实验二 AO 组件库客户化

1 背景知识

由于 ArcGIS 是完全 COM 化的，对于需要进行 ArcGIS 结构定制和功能扩展的高级开发人员来说，极具吸引力。任何与 COM 兼容的编程语言，如 C#、Visual C++、Visual Basic、Delphi 或 Visual J++都能用来定制和扩展 ArcGIS。

编写 ArcGIS 扩展插件，可以完成以下任务：(1) Command，即菜单按钮，通过实现 ICommand、ITool、IToolCommand 接口来创建；(2) Edit Task，与 ArcMap Editor 协同工作的组件，需要实现 IEditTask 接口；(3) Table of Contents，类似左侧的数据和图层视图的小窗口，通过实现 IContentsView 接口来创建；(4) Class extension，自定义要素（feature），即有自己属性和规则的空间要素，例如红绿灯对象、电线、电闸等，需要实现 IClassExtension 等接口。

ArcGIS 插件架构的核心在于应用程序定义插件遵循的接口，然后由自定义组件来实现这个接口（如图 2-1 所示），其中有几个关键点：

（1）主应用程序如何知道要加载插件。对于 ArcGIS 是使用注册表，和微软 Office 类似，在注册表中建立一个“Component categories”的条目列表，而组件要被 ArcGIS 加载，就需要注册后，在这里添加一个条目。

（2）插件的初始化。初始化包括两方面，一方面，主程序要定义一个引用变量，类型为插件所实现的接口，然后使用该引用来创建一个插件对象；另一方面，主程序触发插件的 OnCreate 初始化事件，传入主程序需要暴露给插件的引用（hook），根据传来的变量，初始化插件的运行环境，插件的初始化过程，就是插件和主程序通讯桥梁的过程，这个桥梁，对于 AO 组件库，就是 IHookHelper 接口。

（3）插件和应用程序之间的通讯。对于主应用程序，是通过创建插件对象，获得当前插件的实例，然后控制它，程序的一些状态变化，可以通过定义事件，然后在插件代码中响应这些事件来完成，如果是插件是一个按钮，那么初始化插件的时候，创建了按钮对象，并指定按钮的 OnClick 事件由插件的相应方法来处理。对于插件，则由于在初始化中通过初始化 OnCreate 事件的参数得到了主应用程序的资源对象实例 hook，因此就可以使用它来操纵主应用程序资源。

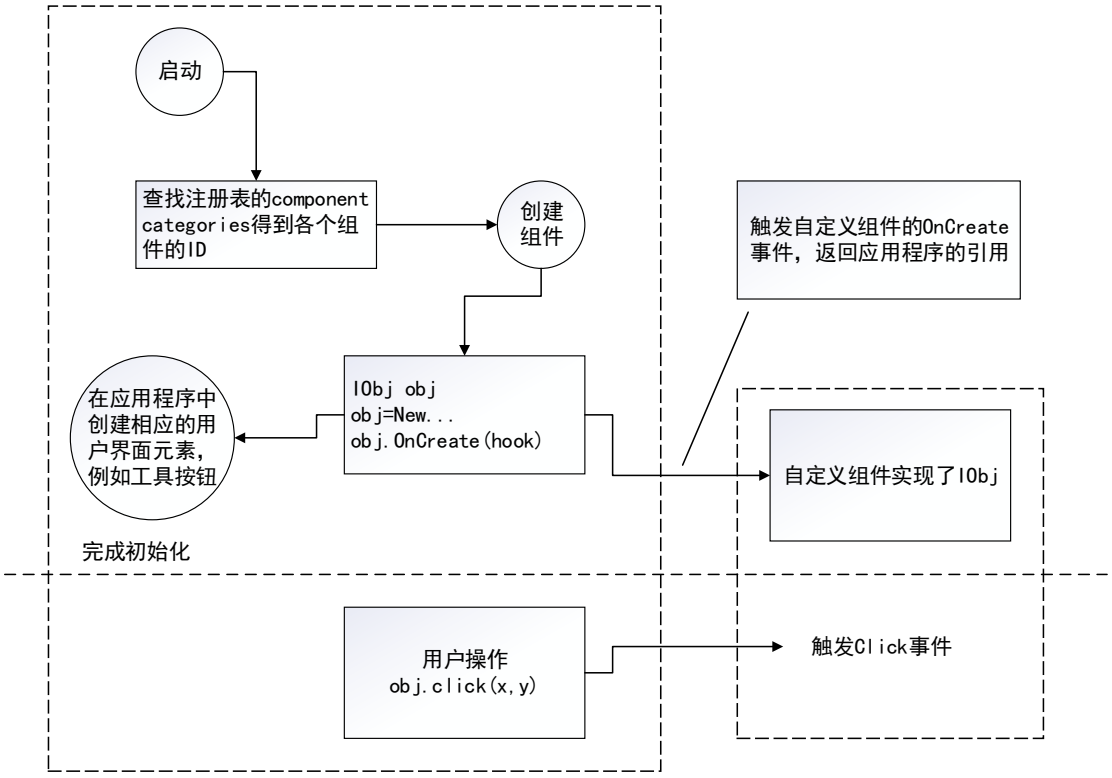

图 2-1 ArcGIS 的插件架构

2 演示实例

AO 客户化插件开发的基本步骤如下：（1）新建工程，添加 AO 的引用；（2）实现需要的接口，添加具体的代码；（3）编译为 DLL，注册该 COM 组件，并在 ArcGIS 的 Component categories 注册；（4）测试和调试。

本例要完成的功能是通过 ICommand 接口来创建定制的按钮 Command，实现过滤显示要素图层中的要素 Features，即根据指定的属性条件显示图层中的要素。先得到要进行过滤显示的要素图层 FeatureLayer, 再更新要素图层所实现的 IFeatureLayerDefinition 接口的 DefinitionExpression 属性来设置查询条件来得到要显示的要素。

（1)以管理员权限运行 Visual Studio 新建项目，选择【Visual C#】→【ArcGIS】→【Extending ArcObjects】→【Class Library（Engine)】模板，项目名称命名为“ArcEngineClassLibrary”，解决方案命名为“Test2”，如图 2-2 所示。

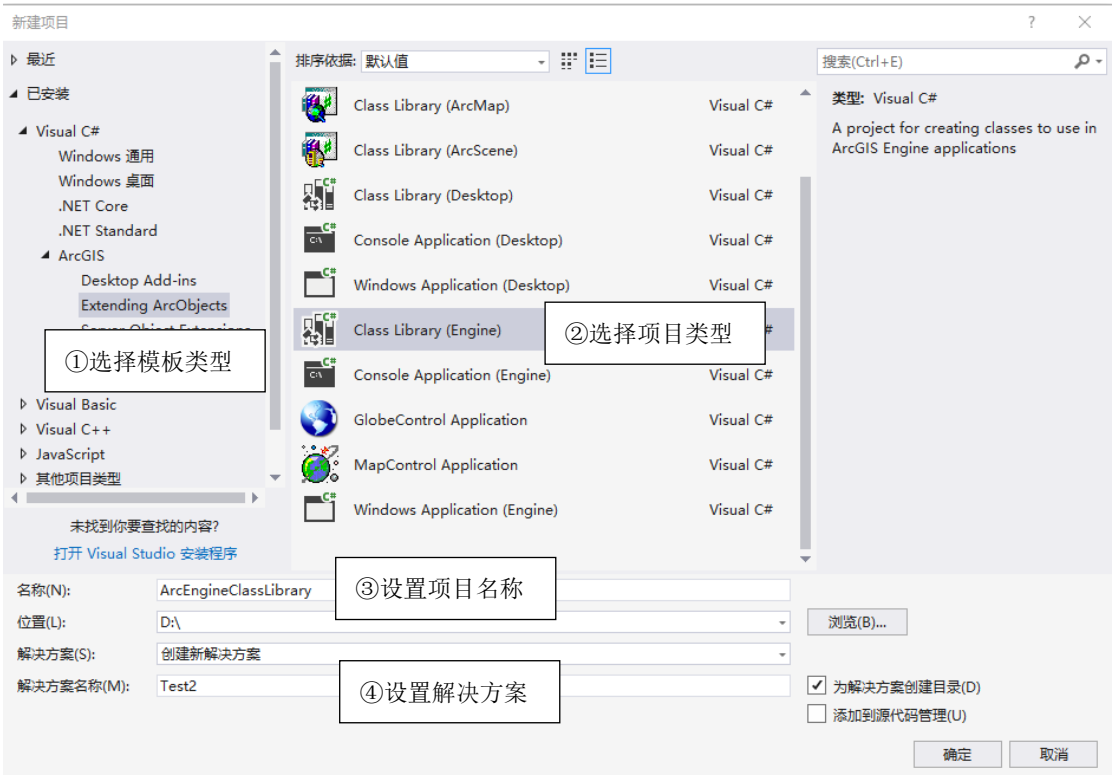

图 2-2 新建 ArcEngine 客户化类库

（2）添加 ArcEngineCore 核心库的引用，如图 2-3 所示，

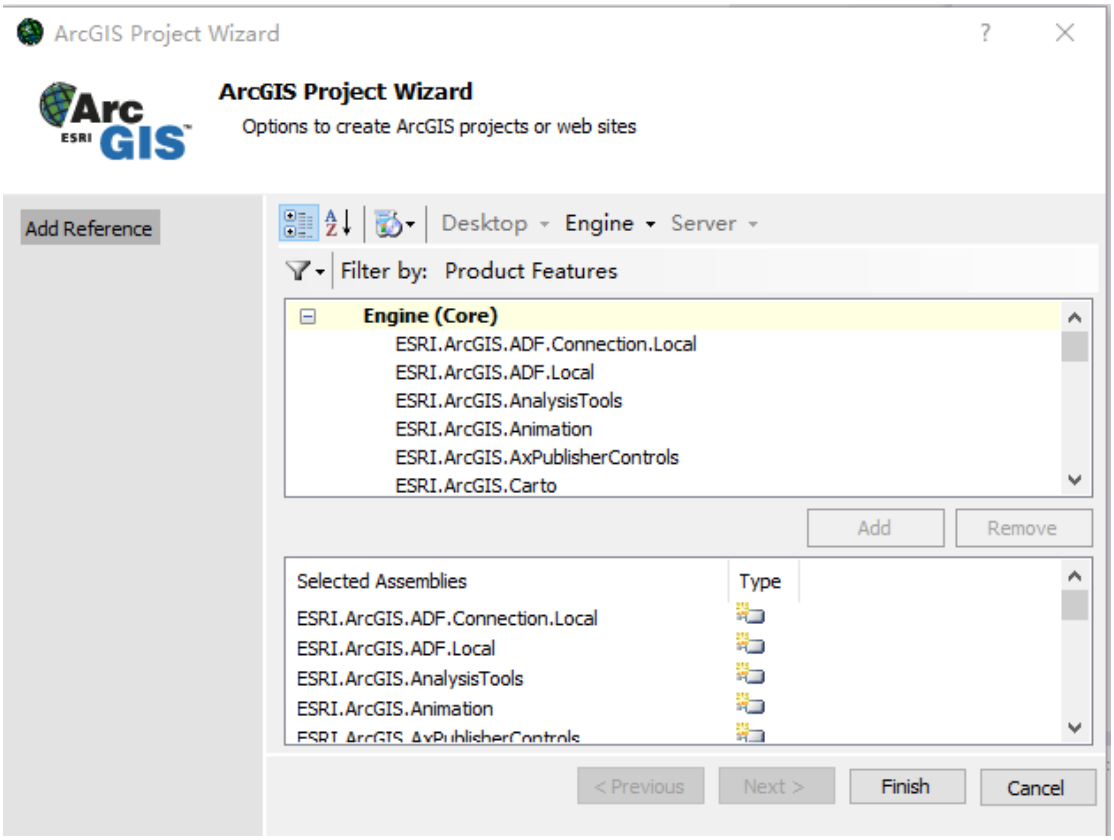

图 2-3 添加 ArcEngine 核心库的引用

（3）解决方案资源管理器选中项目“ArcEngineClassLibrary”，右键选择【添加】→【新建项】，选择【Visual C#】→【ArcGIS】 →【Extending ArcObjects】→【Base Command】模板，类命名为“CmdFilter”，如图 2-4 所示。

