段值，Symbol 参数就是对应于该值的显示符号，而 Heading 参数是用于在 TOCControl 控件中显示的渲染信息。AddValue 函数的调用规则非常简单，但在使用的过程中常常面临的问题是如何有效的为每一种值创建一种合适的显示符号，通常情况下我们希望这些显示符号有统一的风格，只是在部分表达特征上具有可分辨的差别，最常用的处理方式之一是采用同一种符号，但使用不同的绘制颜色（如图 6.2-6.3）。那么如何根据不同情况有效的设置绘制颜色？在同一值比较多的情况下，全部一一进行手工设置是比较繁琐的事，因此 AO 提供了一组可以用于按不同需求自动产生颜色序列的组件，称为“色带”（color ramp）组件。所有的色带组件都实现了 IColorRamp 接口，从该接口的帮助文档中可以查看 AO 提供的所有色带组件（如图 6.6）。

#### Classes that implement IColorRamp

Classes	Description
AlgorithmicColorRamp	Defines an algorithmic color ramp, where ramp is defined by two colors and the algorithm used to traverse the intervening color space between them.
MultiPartColorRamp	Defines a multi part color ramp, where ramp is defined by a list of constituent color ramps.
PresetColorRamp	Defines a preset color ramp, where ramp is defined by a list of exactly 13 manually specified colors.
RandomColorRamp	Defines a random color ramp, where ramp is a list of randomly picked colors.

图 6.6. AO 提供的色带组件

本次实验中主要运用 `RandomColorRamp` 和 `AlgorithmicColorRamp`，其中 `RandomColorRamp` 在同一值渲染中最常用，而 `AlgorithmicColorRamp` 将在分级渲染部分进行介绍。

### 3. 随机色带的运用。

顾名思义，`RandomColorRamp` 所创建的色带中各个元素的颜色值是随机的，因此选用此种色带中的颜色绘制出的矢量图形具有较强的对比度，正好有利于表现同一值渲染时的字段值分类效果。`RandomColorRamp` 生成随机色彩的原理非常简单，它在指定的颜色值区间内利用均匀分布随机数产生颜色值，但其特别之处在于它所使用的颜色值不是常用的 RGB 模式，而是 HSV 模式，即通过 Hue(色彩)、Saturation(纯度)和 Value(明度)三个分量合成表示颜色，在该模式下颜色的对应取值可以用图 6.7 的形式表达，其中 H (Hue) 分量是 0-360 的整数，表示颜色的色彩；S (Saturation) 分量是 0-100 的整数，表示颜色的浓淡效果；V (Value) 分量也是 0-100 的整数，表示颜色的明暗效果。HSV 模型的优势是能有效的表达颜色的视觉效果。`RandomColorRamp` 组件所实现的 `IRandomColorRamp` 接口定义了 `StartHue/EndHue`、`MinSaturation/MaxSaturation` 和 `MinValue/MaxValue` 三组属性分别用于设置 H、S、V 分量的取值范围，然后在指定范围内通过均匀分布随机数的形式产生随机颜色。设置好颜色值范围后，还需要通过 `IRandomColorRamp` 接口定义的 `size` 属性设置色带中的颜色数量，最后通过 `IRandomColorRamp` 接口定义的 `CreateRamp` 方法产生色带，色带本质上是一个颜色列表，可以通过 `IRandomColorRamp` 接口定义的 `Color` 属性获取列表中的每一个颜色。

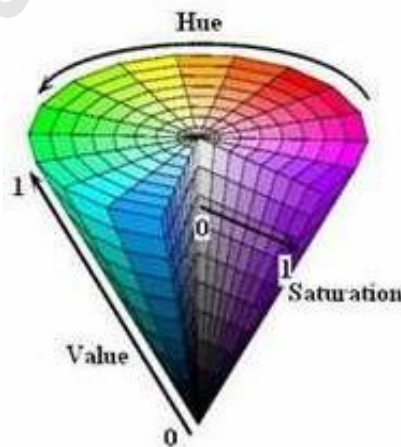


图 6.7. HSV 颜色模型

结合 CODE6.4，在本实验中使用 `RandomColorRamp` 组件的过程可以参考 CODE6.6.

CODE6.6 使用 <code>RandomColorRamp</code> 组件
--

```

IRandomColorRamp randomramp = new RandomColorRampClass();
// 设置颜色值范围
randomramp.StartHue = 30;
randomramp.EndHue = 60;
randomramp.MinSaturation = 20;
randomramp.MaxSaturation = 40;
randomramp.MinValue = 0;
randomramp.MaxValue = 100;
// 设置色带中的颜色数量，其中 strUniqueValues 变量的意义见 CODE6.4
randomramp.Size = strUniqueValues.Length;
bool bOk;
// 调用 CreateRamp 方法，请注意调用规则，特别是参数的规则
randomramp.CreateRamp(out bOk);
// CreateRamp 方法的参数用于指示创建是否成功
if (bOk == true)
{
    // 如果创建成功，依次获取每一个颜色，并创建相应的显示符号。
    // 此处参见 CODE6.7
}

```

如果仅需用通过颜色来区分不同类别的要素，那么就可以直接根据随机色带的结果为每一种同一值设置相应的显示符号，具体过程可以参考 CODE6.7.

#### CODE6.7 根据随机色带的结果为每一种同一值设置相应的显示符号

```

// 依次为每一种同一值创建显示符号
for (int i = 0; i < strUniqueValues.Length; i++)
{
    ISymbol rendersymbol = null;
    // 如果是多边形，创建填充显示符号
    if (fclass.ShapeType == esriGeometryType.esriGeometryPolygon)
    {
        // 每一种同一值都要使用一个单独的显示符号组件对象
        rendersymbol = new SimpleFillSymbolClass();
        (rendersymbol as IFillSymbol).Outline.Width = 1.0;
        // 利用随机色带产生的颜色为显示符号的填充色属性赋值，
        // randomramp 变量的意义见 CODE6.6，
    }
}

```

```

        // IRandomColorRamp 接口的 Color 是一个带参数（索引值）的属性，
        // 需要采用 get_Color 的函数形式获取色带中指定索引值的颜色。
        (rendersymbol as IFillSymbol).Color = randomramp.get_Color(i);
        // 利用 AddValue 方法添加同一值与显示符号的关联，
        // AddValue 方法的调用规范参见图 6.5 及相关说明
        uniquerenderer.AddValue(strUniqueValues[i], strUniqueValues[i], rendersymbol);
    }
    // 如果是多义线，创建填充线型显示符号
    else if (fclass.ShapeType == esriGeometryType.esriGeometryPolyline)
    {
        rendersymbol = new SimpleLineSymbolClass();
        (rendersymbol as ILineSymbol).Width = 1.0;
        (rendersymbol as ILineSymbol).Color = randomramp.get_Color(i);
        uniquerenderer.AddValue(strUniqueValues[i], strUniqueValues[i], rendersymbol);
    }
    // 如果还要处理其他的几何体类型，参考上述模式进行设置.....
}