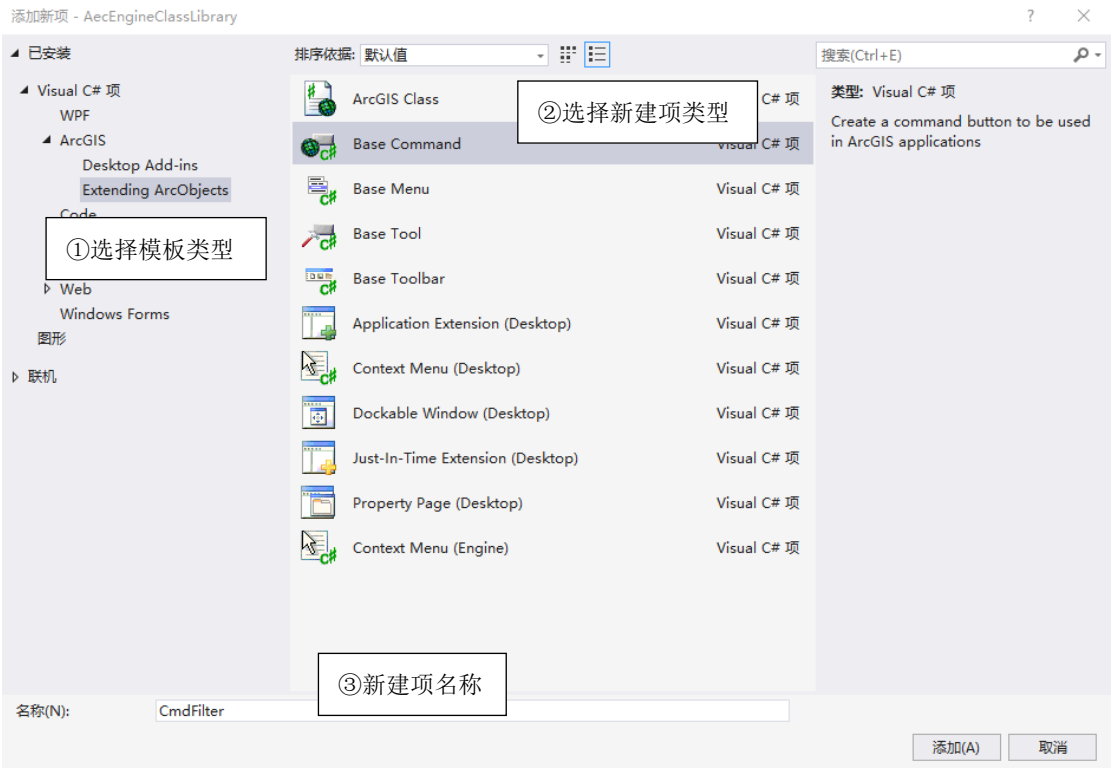

图 2-4 新建按钮命令 Command 类

（4）在弹出的【ArcGIS New Item Wizard Options】新建项向导选项中选择“ArcMap, MapControl or PageLayoutControl Command”，如图 2-5 所示。

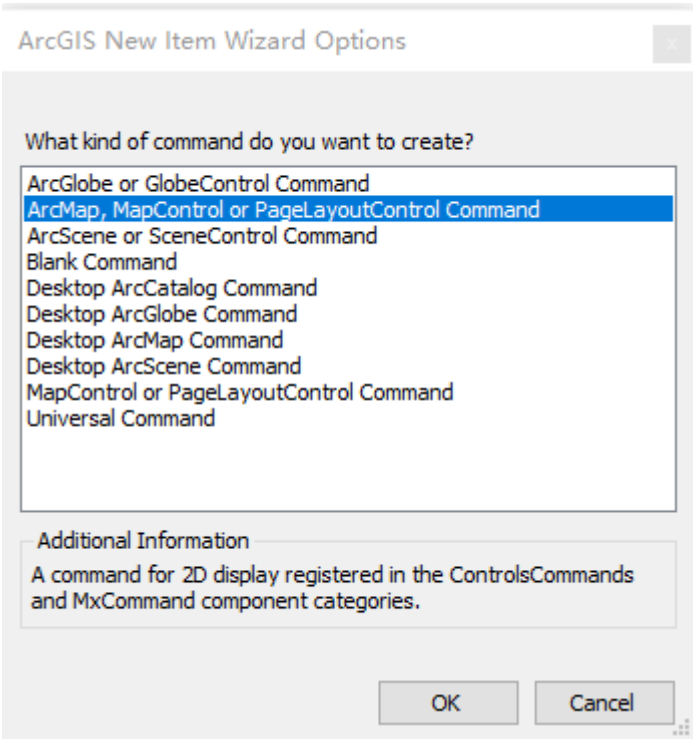

图 2-5 ArcGIS 新建项类型选项

（5）修改“CmdFilter”类的构造函数中关于命令按钮的分类、命名等字符串。ICommand 接口包括 caption、 name、 category、 bitmap、 message（状态栏 StatusBar 的提示信息）、 tooltip（微帮助）、 help context id 、help file、enabled 以及 checked 等十个属性和 OnCreate、OnClick 两个事件。

CmdFilter. cs（节选）	功能：命令按钮的分类、命名等

```
// TODO: Define values for the public properties
base. m_category = ″CSharpTest″; //localizable text
base. m_caption = ″CommandFilter″; //localizable text
base. m_message = ″This should work in″+
                  ″ArcMap/MapControl/PageLayoutControl″; //localizable text
base. m_toolTip = ″CommandFilter″; //localizable text
base. m_name = ″CSharpTest_CommandFilter″;
```

（6）新建一个 Windows 窗体用于选择图层和设置字段条件，并进行属性条件过滤，窗体类名称为“FrmFilter”，如图 2-6 所示。

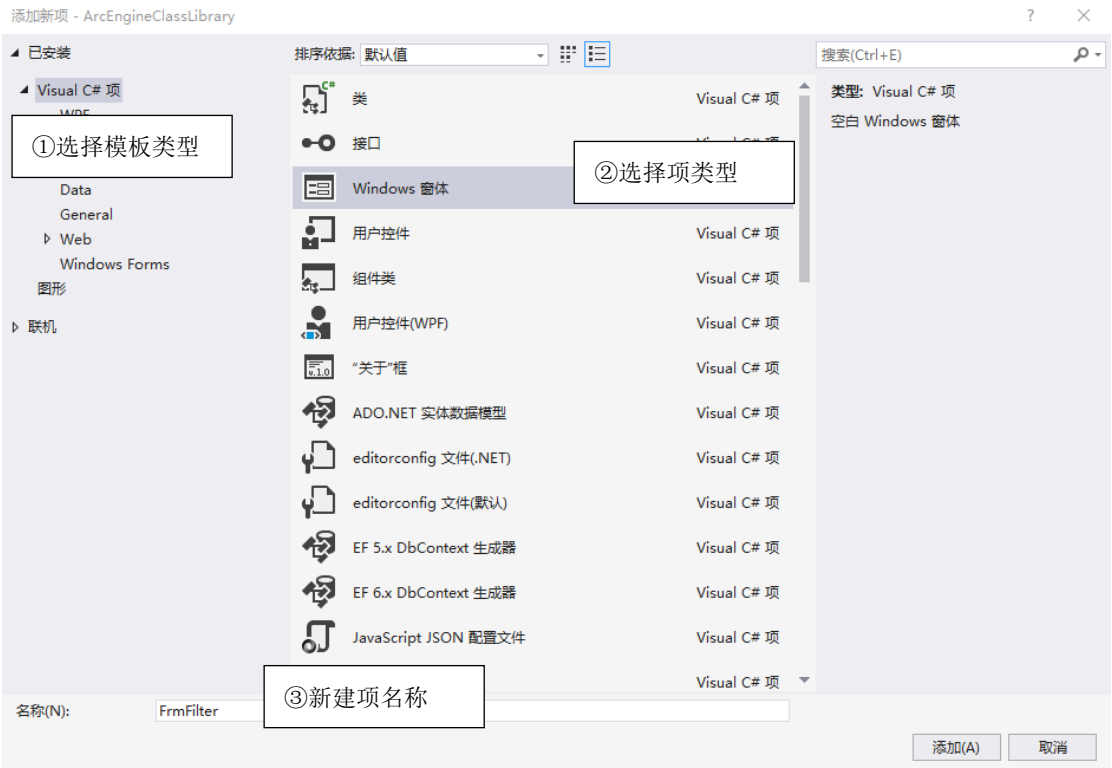

图 2-6 新建属性过滤窗口

（7）为类“FrmFilter”添加 IHookHelper 接口类型的私有成员变量 m_hookhelper，在类的构造函数中将该引用变量指向具体的对象，并在 FrmFilter. cs 文件开始部分中添加引用：

```
using ESRI. ArcGIS. Controls;
```

FrmFilter. cs（节选）	功能：定义 IHookHelper 接口获取主应用程序资源

```
public partial class FrmFilter : Form
{
    private IHookHelper m_hookhelper = null;
    public FrmFilter(IHookHelper hook)
    {
        m_hookhelper = hook;
        InitializeComponent();
    }
}
```

（8）从 ICommand 接口的 OnCreate 事件中获取的 ArcMap 的主应用程序资源实例必须用按钮类的一个成员变来量保存，以便在按钮类的其它事件或方法中(或者其它窗体的事件中)使用。OnCreate 事件传入的参数 hook 是一个 Object 类型引用，要传递给 IHookHelper 接口类型的引用变量 m_hookHelper 的 Hook 属性，通过 m_hookHelper 就可以得到 ActiveView、FocusMap、PageLayout 等组件对象。在 OnClick 事件中写入相关代码，按下按钮时弹出 Windows 窗体调用属性过滤功能。

CmdFilter. cs（节选）	功能：按钮单击弹出“FrmFilter”窗体

```
public override void OnClick()
{
    // TODO: Add CmdFilter. OnClick implementation
    FrmFilter frmFilter = new FrmFilter(m_hookHelper);
    frmFilter. Show();
}
```

（9）修改项目属性，点击【浏览】找到 ArcGIS 安装目录的 ArcMap. exe 可执行文件，选择用 ArcMap 来调试插件，如图 2-7 所示。

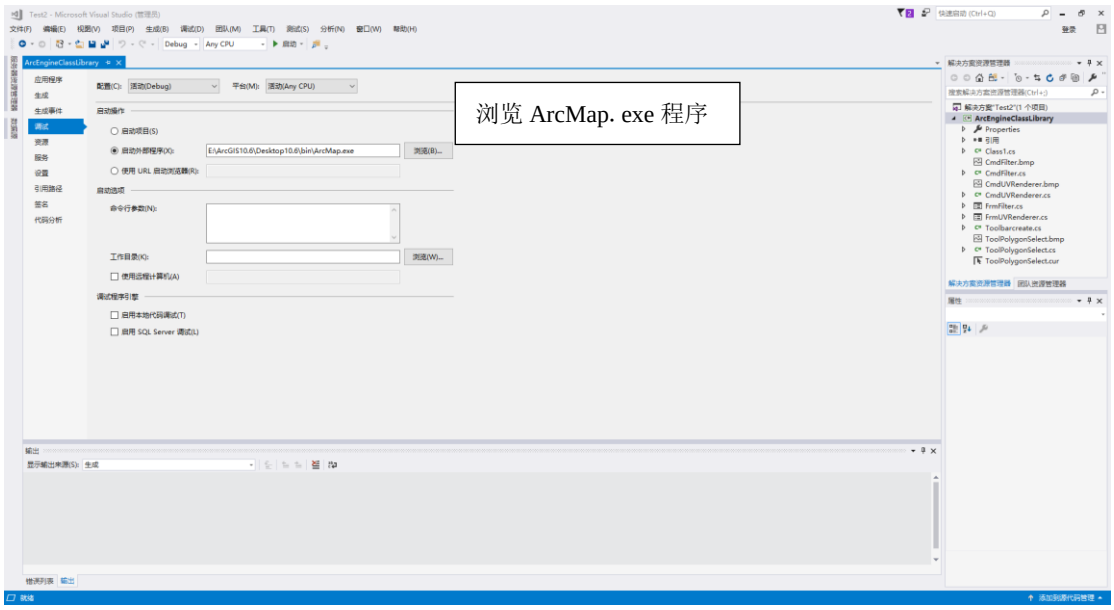

图 2-7 启动外部程序 ArcMap.exe 来调试项目

（10）开发好一个客户化组件后，便可根据实际需要，在 ArcMap 环境下加载这个客户化组件，其一般步骤是：①在 Customize 对话框中，选择【Toolbars】或【Commands】选项卡，然后点击【Add From File】；②如果加载的是【Commands】，可将其拖置于任何工具条上；③如果加载的是【ToolBars】，则可在 ArcMap 中显示。