```

#### 4. 创建分级渲染的基本步骤。

AO 中矢量要素的“渲染”本质上说就是按类别设置要素的显示符号，而要素所属类别是根据目标字段的取值而定的。如果我们认为目标字段上取值相同的要素属于同一类，那么采用同一值渲染是很合适的，此时目标字段本身就等价于其类别属性，这种情况下目标字段的取值往往仅有少数的若干种情况，且值类型往往是整型、字符串型等等离散数据类型。但如果目标字段对应于要素的某种自然属性时，同一值渲染往往就不适用了，这种情况下字段值随着要素个体差异而产生或显著或细微地差异，此时字段值本身不能直接标示要素类别，这类字段取值通常是浮点型等连续数据类型。

一种典型的情况是，假如我们要针对某地区行政区划的面积字段来进行渲染，此时更常用的分类方法不是将不同面积的区划分为不同的类别，而是设置若干个面积等级，每个等级对应于一个面积取值区间，然后根据区划的面积所属的等级来进行分类，也就是“分级”，针对不同的级别设置不同的显示符号。“分级”是针对自然属性字段的一种常用分类模式，AO 也相应地提供的用于分级渲染的组件——ClassBreaksRenderer。该组件的常用方法和属性都定义于 IClassBreaksRenderer 接口，其成员列表如图 6.8 所示，其中最重要的是 Break 和 Symbol 属性，Break 属性以列表的形式保存了每一个级别的值域范围分界点，

Symbol 属性与之对应的以列表形式保存了每一个级别要素的显示符号。如果每一个级别可以用值域(a,b]表示,那么 Break 属性实际上存储的是每个级别的 b 值。由于各等级的值域是彼此连续的, 因此仅存储 b 值就可以完整划分每一个值域,除了第一个值域, 第一个值域必须还要存储 a 值, 这个值保存在 MinimumBreak 属性中。

Members











	All ▾	Description
	BackgroundSymbol	Background fill symbol used when graduated marker symbols are draw on polygon features.
	Break	Break value at the specified index. Break(0) is the lowest break and represents the upper bound of the lowest class.
	BreakCount	Number of class breaks (equal to the number of classes).
	Description	Description at the specified index.
	Field	Classification field.
	Label	Label at the specified index.
	MinimumBreak	Minimum break, i.e. the lower bound of the first class.
	NormField	Normalization field.
	SortClassesAscending	Indicates if classes are displayed in increasing order in legends/TOC.
	Symbol	Symbol at the specified index (used to draw features in the specified class).

图 6.8. IClassBreaksRenderer 接口成员列表

在 ArcMap 中我们使用 Quantities 的显示类型就可以使用分级渲染,而在 ArcMap 的 TOC 部分会显示对应图层的分级信息 (如图 6.9), 那么这个分级信息与 MinimumBreak 和 Break 属性的关系可以用图 6.10 说明。

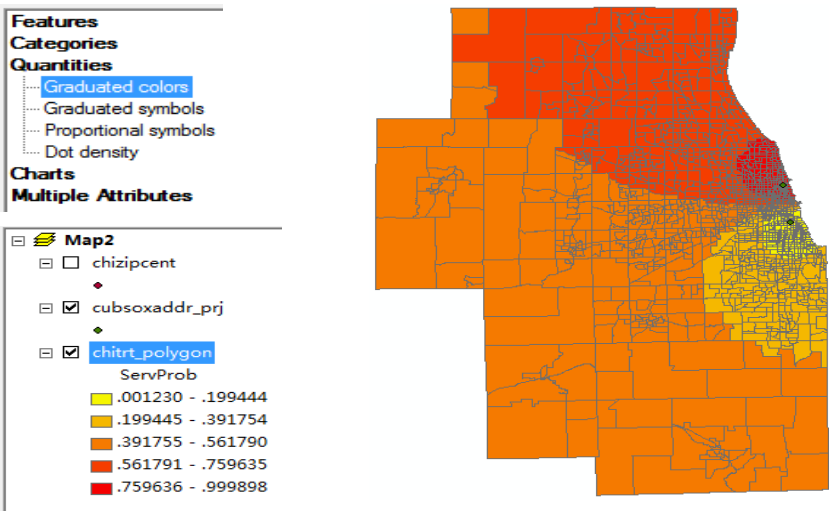


图 6.9. ArcMap 中分级渲染的效果

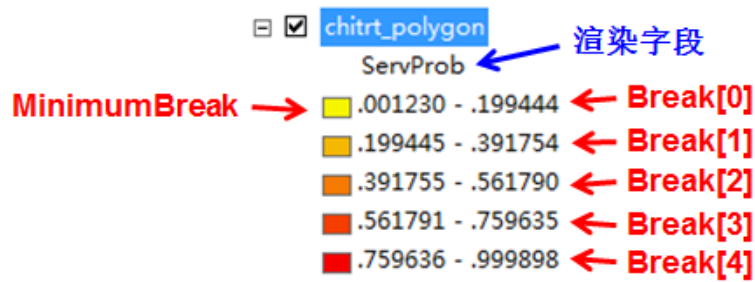


图 6.10. TOC 中分级信息与 MinimumBreak 和 Break 属性的对应关系

使用分级渲染的过程实际上就是定义值域等级并设置相应显示符号的过程，因此 CODE6.3 中的 RenderByClassBreaks 函数可以参考 CODE6.8.

#### CODE6.8 RenderByClassBreaks 函数

```
private void RenderByClassBreaks(IFeatureLayer fLayer, string strFieldName, int
nBreaks)
{
    if (fLayer != null)
    {
        IFeatureClass fclass = fLayer.FeatureClass;
        int nField = fclass.FindField(strFieldName);
        // 分级渲染通常针对浮点型数据，所以此处对字段值类型进行判断
        if (nField != -1 &&
            fclass.Fields.get_Field(nField).Type == esriFieldType.esriFieldTypeDouble)
        {
            // 创建分级渲染组件对象
            IClassBreaksRenderer classrenderer = new ClassBreaksRendererClass();
            // 设置目标字段名称
            classrenderer.Field = strFieldName;
            // 函数的 nBreaks 参数是人为定义的分级级数，
            // 但这个级数不一定适用于目标字段值的全局统计特征，
            // 因此往往要添加一个统计分级的过程，参见 CODE6.10，
            // 统计分级后的结果将自动得到分级的合理值域和级数，
            // 然后再设置渲染分级的等级数。
            classrenderer.BreakCount = nBreaks;
            // 本函数使用不同颜色区分类别要素类别，
            // 使用算法色带创建对应的颜色列表。
            IAlgorithmicColorRamp algorithmramp = new AlgorithmicColorRampClass();
            // 此处要对算法色带进行相应的设置，参见 CODE6.9.
```

```

        // 定义色带中的颜色数量
algorithmramp.Size = nBreaks;
        bool bOk;
        // 创建色带
        algorithmramp.CreateRamp(out bOk);
        if (bOk)
        {
            // 如果色带创建成功，利用色带中的颜色创建相应等级的
            // 显示符号，参见 CODE6.11.
        }
        // 设置图层渲染，此部分与 CODE6.4 相同
        IGeoFeatureLayer geoLayer = fLayer as IGeoFeatureLayer;
        if (geoLayer != null)
        {
            geoLayer.Renderer = classrenderer as IFeatureRenderer;
            IActiveView activeview = axMapControl1.ActiveView;
            activeview.ContentsChanged();
            activeview.PartialRefresh(esriViewDrawPhase.esriViewGeography, null, null);
        }
    }
}
}
}