代码编译通过后（**注意**：在每次清除或编译项目文件过程中，需要注销或重注册组件，因此应先关闭 ArcMap. exe 应用程序），点击菜单栏【调试】→【开始执行】，打开 ArcMap 菜单栏的【自定义】→【自定义模式】，如图 2-8 所示。

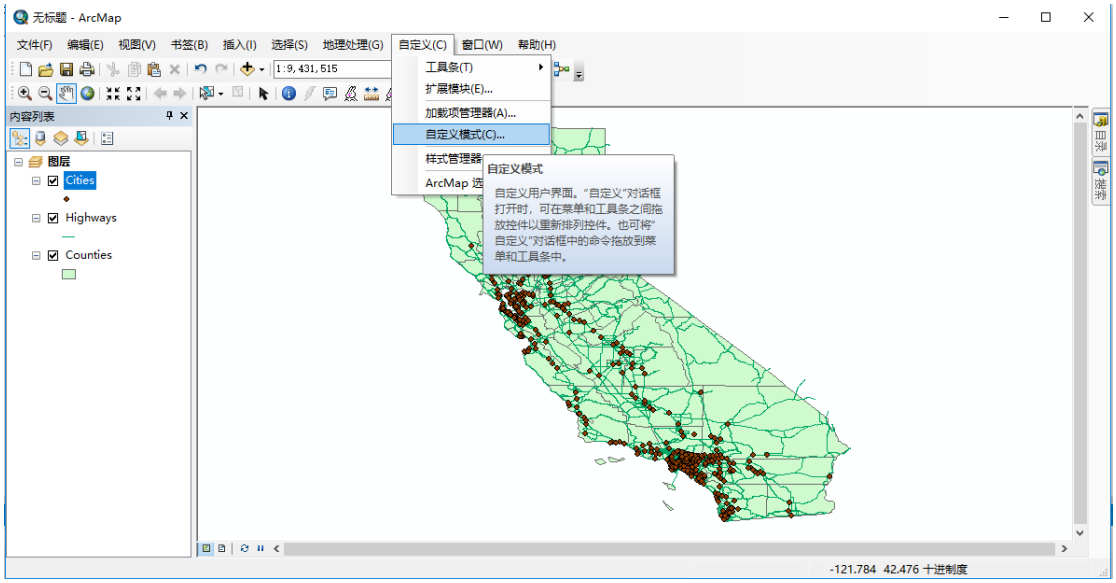

图 2-8 开启自定义模式

点击对话框下方的【Add From File】按钮，找到新开发的命令按钮的 tlb 类型库文件，将所编写的命令按钮导入到 ArcGIS，如图 2-9 所示。

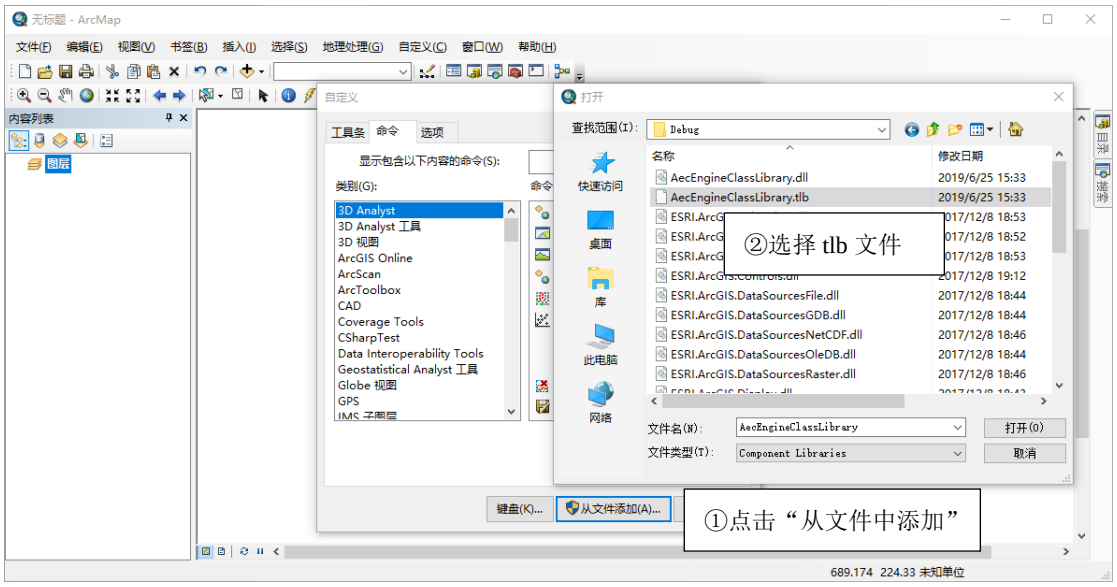

图 2-9 导入客户化组件到 ArcMap 中

（11）将命令按钮拖放到 ArcMap 的工具条上，点击客户化插件生成的“CmdFilter”按钮图标，然后弹出了一个空白的“FrmFilter”窗体，如图 2-10 所示。

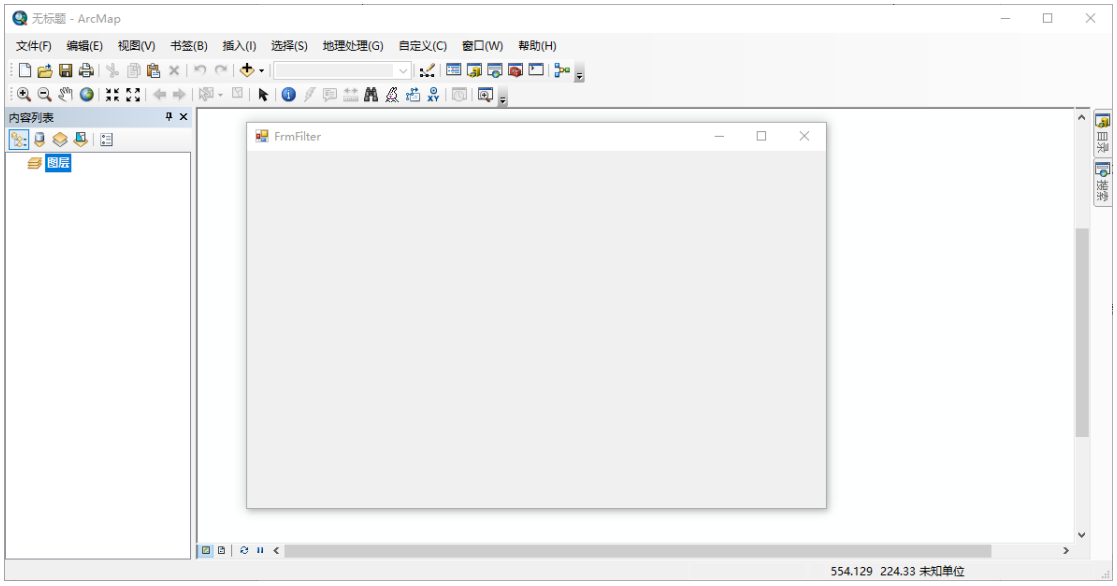

图 2-10 按钮弹出窗体运行效果

（12）下面要实现属性过滤窗体的功能，首先关闭 ArcMap，从 Visual Studio 工具箱中添加以下 Windows 窗体控件到窗体“FrmFilter”上，然后设置各控件的 Text 属性和 Name 属性，如图 2-11 所示。

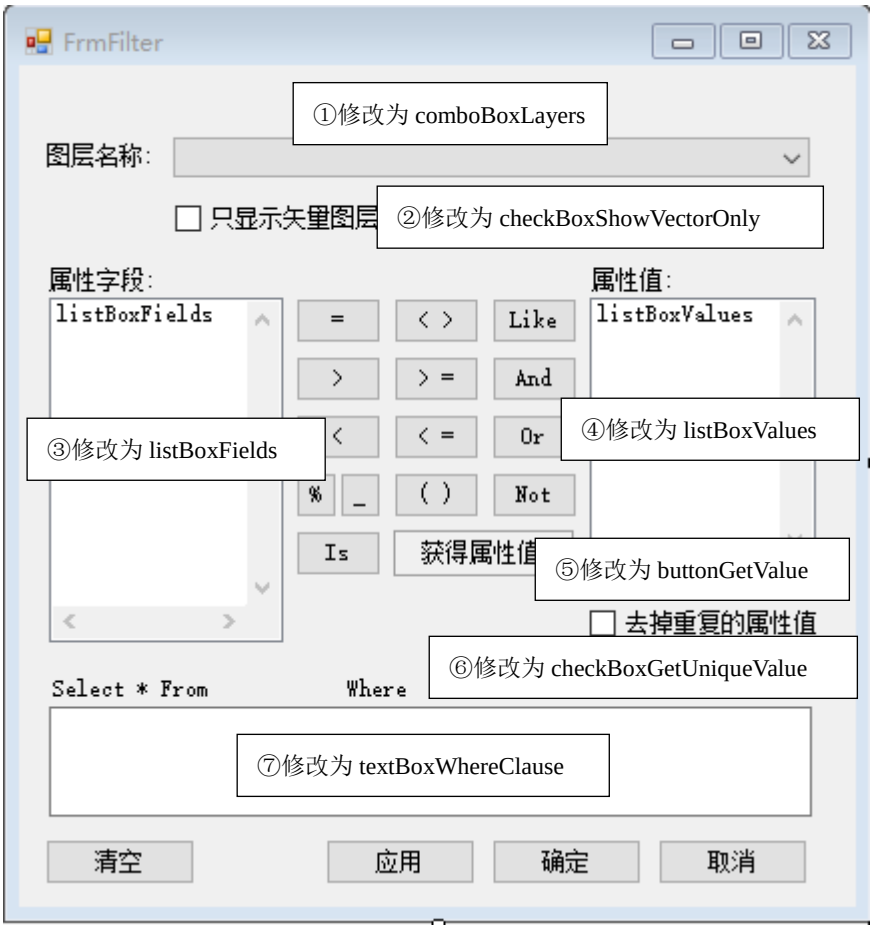

图 2-11 修改各控件的 Name 属性

（13）双击“FrmFilter”的主窗体空白处，添加窗体加载事件的响应函数 FrmFilter_Load()；再添加私有成员函数 AddAllLayerstoComboBox()的定义；双击“只显示矢量图层”复选框控件，添加复选框的选择事件响应函数。在 FrmFilter. cs 开头添加引用：

```
using ESRI. ArcGIS. Carto;
```