```

## 5. 算法色带的运用。

在同一值渲染中我们使用了 **RandomColorRamp** 来产生颜色差异明显的随机色带，正好有利于区分要素的类别差异，而在分级渲染中，目标字段通常代表了同一种自然属性，而字段值的差异代表了要素在同一属性中的不同等级或者强度，因此通常习惯于用渐变效果的颜色进行渲染。

算法色带组件 **AlgorithmicColorRamp** 可以方便的实现渐变颜色效果，它产生的颜色列表中的颜色是从指定起始颜色值至指定终止颜色值的渐变颜色，而且它采用 **RGB** 模式定义起始颜色和终止颜色，使用非常方便，参见 **CODE6.9**。

### CODE6.9 生成从黄色至红色的渐变颜色色带

```

// 变量 algorithmramp 定义于 CODE6.9
// 首先要设置色带生成算法，也就是 Algorithm 属性，
// 要产生渐变色带，通常采用 esriCIELabAlgorithm 算法。

```

```

algorithmramp.Algorithm = esriColorRampAlgorithm.esriCIELabAlgorithm;
// 定义起始颜色值，采用 RGB 模式
IRgbColor fromcolor = new RgbColorClass();
// 黄色
fromcolor.Red = 255;
fromcolor.Green = 255;
fromcolor.Blue = 0;
// 定义终止颜色值，采用 RGB 模式
IRgbColor tocolor = new RgbColorClass();
// 红色
tocolor.Red = 255;
tocolor.Green = 0;
tocolor.Blue = 0;
// 设置色带的起始颜色和终止颜色
algorithmramp.FromColor = fromcolor;
algorithmramp.ToColor = tocolor;

```

## 6. 实现自动分级。

CODE6.8 中预留了利用字段值的统计特征进行自动分级的过程，这一过程是分级渲染中常用的步骤，首先它能根据要素的字段值总体特征自动产生等级值域，省掉了手工分级的麻烦，更重要的是它可以按要求产生符合统计规律的分级方案，往往比手工分级更合理。AO 中提供了一系列自动分级组件（如图 6.11），他们都实现了 IClassifyGEN 接口（另一个共同的接口是 IClassify，但它已被 IClassifyGEN 取代）。

### Classes that implement IClassifyGEN

Classes	Description
<a href="#">DefinedInterval</a>	Defines a defined interval classification method.
<a href="#">EqualInterval</a>	Defines an equal interval classification method.
<a href="#">GeometricalInterval</a>	Defines a geometrical interval classification method.
<a href="#">NaturalBreaks</a>	Defines a natural breaks classification method.
<a href="#">Quantile</a>	Defines a quantile classification method.
<a href="#">StandardDeviation</a>	Defines a standard deviation classification method.

图 6.11. AO 中的自动分级组件

不同的组件采用了不同的分级算法，具体说明可见图 6.11 表格的 Description



列，调用 IClassifyGEN 接口的 Classify 方法后就可以得到分级结果，该方法的说明见图 6.12。从说明中可以看出，AO 中的自动分级主要是基于字段值的频度统计特征的，其中 doubleArrayValues 参数应为 double[] 型，表示全部要素的目标字段值按升序排列的列表，longArrayFrequencies 参数应为 int[] 型，表示每个值出现的频度，numClasses 参数是一个引用类型参数，它用于设置分级的级数，并返回根据算法计算后的合理级数。当然，对全部要素进行字段值频度统计往往比较麻烦，利用 AO 提供的直方图统计组件（BasicTableHistogram）可以进行自动统计，因此该组件在分级渲染中很常用。

```
[C#]public void Classify (
    object doubleArrayValues,
    object longArrayFrequencies,
    ref int numClasses) ;
```

图 6.12. AO 中的自动分级组件

本实验以最简单的等间距分级为例介绍自动分级的过程，如 CODE6.10 所示，等间距分级就是按照指定级数对[最小字段值，最大字段值]的值域进行等间距分段。

CODE6.10 等间距自动分级的实现
<pre>// 使用 BasicTableHistogram 组件 ITableHistogram histo = new BasicTableHistogramClass(); // 设置直方图统计的目标字段，strFieldName 变量定义见 CODE6.8. histo.Field = strFieldName; // 设置直方图统计的目标数据表，也就是矢量要素所在的要素类。 histo.Table = fclass as ITable; //dbArray 变量用于获取统计得到的字段值列表， // nArray 变量用于获取相应的频度列表。 object dbArray, nArray; // 调用自动直方图统计。 (histo as IBasicHistogram).GetHistogram(out dbArray, out nArray); // 将统计结果相应的转换为 double[]和 int[]类型 double[] DataArray = dbArray as double[]; int[] FreqArray = nArray as int[]; // 创建一个等间距分级组件 IClassifyGEN classify = new EqualIntervalClass(); // 进行自动分级，nBreaks 参数是预设值，分级结束后将返回合理的级数。 classify.Classify(DataArray, FreqArray, nBreaks);</pre>

利用自动分级的结果和自动创建渐变色带的结果，可以依次定义渲染中每个等级的显示符号，参考 CODE6.11.

#### CODE 6.11 创建分级显示符号

```
// 本例中的未定义变量都来自 CODE6.8-6.10.
// classify 变量的定义见 CODE6.10，它是一个自动分级组件，
// 其 IClassifyGEN 接口定义的 ClassBreaks 属性就是自动分级后的各级别分断点。
double[] Breaks = classify.ClassBreaks as double[];
// ClassBreaks 中的第一个分断点就是整个值域的起点，
// 对应于分级渲染中的 MinimumBreak。
classrenderer.MinimumBreak = Breaks[0];
// 依次设置分级渲染中的每一级的 Break 和 Symbol 属性。
for (int i = 0; i < classrenderer.BreakCount; i++)
{
    // 请注意分级渲染中的级数与 ClassBreaks 中的级数的对应关系。
    classrenderer.set_Break(i, Breaks[i + 1]);
    // 设置标签，这个标签就是显示在 TOC 中的分级信息。
    classrenderer.set_Label(i, Breaks[i].ToString() + "-" + Breaks[i + 1].ToString());
    ISymbol rendersymbol = null;
    // 如果是多边形要素，创建填充型显示符号。
    if (fclass.ShapeType == esriGeometryType.esriGeometryPolygon)
    {
        rendersymbol = new SimpleFillSymbolClass();
        (rendersymbol as IFillSymbol).Outline.Width = 1.0;
        // 填充颜色来自算法色带创建的渐变色列表。
        (rendersymbol as IFillSymbol).Color = algorithmramp.get_Color(i);
        // 设置每一级对应的显示符号，这与同一值渲染的 AddValue 不同。
        classrenderer.set_Symbol(i, rendersymbol);
    }
    // 如果是多义线要素，创建线型显示符号。
    else if (fclass.ShapeType == esriGeometryType.esriGeometryPolyline)
    {
        rendersymbol = new SimpleLineSymbolClass();
        (rendersymbol as ILineSymbol).Width = 1.0;
        // 线条颜色来自算法色带创建的渐变色列表。
    }
}
```

```
(rendersymbol as ILineStyleSymbol).Color = algorithmramp.get_Color(i);  
classrenderer.set_Symbol(i, rendersymbol);  
}  
// 如果还要处理其他的几何体类型，参考上述模式进行设置.....  
}
```