FrmFilter. cs（节选）	功能：窗体加载

```
private void FrmFilter_Load(object sender, EventArgs e)
{
    AddAllLayerstoComboBox(comboBoxLayers);
    if (comboBoxLayers. Items. Count != 0)
    {
        comboBoxLayers. SelectedIndex = 0;
        buttonOk. Enabled = true;
        buttonClear. Enabled = true;
        buttonApply. Enabled = true;
    }
}
```

```
//只添加当前地图中的所有图层到组合框中
private void AddAllLayerstoComboBox(ComboBox combox)
{
    try
    {
        combox.Items.Clear();
        int pLayerCount = m_hookhelper.FocusMap.LayerCount;
        if (pLayerCount > 0)
        {
            combox.Enabled = true;//组合框可用
            checkBoxShowVectorOnly.Enabled = true;//复选框可用
            for (int i = 0; i <= pLayerCount - 1; i++)
            {
                if (checkBoxShowVectorOnly.Checked)
                {
                    if (m_hookhelper.FocusMap.get_Layer(i) is
                        IFeatureLayer)
                        //只添加矢量图层
                        combox.Items.Add(
                            m_hookhelper.FocusMap.get_Layer(i).Name);
                }
                else
                    combox.Items.Add(
                        m_hookhelper.FocusMap.get_Layer(i).Name);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}
//当只显示矢量数据的复选框改变的时候，则要重新加载图层列表
private void checkBoxShowVectorOnly_CheckedChanged(object sender,
                                                   EventArgs e)
{
    AddAllLayerstoComboBox(comboBoxLayers);
    if (comboBoxLayers.Items.Count != 0)
        comboBoxLayers.SelectedIndex = 0;
    listBoxFields.Items.Clear();
}
```

（14）双击 comboBoxLayers 图层选择组合框，添加图层选择事件响应函数。

FrmFilter. cs（节选）	功能：图层选择事件响应函数

```
private void comboBoxLayers_SelectedIndexChanged(object sender, EventArgs e)
{
    listBoxFields.Items.Clear();
    listBoxValues.Items.Clear();
    string strSelectedLayerName = comboBoxLayers.Text;
    IFeatureLayer pFeatureLayer;
    IDisplayTable pDisPlayTable;
    try
    {
        for (int i = 0; i <= m_hookhelper.FocusMap.LayerCount - 1; i++)
        {
            if (m_hookhelper.FocusMap.get_Layer(i).Name ==
                                                    strSelectedLayerName)
            {
                if (m_hookhelper.FocusMap.get_Layer(i) is IFeatureLayer)
                {
                    pFeatureLayer = m_hookhelper.FocusMap.get_Layer(i)
                                                        as IFeatureLayer;
                    //获得当前选择的图层
                    pDisPlayTable = pFeatureLayer as IDisplayTable;
                    //根据选择图层更新字段列表
                     for (int j = 0; j <=
                          pDisPlayTable.DisplayTable.Fields.FieldCount - 1;
                          j++)
                    {
                    listBoxFields.Items.Add(
                    pDisPlayTable.DisplayTable.Fields.get_Field(j).Name);
                    }
                }
                else
                { MessageBox.Show("您选择的图层不能进行属性查询！"+
                            "请重新选择"); break; }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
```

```
}
```

（15）属性窗口中点击添加 listBoxFields 的 DoubleClick 事件；以及添加 listBoxValues 的 DoubleClick 事件，如图 2-12 所示，并添加相应的事件响应函数代码。

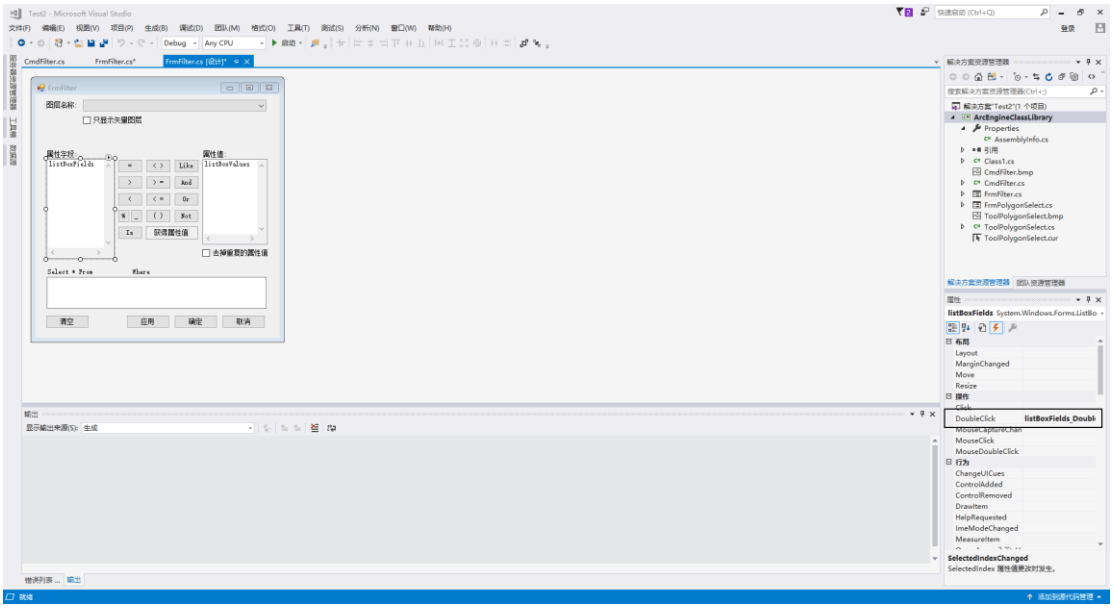

图 2-12 添加 listBoxFields 的双击事件响应函数

FrmFilter. cs（节选）	功能：双击字段和属性值时添加到 Where 子句

```
private void listBoxFields_DoubleClick(object sender, EventArgs e)
{
    textBoxWhereClause.SelectedText =
                                listBoxFields.SelectedItem.ToString() + " ";
}

private void listBoxValues_DoubleClick(object sender, EventArgs e)
{
    textBoxWhereClause.SelectedText = " " +
                                listBoxValues.SelectedItem.ToString();
}
```

（16）双击“获取属性值”按钮，加入按钮点击事件响应函数代码，添加私有成员函数 GetLayerByName()来由名字获取图层。在 FrmFilter. cs 文件开头导入引用：

```
using ESRI.ArcGIS.Geodatabase;
```

FrmFilter. cs（节选）	功能：获取属性字段的唯一值

```
private void buttonGetValue_Click(object sender, EventArgs e)
```

```
{
    if (listBoxFields.Text == "")
        MessageBox.Show("请选择一个属性字段！");
    else
    {
        try
        {
            //这个名字是选中的属性字段的名称
            string strSelectedFieldName = listBoxFields.Text;
            listBoxValues.Items.Clear();
            label1.Text = "";
            IFeatureCursor pFeatureCursor;
            IFeatureClass pFeatureClass;
            IFeature pFeature;
            if (strSelectedFieldName != null)
            {
                pFeatureClass = (GetLayerByName(comboBoxLayers.Text)
                                        as IFeatureLayer).FeatureClass;
                pFeatureCursor = pFeatureClass.Search(null, true);
                pFeature = pFeatureCursor.NextFeature();
                int index = pFeatureClass.FindField(strSelectedFieldName);
                while (pFeature != null)
                {
                    //获取当前要素pFeature的第index个字段的属性值
                    string strValue = pFeature.get_Value(index).ToString();
                    //如果需要去掉重复的值
                    if (checkBoxGetUniqueValue.Checked)
                    {
                        //如果属性值是字符型，则添加单引号''，
                        //方便后面WhereClause格式设置
                        if (pFeature.Fields.get_Field(index).Type ==
                            esriFieldType.esriFieldTypeString)
                            strValue = "'" + strValue + "'";
                        if (listBoxValues.FindStringExact(strValue) ==
                            ListBox.NoMatches)
                        {
                            //将字段唯一值添加到listBoxValues组合框
                            listBoxValues.Items.Add(strValue);
                        }
                    }
                    else//否则添加所有的值，不考虑有没有重复
                    {
                        if (pFeature.Fields.get_Field(index).Type ==
                                        esriFieldType.esriFieldTypeString)
```

```
                                strValue = "'" + strValue + "'";
                            listBoxValues.Items.Add(strValue);
                        }
                        //获取下一个要素
                        pFeature = pFeatureCursor.NextFeature();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                return;
            }
        }
    }
    //由名字获取图层
    private ILayer GetLayerByName(string strLayerName)
    {
        ILayer pLayer = null;
        for (int i = 0; i <= m_hookhelper.FocusMap.LayerCount - 1; i++)
        {
            if (strLayerName == m_hookhelper.FocusMap.get_Layer(i).Name)
            { pLayer = m_hookhelper.FocusMap.get_Layer(i); break; }
        }
        return pLayer;
    }
```

（17）双击“应用”、“确定”、“清空”和“取消”按钮，分别添加各按钮点击事件响应函数代码；添加属性过滤功能的私有成员函数 PerformAttributeFilter()；双击添加“=”运算符和其他运算符按钮，分别添加各按钮点击事件响应函数代码。

FrmFilter.cs（节选）	功能：WhereClause 语句设置及查询过滤

```
private void buttonApply_Click(object sender, EventArgs e)
{
    if (textBoxWhereClause.Text == "")
    {
        MessageBox.Show("请生成查询语句！");
        return;
    }
    //通过位置查询窗口最小化
    this.WindowState = FormWindowState.Minimized;
    PerformAttributeFilter();
```

```
        this. WindowState = FormWindowState. Normal;
    }
    private void buttonOk_Click(object sender, EventArgs e)
    {
        PerformAttributeFilter();
        this. Dispose();
    }
    //进行查询过滤实现条件显示要素图层
    private void PerformAttributeFilter()
    {
        try
        {
            IFeatureLayer pFeatureLayer;
            pFeatureLayer = GetLayerByName(comboBoxLayers.
                                SelectedItem. ToString()) as IFeatureLayer;
            IFeatureLayerDefinition fLyrDef = pFeatureLayer as
                                 IFeatureLayerDefinition;
            fLyrDef. DefinitionExpression = textBoxWhereClause. Text;
            m_hookhelper. ActiveView. Refresh();
        }
        catch (Exception ex)
        {
            MessageBox. Show("您的查询语句可能有误,请检查 | " +
                              ex. Message);
            return;
        }
    }
    private void buttonClear_Click(object sender, EventArgs e)
    {
        textBoxWhereClause. Clear();
    }
    private void buttonCancel_Click(object sender, EventArgs e)
    {
        this. Dispose();
    }
    //其他运算符的按钮点击事件响应函数代码类似
    private void buttonEqual_Click(object sender, EventArgs e)
    {
        textBoxWhereClause. SelectedText = " = ";
```

```
}
```

在 COM 中，一个接口可以被多个类实现，一个类也实现多个接口。接口定义了方法，类必须实现接口中定义的方法，一个接口只能使用自己内部定义的方法。因为一个类可以实现多个接口，每个接口只可以访问自己定义的方法，如果要使用定义在类实现的其它接口中的方法就需要将这个接口“切换”到其它接口，也就是进行接口的转换，如图 2-13 所示。本例中 FeatureLayer 类实现的 IFeatureLayer 和 IFeatureLayerDefinition 两个接口之间的转换就是根据这个原理。

功能：接口转换示例

```
//通过IDrive接口创建一个新的RaceCar对象
IDrive  pCar =  null;
pCar=new  RaceCar();
pCar. Accelerate();
//交换接口
IRace  pRace =  null;
pRace=pCar;
pRace. PitStop();
```