## 六、思考题

1. 请选择除 EqualInterval 之外的至少两种自动分级组件进行分级渲染测试，并解释其意义。

## 实验七. 网络数据集环境下的路径分析

### 一、实验目的

了解网络数据集的创建、加载和路径分析等基本操作的实现方法，认识网络数据集的组织结构和操作特点。

### 二、实验仪器

常规配置微机，Windows 操作系统，Visual Studio2005 及以上版本，ArcGIS Engine9.2 及以上版本。本指导书将以 Visual Studio 2010 和 ArcGIS Engine10.0 为基本开发环境，在其他版本的 Visual Studio 和 ArcGIS Engine 中进行开发的过程仅有细微差别，读者可自行查找相关技术资料进行详细了解和处理。

### 三、实验要求

熟悉在 ArcMap 中创建网络数据集和进行路径分析的基本操作，了解 Geodatabase 数据模型的基本结构及其在 AO 中的对应组件和接口，掌握打开工作空间并读取数据的基本操作。

### 四、实验内容

本次实验定位于综合性和设计性实验，需要基于前次实验所构建的软件系统，添加网络数据集创建和加载的功能、并能在网络数据集中进行简单的路径计算。本节实验指导书并不给出完整的实现方案和步骤，仅就网络数据集操作的相关基础组件和基本调用方法进行介绍，由实验者自行设计软件的相关功能并编写实现代码。另外，在实验过程中，希望实验者能利用 ArcCatalog 和 ArcMap 进行网络数据集的相关操作，以对比理解基本的属性设置和运行结果。

### 五、基本实验原理和组件调用方法

#### 1. 在 ArcEngine 中调用网络分析模块的注意事项。

ArcGIS 中进行用于网络分析的数据模型主要有两类：Geometry Network（几何网络）和 Network Dataset（网络数据集），分别适用于效用网络建模和交通网络建模，处理这两种网络的扩展模块分别是 Network Analysis 和 Network Analyst。本次课程主要介绍基于网络数据集的分析应用，在 ArcObjects 中有一系列的组件和接口用来实现对网络数据集的各种操作，但是在 ArcEngine 中要调用这些功能，需要在 LicenseControl 中启用 Network Analyst 模块，具体方法如图 7.1-7.2 所示。除此之外，还应在项目的“引用”中添加 Network Analyst 程序集引用（如图 7.3）。

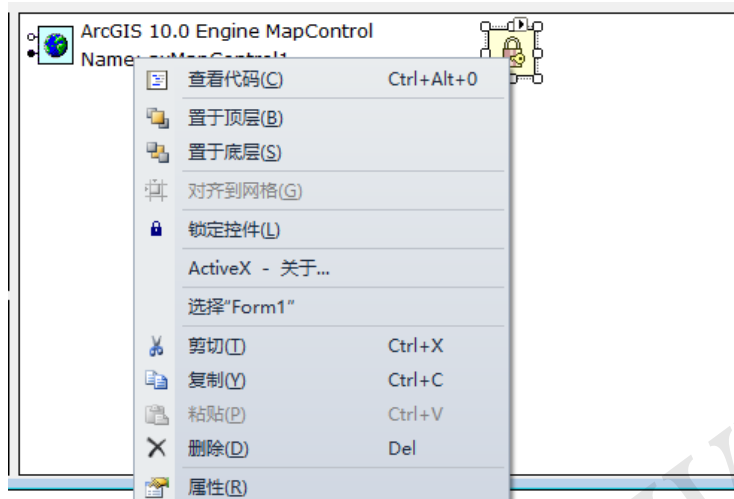


图 7.1. 设置 LicenseControl 的属性

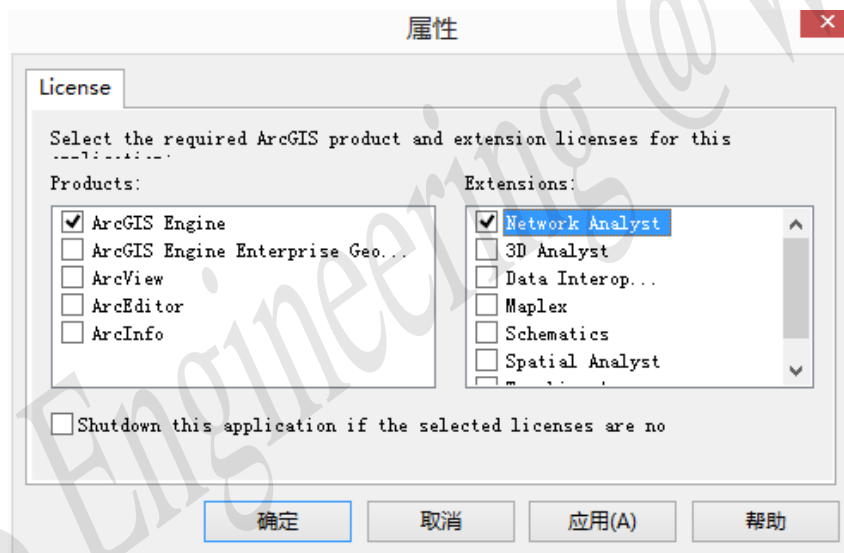


图 7.2. 启用 Network Analyst 模块

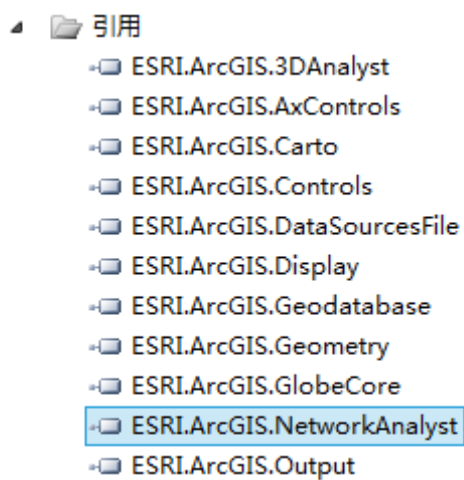


图 7.3. 添加 Network Analyst 程序集引用

## 2. 加载网络数据集图层的基本方法。

ArcGIS 系列产品采用 Geodatabase 数据模型来统一组织各类数据，并采用类似的数据访问框架。前面的实验中已经介绍过利用 Geodatabase API 读取 Shapefiles 数据的方法，那么加载网络数据集的方法与之类似，仍然要通过 Workspace 来管理和组织数据。回顾实验四中介绍的加载 shapefile 图层的过程，基本步骤是先利用 ShapefileWorkspaceFactory 组件打开 shapefile 文件所在的目录并生成对应的 Workspace，这个目录可以看作一个数据库，然后每一组 shapefile 文件相当于数据库中的一个 Dataset（数据集），可以通过枚举的方式依次访问每一个数据集，并根据文件名称找到所要加载的数据集。