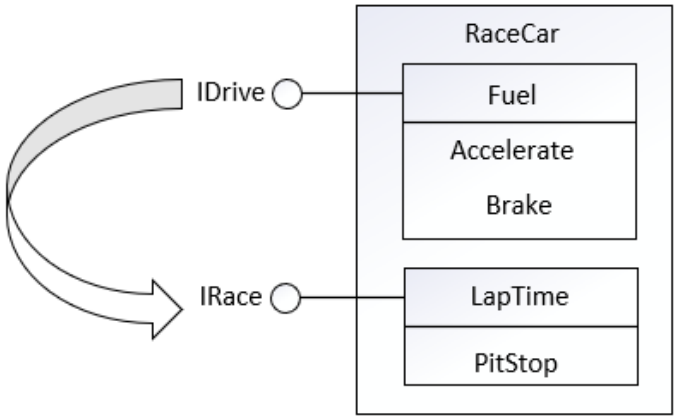

图 2-13 接口访问规则

（18）点击【生成】→【重新生成解决方案】，然后点击【调试】→【开始执行】。加载地图文档，点击“CmdFilter”按钮。选择需要过滤的图层，点击获得属性值，并输入属性查询条件，如图 2-14 所示。

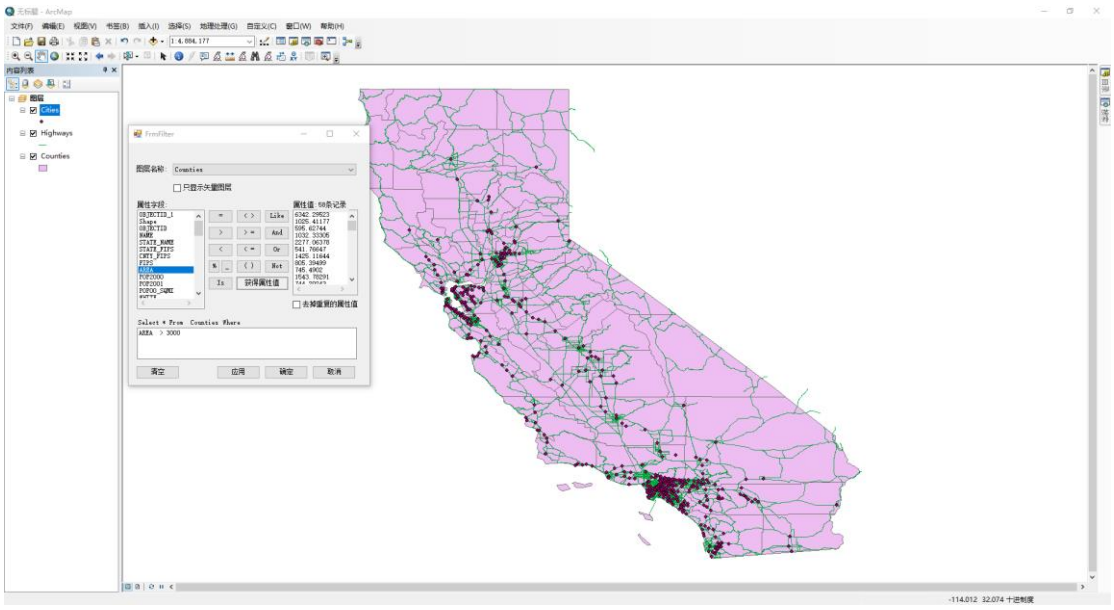

图 2-14 输入属性过滤的字段条件

点击应用，查看属性过滤功能运行效果，如图 2-15 所示。

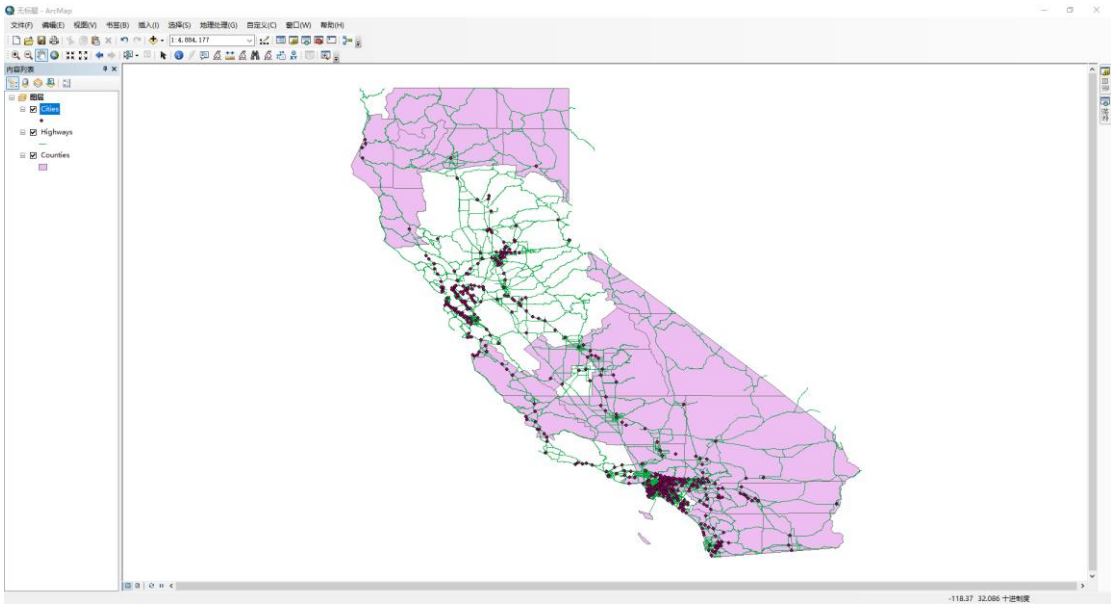

图 2-15 属性过滤运行效果

3 实验目的

（1）在 C#开发环境中进行 AO 客户化组件的开发，了解 ArcObjects 客户化组件库扩展的基本思路和方法；

（2）尝试阅读、理解 ArcGIS 开发包内提供的开发实例（Samples）中相关的 AO 客户化组件实例程序。

4 实验内容

（1）创建定制的按钮命令 command，实现要素图层 FeatureLayer 的唯一值渲染；

（2）创建定制的工具 tool，实现交互的多边形 Polygon 要素查询；

（3）创建定制的工具条 toolbar，加载按钮和工具。

5 实验数据

实验数据位于 ArcEngine10.6 安装目录：

...\DeveloperKit10.6\Samples\data\ California

6 实验步骤

6.1 图层唯一值渲染

本例要实现的是为要素图层设置唯一值渲染器 UniqueValueRenderer，对不同属性字段值采用不同的颜色渲染。首先通过类 UniqueValueRenderer 实现的 IUniqueValueRender 接口，对 IUniqueValueRender 的属性进行赋值，最后将该接口赋值给 IGeoFeatureLayer. Render 属性，实现对要素图层的唯一值渲染。

（1）新建一个 Windows 窗体，用于实现对要素图层的选定字段进行唯一值渲染，窗体类名称为“FrmUVRenderer”，从工具箱中添加 Windows 窗体控件，然后设置各控件的 Text 属性和 Name 属性，如图 2-16 所示。

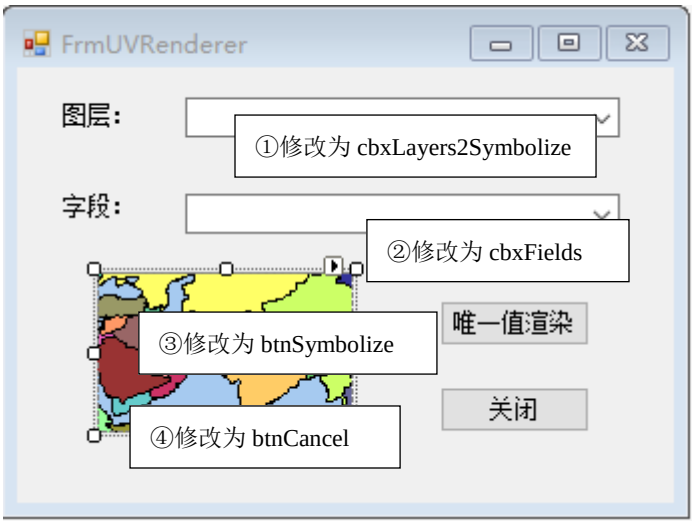

图 2-16 修改控件的 Name 属性

（2）添加类“FrmUVRenderer”的私有成员变量，并在类的构造函数中将私有成员变量引用 m_hookhelper 指向具体的对象。在 FrmUVRenderer.cs 文件导入引用：

```
using ESRI.ArcGIS.Carto;
```

FrmUVRenderer.cs（节选）	功能：将私有成员变量引用指向具体的对象

```
namespace ArcEngineClassLibrary
{
    public partial class FrmUVRenderer : Form
    {
        private IHookHelper m_hookhelper = null;
        private IActiveView m_activeView = null;
        private IMap m_map = null;
        private IFeatureLayer m_layer2Symbolize = null;
        public FrmUVRenderer(IHookHelper hook)
        {
            m_hookhelper = hook;
            m_activeView = m_hookhelper.ActiveView;
            m_map = m_hookhelper.FocusMap;
            InitializeComponent();
        }
}
```

（3）双击 FrmUVRenderer 主窗体空白处，在 FrmUVRenderer_Load()窗体加载事件响应函数中加载图层列表：然后添加私有成员函数 CbxLayersAddItems()和 GetLayers()的定义。在 FrmUVRenderer.cs 中添加引用：

```
using ESRI.ArcGIS.esriSystem;
```

FrmUVRenderer.cs（节选）	功能：加载图层列表

```
private void FrmUVRenderer_Load(object sender, EventArgs e)
{
    CbxLayersAddItems();
}
//将图层列表添加到组合框
private void CbxLayersAddItems()
{
    if (GetLayers() == null) return;
    IEnumLayer layers = GetLayers();
    layers.Reset();
    ILayer layer = layers.Next();
    while (layer != null)
    {
```

```
            if (layer is IFeatureLayer)
            {
                cbxLayers2Symbolize.Items.Add(layer.Name);
            }
            layer = layers.Next();
        }
    }
    //获得当前地图的要素图层列表
    private IEnumLayer GetLayers()
    {
        UID uid = new UIDClass();
        // 接口IFeatureLayer的IID
        uid.Value = "{40A9E885-5533-11d0-98BE-00805F7CED21}";
        if (m_map.LayerCount != 0)
        {
            IEnumLayer layers = m_map.get_Layers(uid, true);
            return layers;
        }
        return null;
    }
```

（4）双击图层列表的 cbxLayers2Symbolize 控件，添加选择图层事件的响应函数代码，在 FrmUVRenderer 类中添加私有成员函数 CbxFieldsAdditems()。在 FrmUVRenderer.cs 中添加引用：

```
using ESRI.ArcGIS.Geodatabase;
```

FrmUVRenderer.cs（节选）	功能：实现图层选择更新字段列表

```
private void cbxLayers2Symbolize_SelectedIndexChanged(object sender,
                                                      EventArgs e)
{
    if (cbxLayers2Symbolize.SelectedItem != null)
    {
        string strLayer2Symbolize =
                cbxLayers2Symbolize.SelectedItem.ToString();
        //获得选择到的图层
        m_layer2Symbolize = GetFeatureLayer(strLayer2Symbolize);
        CbxFieldsAdditems(layer2Symbolize);
        strRendererField = cbxFields.Items[0].ToString();
        cbxFields.Text = strRendererField;
    }
}
```

```
//添加字段列表到组合框
private void CbxFieldsAdditems(IFeatureLayer featureLayer)
{
    IFields fields = featureLayer.FeatureClass.Fields;
    IField field = null;
    cbxFields.Items.Clear();
    for (int i = 0; i < fields.FieldCount; i++)
    {
        field = fields.get_Field(i);
        if (field.Type != esriFieldType.esriFieldTypeGeometry)
            cbxFields.Items.Add(field.Name);
    }
}
```

（5）双击“唯一值渲染”和 “关闭”按钮，添加按钮单击事件响应函数代码；添加实现渲染功能的私有成员函数 Renderer()等的定义。添加引用：

```
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Geometry;
```