如果在 shapefile 文件目录中有一个网络数据集（如果基于某个 shapefile 图层创建网络数据集，那么这个网络数据集就在图层文件的同一个目录中），那么加载该数据集的过程与加载其中的 shapefile 文件是非常类似的：首先通过 ShapefileWorkspaceFactory 组件生成文件目录对应的 Workspace，然后找到网络数据集对应的 Dataset。但是其中有一点特殊之处，网络数据集并不是 shapefile 格式的文件，它是寄存在 shapefile 数据库中的另外一种数据格式，因此它不能作为 shapefile 的数据集被读取。这种在某种类型的数据库中包含其他类型数据的结构称为“工作空间扩展”（workspace extension），一个工作空间扩展可以看作某个数据库中的一个子库，AO 提供了一组用于操作工作空间扩展的组件，它们都具有 IWorkspaceExtension 接口，这个接口中提供了大部分常用的属性和方法。因此，要读取 shapefile 文件目录中的网络数据集文件，必须先获取对应的工作空间扩展，然后在这个子库中读取网络数据集对应的数据集（dataset），Workspace 组件实现了 IWorkspaceExtensionManager 接口用于管理它所支持的工作空间扩展，可以通过该接口或者所需的子库，再进行进一步的数据读取操作。假设我们要打开如图 7.4 所示的目录中的 routeseg\_ND.nd 网络数据集（注意，在文件型数据存储模式中，网络数据集是一个子目录），则主要步骤如下：

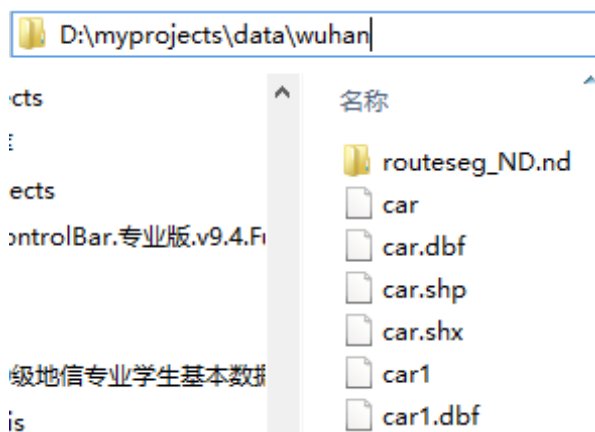


图 7.4. 加载网络数据集案例的文件路径结构

2.1. 假设分别用 sShapefilePath 和 sNDName 两个字符串变量分别表示 shapefiles 数据目录（本例中为 “D:\myprojects\data\wuhan”）和网络数据集的名称（本例中为 “routeseg\_ND.nd”）；

2.2. 获得与 shapefiles 数据目录对应的 Workspace 组件对象，参考 CODE7.1

CODE7.1 打开 shapefiles 文件所在的工作空间

```
IWorkspaceFactory workspaceFactory = new ShapefileWorkspaceFactoryClass() as
IWorkspaceFactory;
IWorkspace workspace = workspaceFactory.OpenFromFile(sShapefilePath, 0);
```

2.3. 获得对应于网络数据集的 workspace extension 组件对象，参考 CODE7.2

CODE7.2 获得对应于网络数据集的工作空间扩展

```
IWorkspaceExtensionManager wsExtMgr = workspace as IWorkspaceExtensionManager;
UID myUID = new UIDClass();
myUID.Value = "esriGeoDatabase.NetworkDatasetWorkspaceExtension";
// 请注意 FindExtension 方法的调用方式及参数设置
IWorkspaceExtension wsExt = wsExtMgr.FindExtension(myUID);
// 工作空间扩展的 IDatasetContainer2 接口定义了读取网络数据集的方法
IDatasetContainer2 dsCont = wsExt as IDatasetContainer2;
```

2.4. 打开指定名称（本例中为 sNDName）的网络数据集，参考 CODE7.3

CODE7.3 打开指定名称的网络数据集

```
// 利用 IDatasetContainer2 接口可以直接通过名称读取数据集，
// 注意 DatasetByName 属性的变量类型和参数设置。
IDataset dataset = dsCont.get_DatasetByName(esriDatasetType.esriDTNetworkDataset,
                                             sNDName);
// INetworkDataset 接口就是网络数据集组件的基本接口。
INetworkDataset networkDataset = dataset as INetworkDataset;
```

2.5. 创建网络图层，参考 CODE7.4

CODE7.4 创建网络数据集图层

```
// 一个网络数据集图层可以关联网络数据集对象，
// 并能作为图层添加到 map 中。
// 可以参考 FeatureLayer 和 FeatureClass 的关系。
layerNetwork = new NetworkLayerClass();
layerNetwork.NetworkDataset = networkDataset;
axMapControl1.AddLayer(layerNetwork as ILayer);
```

3. 基于 shapefile 图层创建简单的网络数据集的基本方法。

Shapefiles 格式的矢量网络数据在创建网络数据集之前要进行一些数据的初始编辑工作，例如线矢量的分段化、拓扑检查等，这些功能也可以通过 AO 二次开发实现，但为了简化内容，本课程只在初始化编辑工作完成后的矢量网络数据基础上简要介绍创建网络数据集的基本方法，这些方法（在数据读取方面）稍加改变就可以应用于其它数据格式的网络数据集构建方面。

由于整个 ArcGIS 产品都是基于 AO 组件库开发的，因此我们往往可以通过 ArcGIS 软件的操作来了解二次开发中的基本步骤和参数。如果我们利用 ArcCatalog 软件实践一个 shapefiles 网络数据集创建过程就可以大致总结出以下几个特点：

- (1) 最基础的数据是描述网络连接的线状矢量要素类；
- (2) 网络数据集创建过程不是完全自动化的，需要人工配置一些参数和规则；
- (3) 必须设置的参数有以下几个：
  - a. 网络数据集的名称；
  - b. 网络连通性设置，此处 ArcCatalog 软件提供的默认配置，即在矢量线段的端点处连接，也可以人工修改连接规则；
  - c. 设置属性，网络数据集可以指定的属性包括成本、等级、描述符、限制四种，为了进行后续的网络分析，必须指定至少一个属性，作为路径分析，必须设置成本属性；属性值将与具体的字段相关。
- (7) 可选设置的参数有以下几个：
  - a. 在网络数据集中构建转弯模型；
  - b. 对网络数据进行高程建模；
  - c. 建立行驶方向设置。
- (8) 可选设置不是必须的，但如果选择进行某项设置，则在后续过程中可能需要进行其他必须的相关设置。

为了简化内容，我们在创建过程中忽略所有可选项，相关设置方法可以参考帮助文档详细了解。从上述总结可以看出，网络数据集的创建的总体步骤分为三个阶段：

- (1) 选择基础线状矢量要素类；
- (2) 设置相应的参数；
- (3) 生成网络数据集。

假设要基于一个进行过分段化和拓扑检查的线状矢量图层构建网络数据集，图层对象由变量 `fSourceLayer` 表示，类型为 `IFeatureLayer`，可以参考 CODE7.5 创建出一个能用于最短路径计算的简单网络数据集，该示例中涉及到一些新的组件和接口，此处不做详细说明，相关资料可以查阅帮助文档，该示例所创建的网络