FrmUVRenderer.cs（节选）	功能：唯一值渲染

```
private void btnSymbolize_Click(object sender, EventArgs e)
{
    if (m_layer2Symbolize == null) return;
    Renderer();
}

private void btnCancel_Click(object sender, EventArgs e)
{
    this.Close();
}

//唯一值渲染
private void Renderer()
{
    IGeoFeatureLayer pGeoFeatureL = (IGeoFeatureLayer)m_layer2Symbolize;
    IFeatureClass featureClass = pGeoFeatureL.FeatureClass;
    string strRendererField = string.Empty;
    strRendererField = cbxFields.SelectedItem.ToString();
    //找出rendererField在字段中的编号
    int lfieldNumber = featureClass.FindField(strRendererField);
    if (lfieldNumber == -1)
    {
        MessageBox.Show("Can't find field called " + strRendererField);
```

```
            return;
        }
        IUniqueValueRenderer pUniqueValueR = CreateRenderer(featureClass);
        if (pUniqueValueR == null) return;
        pGeoFeatureL.Renderer = (IFeatureRenderer)pUniqueValueR;
        m_activeView.PartialRefresh(esriViewDrawPhase.esriViewGeography, null,
            m_activeView.Extent);
    }
    //创建唯一值渲染器
    private IUniqueValueRenderer CreateRenderer(IFeatureClass featureClass)
    {
        int uniqueValuesCount = GetUniqueValuesCount(featureClass,
                                                     strRendererField);
        System.Collections.IEnumerator enumerator = GetUniqueValues(
                                                featureClass, strRendererField);
        if (uniqueValuesCount == 0) return null;
        IEnumColors pEnumRamp =
                    GetEnumColorsByRandomColorRamp(uniqueValuesCount);
        pEnumRamp.Reset();
        IUniqueValueRenderer pUniqueValueR = new UniqueValueRendererClass();
        //只用一个字段进行单值着色
        pUniqueValueR.FieldCount = 1;
        //用于区分着色的字段
        pUniqueValueR.set_Field(0, strRendererField);
        IColor pColor = null;
        ISymbol symbol = null;
        enumerator.Reset();
        while (enumerator.MoveNext())
        {
            object codeValue = enumerator.Current;
            pColor = pEnumRamp.Next();
            switch (featureClass.ShapeType)//不同的要素图层类型
            {
                case esriGeometryType.esriGeometryPoint: //点
                    ISimpleMarkerSymbol markerSymbol = new
                        SimpleMarkerSymbolClass() as ISimpleMarkerSymbol;
                    markerSymbol.Color = pColor;
                    symbol = markerSymbol as ISymbol;
                    break;
                case esriGeometryType.esriGeometryPolyline: //线
                    ISimpleLineSymbol lineSymbol = new
                                SimpleLineSymbolClass() as ISimpleLineSymbol;
                    lineSymbol.Color = pColor;
                    symbol = lineSymbol as ISymbol;
```

```
                    break;
                case esriGeometryType.esriGeometryPolygon: //多边形
                    ISimpleFillSymbol fillSymbol = new
                            SimpleFillSymbolClass() as ISimpleFillSymbol;
                    fillSymbol.Color = pColor;
                    symbol = fillSymbol as ISymbol;
                    break;
                default:
                    break;
            }
            //将每次得到的要素字段值和修饰它的符号放入着色对象中
            pUniqueValueR.AddValue(codeValue.ToString(),
                                                    strRendererField, symbol);
        }
        return pUniqueValueR;
    }
```

（6）添加私有成员函数 GetUniqueValues()、GetUniqueValuesCount()、GetEnumColorsByRandomColorRamp()等的定义。

FrmUVRenderer.cs（节选）	功能：获得唯一值及个数、颜色色带

```
//使用数据统计组件对象获得字段的唯一值列表
private System.Collections.IEnumerator GetUniqueValues(
                                    IFeatureClass featureClass, string strField)
{
    ICursor cursor = (ICursor)featureClass.Search(null, false);
    IDataStatistics dataStatistics = new DataStatisticsClass();
    dataStatistics.Field = strField;
    dataStatistics.Cursor = cursor;
    System.Collections.IEnumerator enumerator = dataStatistics.UniqueValues;
    return enumerator;
}
//使用数据统计组件对象获得字段的唯一值的个数
private int GetUniqueValuesCount(IFeatureClass featureClass, string strField)
{
    ICursor cursor = (ICursor)featureClass.Search(null, false);
    IDataStatistics dataStatistics = new DataStatisticsClass();
    dataStatistics.Field = strField;
    dataStatistics.Cursor = cursor;
    System.Collections.IEnumerator enumerator = dataStatistics.UniqueValues;
    return dataStatistics.UniqueValueCount;
}
//生成一个颜色列表，用于唯一值渲染
private IEnumColors GetEnumColorsByRandomColorRamp(int colorSize)
```

```
{
    IRandomColorRamp pColorRamp = new RandomColorRampClass();
    //起始和终止颜色
    pColorRamp.StartHue = 0;
    pColorRamp.EndHue = 360;
    pColorRamp.MinSaturation = 15;
    pColorRamp.MaxSaturation = 30;
    pColorRamp.MinValue = 99;
    pColorRamp.MaxValue = 100;
    pColorRamp.Size = colorSize;
    bool ok = true;
    pColorRamp.CreateRamp(out ok);
    IEnumColors pEnumRamp = pColorRamp.Colors;
    pEnumRamp.Reset();
    return pEnumRamp;
}
```

（7）新建一个按钮命令 Command 类，类名称为“CmdUVRenderer”，然后修改类的构造函数中命令按钮的分类、命名等字符串。在按钮命令类“CmdUVRenderer”的 OnClick 事件中将 IHookHelper 接口的引用传递给窗体类“FrmUVRenderer”并弹出该窗体，使得窗体能够访问 ArcMap 主应用程序的资源。重新生成解决方案并编译运行，在 ArcMap 中添加按钮。运行选择需要进行唯一值渲染的图层和字段，如图 2-17 所示；

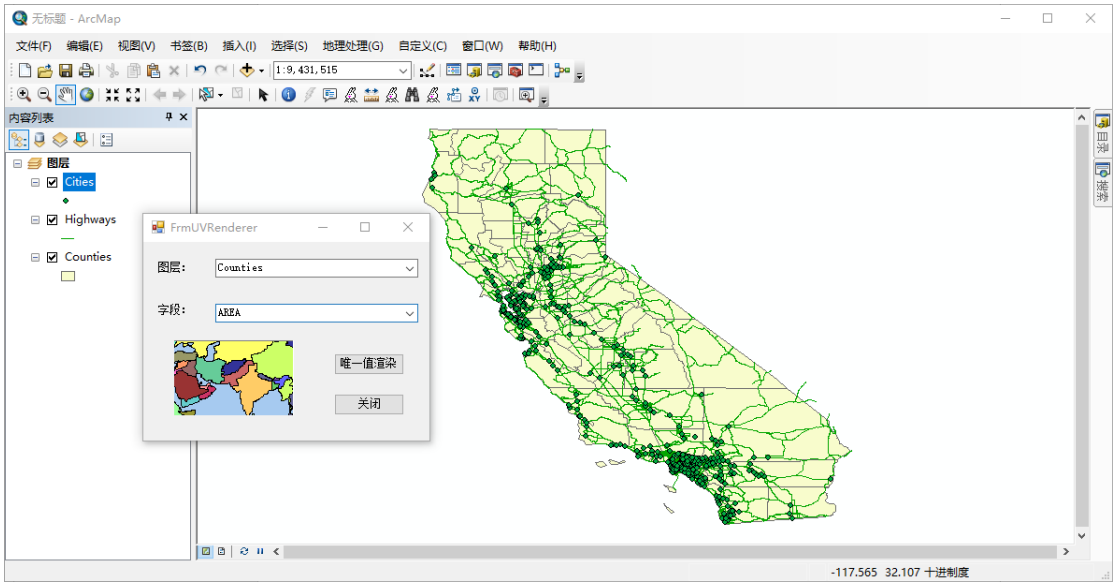

图 2-17 选择需要进行唯一值渲染的图层和字段

点击【唯一值渲染】，渲染效果如图 2-18 所示。

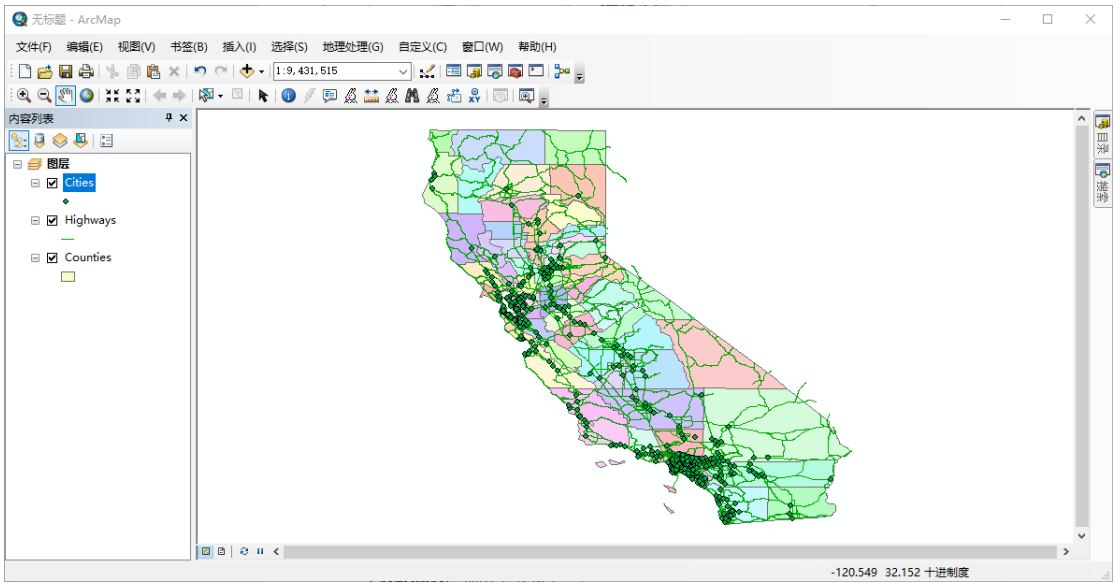

图 2-18 唯一值渲染效果

ArcObjects 使用点样式 MarkerSymbol、线样式 LineSymbol 和填充样式 FillSymbol 来绘制地理要素或图形几何形状。此外还有两种特殊的符号：一种是用于文字标注的 TextSymbol；另外一种是用于显示饼图等三维对象 3D ChartSymbol，如图 2-19 所示。所有的符号类都实现了 ISymbol 和 IMapLevel 接口，前者定义了一个符号对象的基本属性和方法；后者定义的 MapLevel 属性可以确定符号的显示顺序，当绘制了一个面符号和一条线符号后，可以使用这个属性来确定哪一个符号在上面而不被其它的符号遮盖。

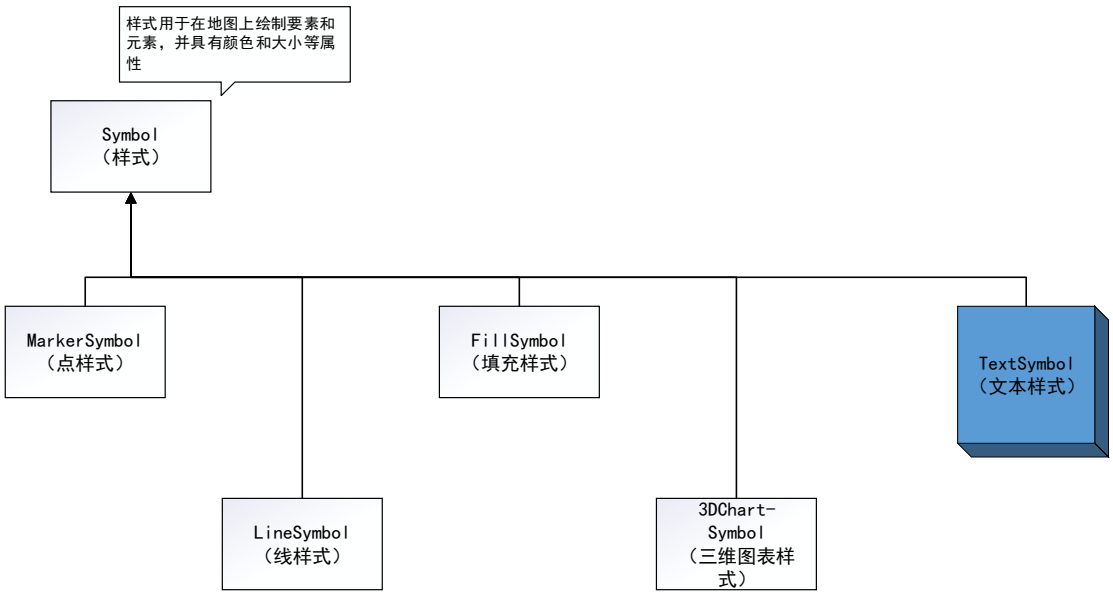

图 2-19 样式对象模型图

点样式 MarkerSymbol 对象是用于修饰点对象的符号，它拥有五个子类，其中不同的子类可以产生不同类型的点符号，如图 2-20 所示。所有的 MarkerSymbol 类都实现了 IMarkerSymbol 接口，这个接口定义了标记符号的公共方法和属性，如角度、颜色、大小和 XY 偏移量等。

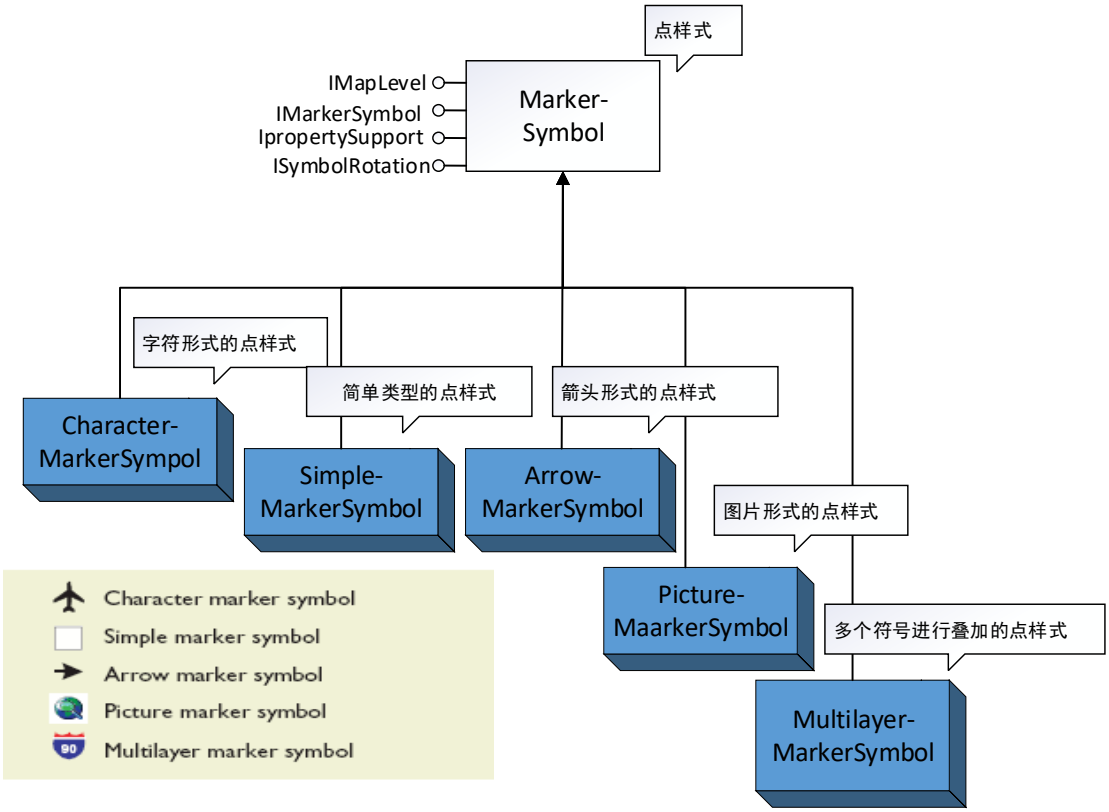

图 2-20 点样式对象

线样式 LineSymbol 对象是用于修饰线型几何对象的符号，ILineSymbol 作为每一种 LineSymbol 类都要实现的接口定义了两个公共属性，即 Color 和 Width。这两个属性为所有的线符号所需要，前者用于设置线符号的颜色，后者用于设定线符号对象的宽度。LineSymbol 抽象类有 6 个子类，分别是 SimpleLineSymbol、CartographicLineSymbol 、 HashLineSymbol 、 MarkerLineSymbol 、 MultiLayerLineSymbol、PictureLineSymbol，如图 2-21 所示。

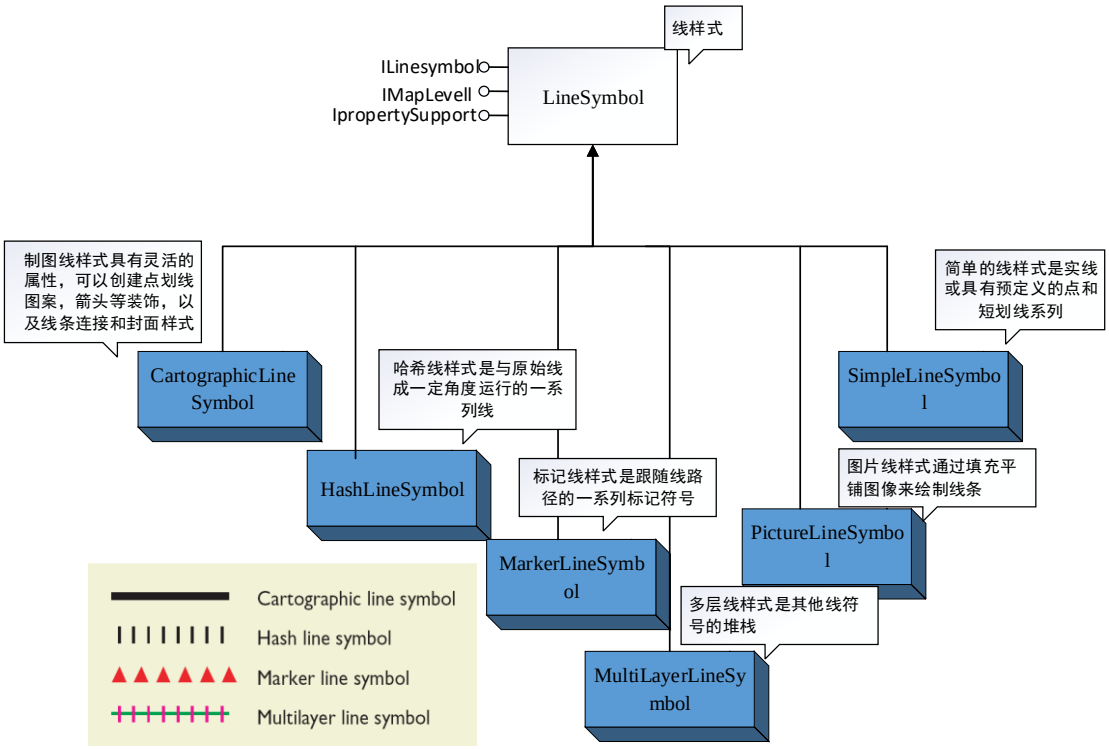

图 2-21 线样式对象

填充样式 FillSymbol 是用来修饰具有面积的几何形体的符号对象，它实现了 IFillSymbol，这个接口定义了两个属性 Color 和 OutLine，以满足所有类型的 FillSymbol 对象的公共属性设置，如图 2-22 所示。IFillSymbol.color 可以设置填充符号的基本颜色，如果不设置这个属性，则使用默认颜色进行填充。IFillSymbol.OutLine 属性可以设置填充符号的外边框，这个外边框是一个线对象，使用 ILineSymbol 对象修饰，在默认情况下它是一个 Soild 类型的简单线符号。

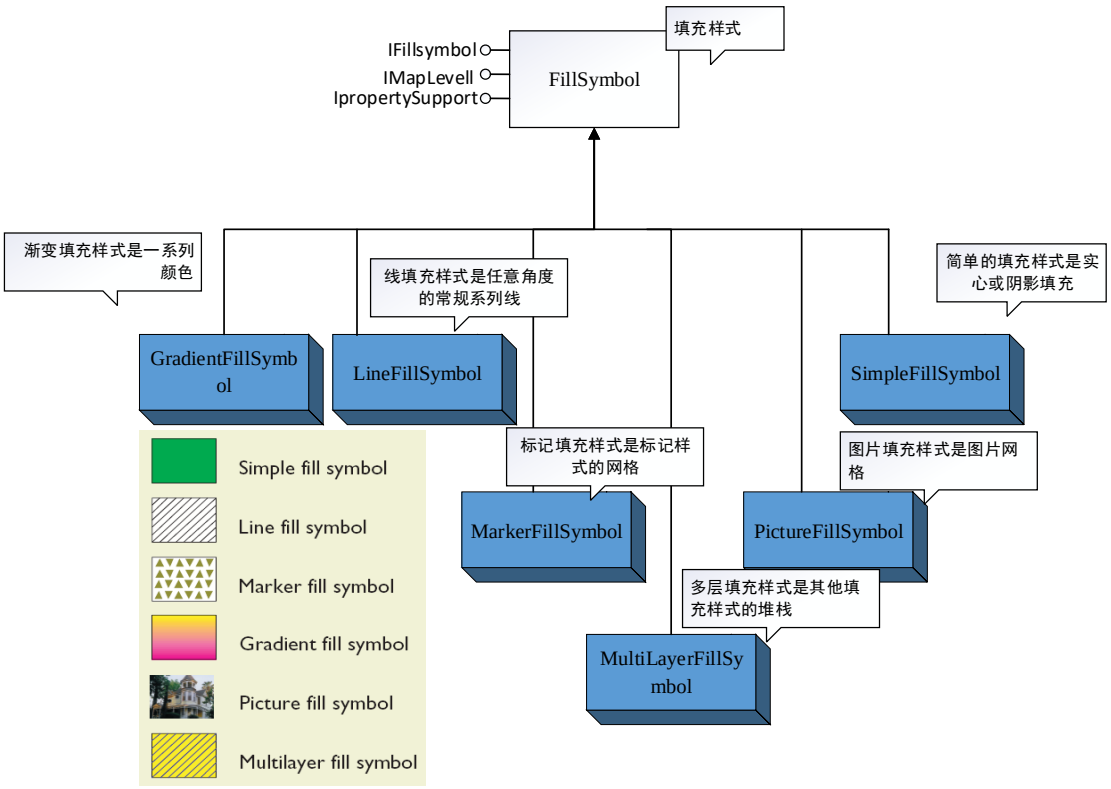

图 2-22 填充样式对象模型图

6.2 多边形要素查询工具

ArcObjects 中的工具 Tool 既具有 Command 的功能，又具有与 ArcMap 界面交互的功能，Command 的功能代码必须写在 ICommand 的 OnClick 事件中，而所有实现交互功能的代码必须写在 ITool 接口的各个事件中。用户可在 ITool 接口的各个事件中写入相关代码，表示用户与 ArcMap 界面交互时一旦触发某事件要调用的功能。用户通常在工具 Tool 类中实现 ICommand 和 ITool 接口。ITool 接口包括 mouse move, mouse button press/release, keyboard key press/release, double-click 以及 right click 等事件、Cursor 属性和 Refresh 方法。