数据集只设置了简单的属性信息，仅用于最短路径计算，如要进行更复杂的应用，请参考帮助文档中的详细介绍。

#### CODE7.5 基于线状矢量图层 fSourceLayer 创建网络数据集

```
// AO 提供了 DENetworkDataset 组件用于创建和操作网络数据集；
// 该组件的常用主要接口是 IDENetworkDataset2。
IDENetworkDataset2 deNetworkDataset = new DENetworkDatasetClass();
// 要将该组件对象设置为“可构建”，才能执行创建网络数据集的工作。
deNetworkDataset.Buildable = true;
// 通过 ArcCatalog 和 ArcMap 的相关操作可知，网络数据集自身具有空间信息，
// 该信息来源于源矢量图层，因此要将网络数据集的空间引用信息与源图层
// 的相关信息对应起来，以下操作将源图层的范围和空间引用赋给网络数据集
IGeoDataset geoSource = fSourceLayer as IGeoDataset;
IDEGeoDataset deGeoDataset = (IDEGeoDataset)deNetworkDataset;
deGeoDataset.Extent = geoSource.Extent;
deGeoDataset.SpatialReference = geoSource.SpatialReference;
// 设置网络数据集的名称
// 名称可以自定义，在本例中采用通常的命名规则，
// 即在图层名称后面增加“_ND”后缀，注意不需要添加“.nd”的目录文件扩展名
IDataElement dataElement = deNetworkDataset as IDataElement;
dataElement.Name = fSourceLayer.Name + "_ND";

// 设置网络数据集的来源数据为 shapefile 格式
deNetworkDataset.NetworkType = esriNetworkDatasetType.esriNDTShapefile;
// 设置网络的高程模型，本例中的网络不提供高程模型
deNetworkDataset.ElevationModel = esriNetworkElevationModel.esriNEMNone;
// 设置网络的转弯模式，本例中的网络不提供转弯模式
deNetworkDataset.SupportsTurns = false;
// 创建网络数据集的数据源，并将其与源图层向关联；
INetworkSource edgeNetworkSource = new EdgeFeatureSourceClass();
edgeNetworkSource.Name = fSourceLayer.Name;
// 由于本例中的源图层表示了网络的边，因此设置数据源的类型为“边”
edgeNetworkSource.ElementType = esriNetworkElementType.esriNETEdge;
//设置网络数据源的连通规则，也就是定义边的矢量要素间如何连通。
// 在 ArcCatalog 的创建网络数据集操作中可以看到，网络的连通规则有三种，
// 对于已经分段化的线状图层来说，最常用的是端点连通，
```

```
// 也就是在端点处邻接的边是连通的。
// 连通规则的设置通过网络数据源的 IEdgeFeatureSource 接口完成
IEdgeFeatureSource edgeFeatureSource = edgeNetworkSource as
                                                                    IEdgeFeatureSource;

edgeFeatureSource.UsesSubtypes = false;
edgeFeatureSource.ClassConnectivityGroup = 1;
// ClassConnectivityPolicy 是关键属性，也就是前述的连通规则。
// 本例中使用端点连通的规则。
edgeFeatureSource.ClassConnectivityPolicy =
                                                                    esriNetworkEdgeConnectivityPolicy.esriNECPEndVertex;

// 将创建好的网络数据源添加到用于创建网络的 DENetworkDataset 组件对象中
// 本例仅需一个网络边的数据源，
// 对于复杂的网络还可以提供更多类型的数据源，此处不做介绍。
// DENetworkDataset 组件对象可以接纳多个数据源，因此要以数组的形式设置，
// AO 提供了 Array 组件用于数组操作。
IArray sourceArray = new ArrayClass();
sourceArray.Add(edgeNetworkSource);
deNetworkDataset.Sources = sourceArray;

// 除了数据源，创建网络时还需要设置属性，网络有多种属性，
// 因此也要通过数组来设置
IArray attributeArray = new ArrayClass();
// 为了简化，本例只设置一个属性，即网络边的运行成本，
// 这是最短路径分析所必需的属性。
// 以下定义几个设置属性操作所需的变量。
IEvaluatedNetworkAttribute evalNetAttr;
INetworkAttribute2 netAttr2;
INetworkFieldEvaluator netFieldEval;
INetworkConstantEvaluator netConstEval;
// 边的成本属性属于网络赋值属性，对应于 EvaluatedNetworkAttribute 组件，
// 因此先创建一个该组件对象。
evalNetAttr = new EvaluatedNetworkAttributeClass();
// 设置操作在其 INetworkAttribute2 接口中完成，
// 这部分设置与 ArcCatalog 的操作相对应
```

```
netAttr2 = (INetworkAttribute2)evalNetAttr;
netAttr2.Name = "METRES";
// 设置其为边的成本属性
netAttr2.UsageType = esriNetworkAttributeUsageType.esriNAUTCost;
// 设置数值类型
netAttr2.DataType = esriNetworkAttributeDataType.esriNADTDouble;
// 设置数值单位
netAttr2.Units = esriNetworkAttributeUnits.esriNAUMeters;
netAttr2.UseByDefault = false;
// 一般来说成本应根据矢量要素的属性来计算,
// NetworkFieldEvaluator 组件用于根据矢量要素字段值计算网络属性值。
netFieldEval = new NetworkFieldEvaluatorClass();
// 本例中边的成本由边的长度表达, 以下设置计算方法。
netFieldEval.SetExpression("[LENGTH]", "");
// 由于网络的边有两个方向, 因此需要分别设置正向和负向的运行成本,
// 本例中设置为相同值, 即都为边的长度。
evalNetAttr.set_Evaluator(edgeNetworkSource,
esriNetworkEdgeDirection.esriNEDAlongDigitized, (INetworkEvaluator)netFieldEval);
evalNetAttr.set_Evaluator(edgeNetworkSource,
esriNetworkEdgeDirection.esriNEDAgainstDigitized,
(INetworkEvaluator)netFieldEval);
// 网络中的其他元素(例如节点、转弯)等也需要成本属性,
// 但本例中不存在上述元素的矢量要素数据源。
// 另外, 在某些特殊的数据格式中, 线状矢量要素的长度属性不是自动创建的
// 因此可能无法根据长度字段设置边的成本属性。
// 在上述无法根据矢量要素字段设置网络元素属性的情况下,
// 需要提供一种缺省的属性赋值方案, 本例中直接将缺省值指定为 0.
// 如果要采用常数作为网络元素属性的缺省值,
// 需要使用 NetworkConstantEvaluator 组件。
netConstEval = new NetworkConstantEvaluatorClass();
// 设置缺省值为 0.
netConstEval.ConstantValue = 0;
// 依次为边、节点和转弯等网络元素的成本属性设置缺省赋值方案
evalNetAttr.set_DefaultEvaluator(esriNetworkElementType.esriNETEdge,
(INetworkEvaluator)netConstEval);
```

```

evalNetAttr.set_DefaultEvaluator(esriNetworkElementType.esriNETJunction,
(INetworkEvaluator)netConstEval);
evalNetAttr.set_DefaultEvaluator(esriNetworkElementType.esriNETTurn,
(INetworkEvaluator)netConstEval);
// 以上已经完整的创建并设置了一个网络属性，将其添加到数组中。
attributeArray.Add(evalNetAttr);
// 将其设置为创建网络数据集的 DENetworkDataset 组件对象的网络属性
deNetworkDataset.Attributes = attributeArray;
// 创建网络数据集，
//与加载网络数据集一样，创建网络数据集的步骤也要通过工作空间扩展来完成
IWorkspace pWorkSpace = (fSourceLayer.FeatureClass as IDataset).Workspace;
IWorkspaceExtensionManager      pWEM      =      pWorkSpace      as
IWorkspaceExtensionManager;
UID uid = new UID();
uid.Value = "esriGeoDatabase.NetworkDatasetWorkspaceExtension";
IWorkspaceExtension pWSExtension = pWEM.FindExtension(uid);
IDatasetContainer2 pDContainer2 = pWSExtension as IDatasetContainer2;
IDEDataset deDataset = deNetworkDataset as IDEDataset;
// DENetworkDataset 组件对象只是网络数据集的基本元素，
// 要基于这个基本元素，调用 IDatasetContainer2 接口的 CreateDataset 方法，
// 才能最终创建网络数据集。
INetworkDataset networkDataset = pDContainer2.CreateDataset(deDataset) as
INetworkDataset;
// 但是此方法创建的网络数据集仅包含基本数据，并未构建网络拓扑关系，
// 因此还需进行网络拓扑的构建，
// 这一过程通过网络数据集组件对象的 INetworkBuild 接口完成。
INetworkBuild networkBuild = networkDataset as INetworkBuild;
networkBuild.BuildNetwork(deGeoDataset.Extent);

```

#### 4. 在网络数据集中进行路径计算的基本方法。

最短路径计算是网络分析中最常用的功能之一，我们仍然可以从 ArcGIS 软件的最短路径计算操作过程中获得启发，了解利用 AO 二次开发实现最短路径计算功能的基本原理。我们以图 7.5 中的网络数据集作为基本案例，这是一个只包含了最简单必选参数的无向无约束网络数据集，当我们在 ArcMap 中进行最短路

径求解后的状态如图 7.6 所示。根据具体的操作过程和求解后的状态图，我们可以初步总结出以下特点：

- (1) 路径求解的相关设置和可视化过程是在临时添加的“路径图层”上完成的；
- (2) 路径图层上一般可以显示三类对象：停靠点、路径和障碍（点障碍、线障碍、面障碍），其中停靠点和障碍对象是用户设置的，路径对象是求解结果；
- (3) 对于路径求解来说，停靠点是必须的，障碍是可选的；
- (4) 如果进一步查看三类对象的数据结构，可以发现它们实际上都是参照矢量要素（Feature）的格式进行设计的，但是不同类对象的数据结构之间是存在差异的。如果我们把对象就当作矢量要素，那么可以说路径图层是一种混合要素图层，这与常规的 shapefiles 图层只能包含单一类型的矢量要素是有所区别的。

上述特点对于开发过程是具有启发意义的，它提示我们关注以下的实现过程：如何添加路径图层？如何添加各类对象？如何将各类对象与网络数据关联起来？如何根据对象的设置求解出结果？如何通过路径对象展现结果？

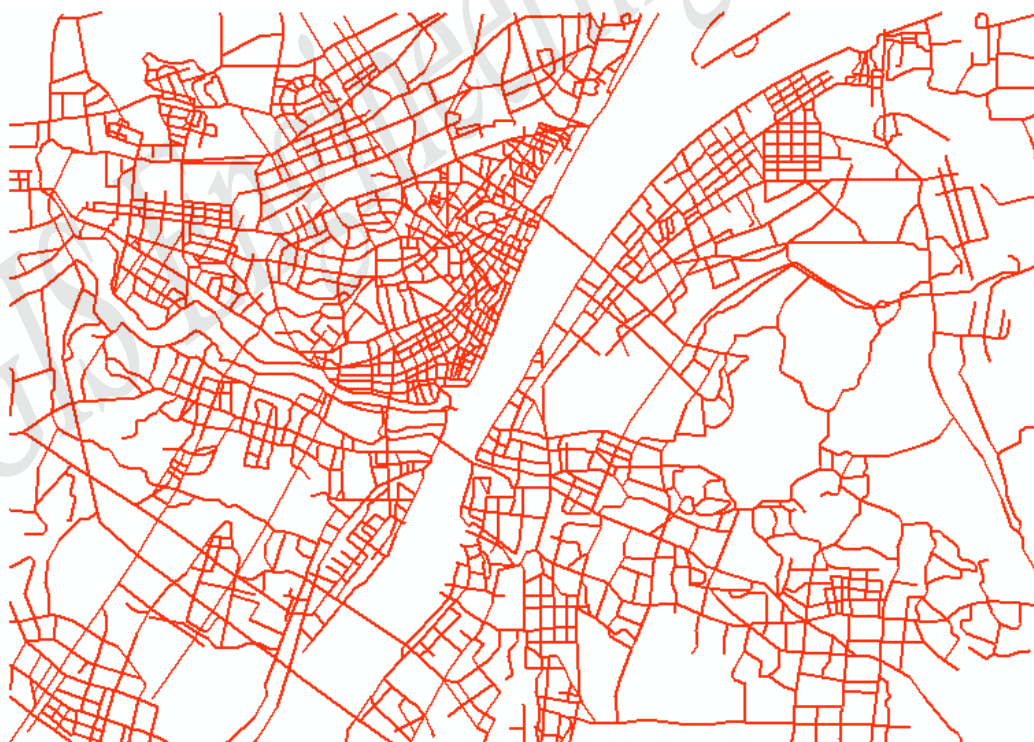


图 7.5. 路径计算实验的案例

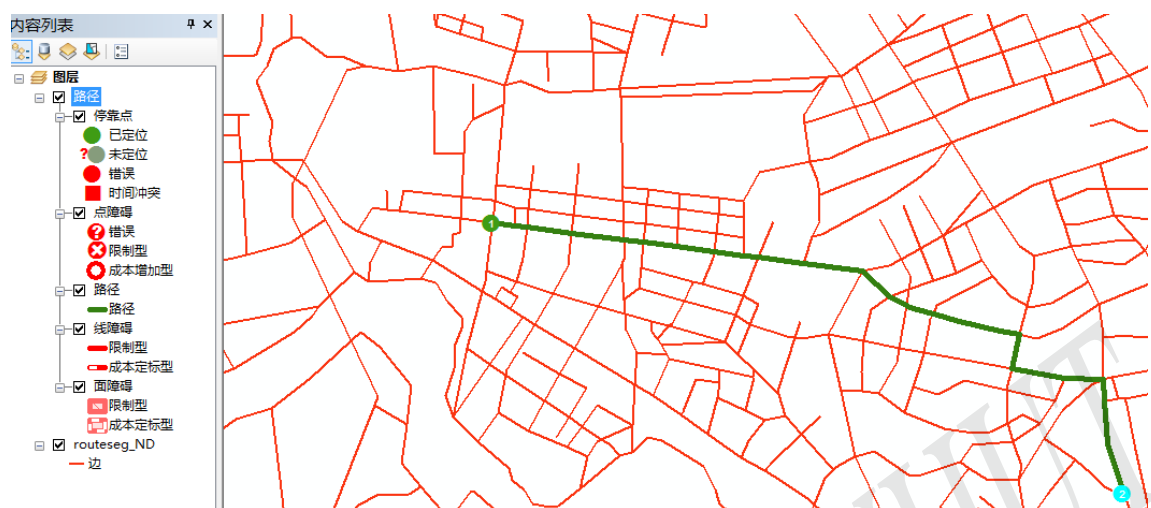


图 7.6. 路径计算实验案例的结果

#### 4.1. 网络数据集路径求解中用到的一些基本组件和接口

路径求解过程中需要用到主要组件和接口基本上都定义在 **NetworkAnalyst** 组件库中，具体说明见表 7.1.

表 7.1. 路径分析所涉及的组件和接口简介

组件类	需要用到的接口	说明
NARouteSolver	INARouteSolver, INASolver	针对各种网络分析应用， <b>NetworkAnalyst</b> 组件库中提供了各种求解器组件类，名称中都以 <b>Solver</b> 为后缀，求解器是整个分析过程的起点，所有相关对象都由它生成。 <b>NARouteSolver</b> 顾名思义是面向路径分析的求解器。
NALayer	INALayer	网络分析图层组件。路径图层是一种特殊的网络分析图层。由 <b>NARouteSolver</b> 可以生成 <b>NALayer</b>
NAContext	INAContext, INAContextEdit	网络分析环境组件。与其他类型图层一样，网络分析图层实际上只提供了一个显示和管理介质，分析过程中的所有相关参

		数和其他数据都由网络分析环境组件来管理，两者的关系可以类比于 <b>FeatureLayer</b> 和 <b>FeatureClass</b> 的关系。这是整个网络分析中最重要的组件类。
<b>NAClass</b>	<b>INAClass, IFeatureClass</b>	网络分析对象类组件类，前面分析过，路径分析图层上的各种对象的数据结构十分类似于矢量要素，因此网络分析组件库中就参照 <b>FeatureClass</b> 的特点定义了 <b>NAClass</b> ，它相当于网络分析对象的数据表结构，更有趣的是这个组件类也实现了 <b>IFeatureClass</b> 结构，说明在 AO 的设计者眼中，它们的确是脱胎于 <b>FeatureClass</b> 的。
<b>NAClassLoader</b>	<b>INAClassLoader</b>	网络分析对象类的加载器组件。用于自动将各种感兴趣的地图元素作为相应的分析对象加载到网络分析图层中。

#### 4.2. 创建路径图层的方法

我们只介绍创建最简化的路径图层的方法，参考 **CODE7.6**，更复杂的参数设置请参考帮助文档。

##### CODE7.6 创建路径图层

```
//创建一个路径求解器组建对象
INARouteSolver naRouteSolver = new NARouteSolverClass();
INASolver naSolver = naRouteSolver as INASolver;
//获取目标网络数据集的数据元素，用于后续的网络分析环境生成操作
// networkDataset 变量表示当前打开的网络数据集
```

```

IDatasetComponent datasetComponent = networkDataset as IDatasetComponent;
IDENetworkDataset deNetworkDataset = datasetComponent.DataElement as
IDENetworkDataset;
//由路径求解器对象生成网络分析环境并将其与网络数据集绑定
INaContext naContext;
naContext = naSolver.CreateContext(deNetworkDataset, layerName);
INaContextEdit naContextEdit = naContext as INaContextEdit;
naContextEdit.Bind(networkDataset, new GPMessagesClass());
//由路径求解器对象根据网络分析环境生成路径图层
INaLayer naLayer;
naLayer = naSolver.CreateLayer(naContext);
// layerName 变量表示路径图层的名称，由开发者自定义
(naLayer as ILayer).Name = layerName;
//设置相关的求解参数，参数的意义参见帮助文档
naRouteSolver.FindBestSequence = true;
naRouteSolver.PreserveFirstStop = true;
naRouteSolver.PreserveLastStop = false;
naRouteSolver.UseTimeWindows = false;
// OutputLines 参数指定了求解结果的输出形式，这里选择以原始形状进行输出
//其他输出形式参见帮助文档。
naRouteSolver.OutputLines = esriNAOutputLineType.esriNAOutputLineTrueShape;
//参数设置后更新分析环境
naSolver.UpdateContext(naContext, deNetworkDataset, new GPMessagesClass());
//将路径图层加入到地图中
axMapControl1.AddLayer(naLayer as ILayer);

```

#### 4.3. 设置停靠点（stop）对象的方法

为了简化内容，我们仅以停靠点为例介绍分析对象的设置方法，设置一个分析对象相当于在路径图层中添加一个矢量要素，其数据格式由相应的 **NAClass** 对象定义，我们可以入 **ArcMap** 的操作中那样交互式的手工设置，但是更多的实际情况下是根据已有的地图数据中的某些特征位置进行设置（例如在公交查询中基于车站位置进行设置），这些已有位置往往保持在其他 **shapefiles** 中。我们以此为例介绍设置方法，参考 **CODE7.7**：

**CODE7.7** 设置停靠点的基本方法



```

//获取停靠点（Stops）对象类
INAClass stopsNAClass = naContext.NAClasses.get_ItemByName("Stops") as
INAClass;
// 创建一个加载器对象用于分析对象的自动加载
INAClassLoader naLoader = new NAClassLoaderClass();
naLoader.Locator = naContext.Locator;
naLoader.NAClass = stopsNAClass;
int rowsInCursor = 0;
int rowsLocated = 0;
// 创建一个查询条件对象用于设置停靠点的查找条件
IQueryFilter filter = new QueryFilterClass();
//设置查找条件用于在包含特征位置的矢量要素类中搜索用于最短路径分析的
停靠点
// 例如: filter.WhereClause = "FID = 384 or FID = 1021";
// 加载停靠点, 其中 inputStopsFClass 表示包含特征位置的矢量要素类,
// 需要由开发者另行指定,
// 另外可多次调用 Load 方法以在路径图层上累积设置多个停靠点。
naLoader.Load(inputStopsFClass.Search(filter, true) as ICursor,
              new CancelTrackerClass(), ref rowsInCursor, ref rowsLocated);

```

#### 4.4. 路径求解和结果显示的方法

求解过程实际上就是调用求解器组件中的 Solver 方法, 结果显示已经在求解器的 OutputLines 属性中设定了, 参考代码 CODE7.8

#### CODE7.8 执行路径求解过程

```

INASolver naSolver = naContext.Solver;
naSolver.Solve(naContext, new GPMessagesClass(), new CancelTrackerClass());

```

## 六、思考题

本实验为综合设计性实验, 请结合本指导书中的基本原理和案例介绍, 自行设计并实现程序。