本例实现的是交互画一个多边形 polygon，根据该 polygon 查询出某图层上与 polygon 相交的要素并高亮显示出来。通过 RubberPolygon 对象（见图 2-23）来实现接口 IRubberBand 的接口对象，用 IRubberBand.TrackNew 方法在地图上交互画出 polygon；然后创建 ISpatialFilter 接口对象实现多边形选择功能，通过 ILayer 接口实例获得 IFeatureSelection 接口，调用 IFeatureSelection 菜单 SelectFeatures 方法将结果高亮显示。具体步骤如下：

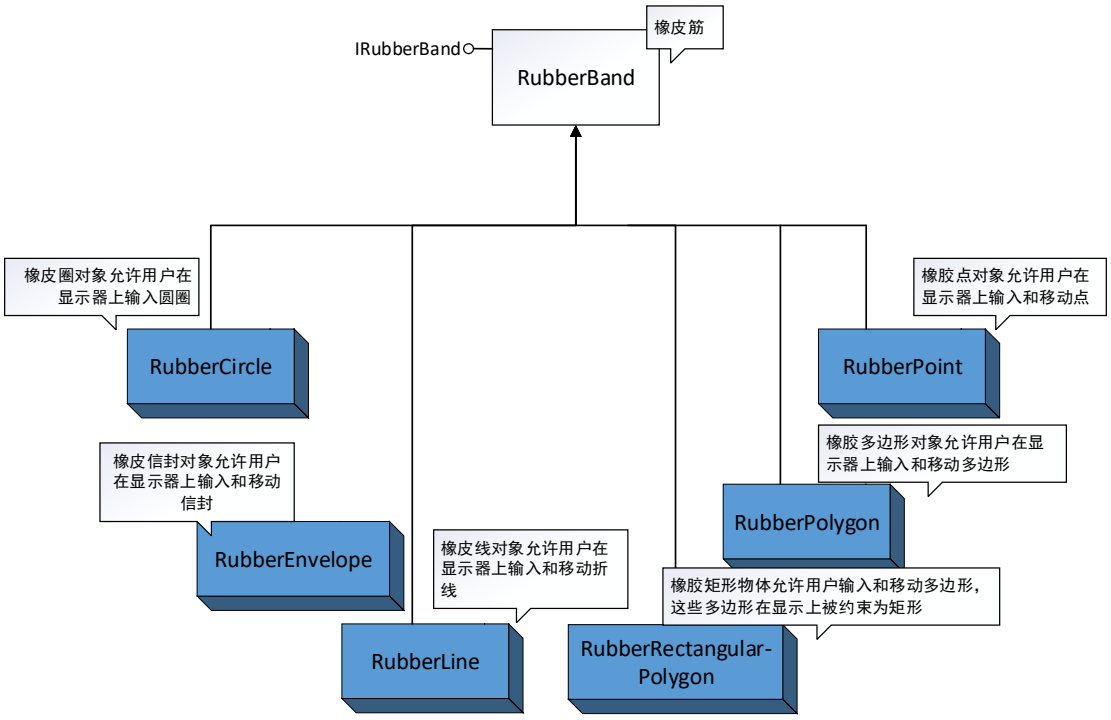

图 2-23 橡皮筋对象模型图

（1）新建工具类，选中“ArcEngineClassLibrary”项目，单击鼠标右键菜单，选择【添加】→【新建项】，选择【Visual C#】→【ArcGIS】→【Extending ArcObjects】→【Base Tool】模板，工具类命名为“ToolPolygonSelect”，如图 2-24 所示。修改类的构造函数中关于命令按钮的分类、命名等字符串。

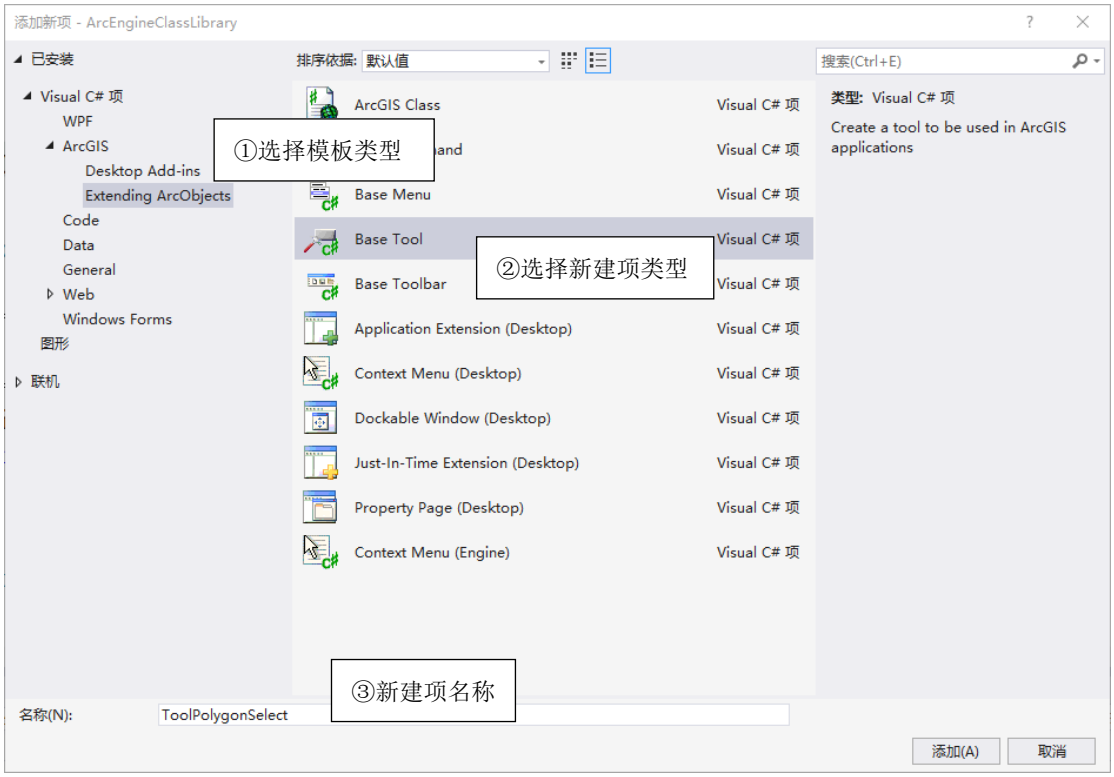

图 2-24 新建工具类

（2）修改类“ToolPolygonSelect”按钮点击事件OnClick()和鼠标按下事件OnMouseDown()的响应函数代码，参考代码如下。并在ToolPolygonSelect.cs文件开头添加引用：

```
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Carto;
```

ToolPolygonSelect.cs（节选）	功能：选择要素图层及多边形选择要素高亮显示

```
public override void OnClick()
{
    // TODO: Add ToolPolygonSelect.OnClick implementation
    // 获得选择图层
    FrmSelectLayer frmSelectLayer = new FrmSelectLayer(m_hookHelper);
    frmSelectLayer.ShowDialog();
    m_layer = frmSelectLayer.lyr;
}
public override void OnMouseDown(int Button, int Shift, int X, int Y)
{
    // TODO:  Add ToolPolygonSelect.OnMouseDown implementation
    IRubberBand polygonRubber = new RubberPolygonClass();
    IPolygon polygon = polygonRubber.TrackNew(
            m_hookHelper.ActiveView.ScreenDisplay, null) as IPolygon;
    // 进行多边形选择
    ISpatialFilter spFilter = new SpatialFilterClass();
    spFilter.Geometry = polygon;
    spFilter.SpatialRel = esriSpatialRelEnum.esriSpatialRelIntersects;
    IFeatureSelection fSel = m_layer as IFeatureSelection;
    fSel.SelectFeatures(spFilter,
            esriSelectionResultEnum.esriSelectionResultNew, false);
    fSel.SelectionSet.Refresh();
    m_hookHelper.ActiveView.Refresh();
    // 显示选择结果属性表格
    FrmSelectResult pResult1 = new FrmSelectResult(m_layer as IFeatureLayer);
    pResult1.Show();
}
```

（3）新建一个 Windows 窗体用来选择图层以供要素查询，名称为“FrmSelectLayer”，如图 2-25 所示。

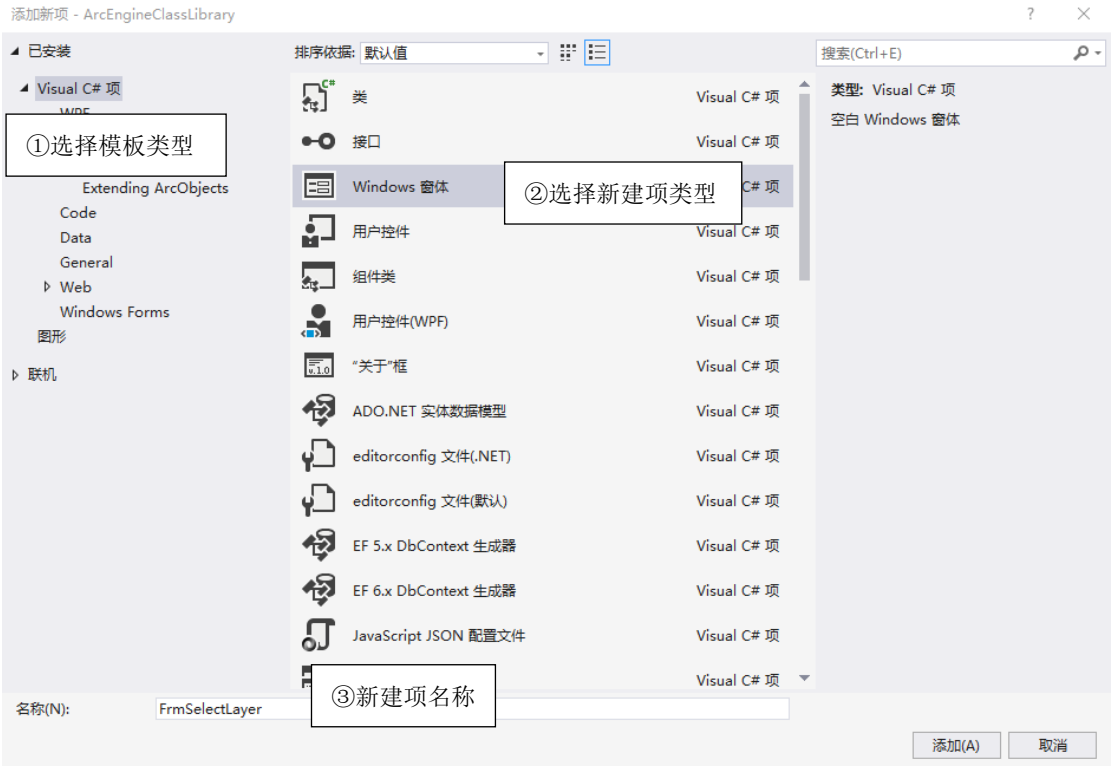

图 2-25 新建图层选择窗体

为窗体类“FrmSelectLayer”添加窗体控件并修改各控件的 Name 属性值，如图 2-26 所示。

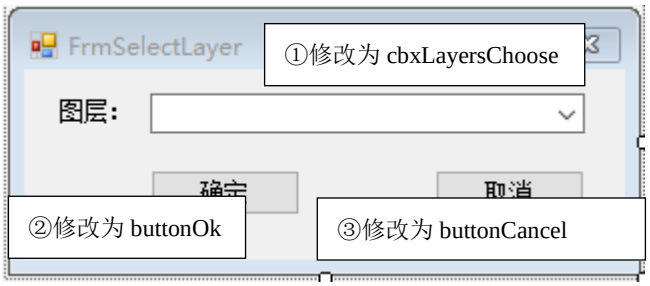

图 2-26 修改各控件的 Name 属性

（4）为类“FrmSelectLayer”添加 IHookHelper 接口类型的私有成员变量 m_hookhelper，在类的构造函数中将该引用变量指向具体的对象。添加私有成员变量“ILayer m_layer”和只读属性“ILayer lyr”，用于保存选择到的要素图层。

FrmSelectLayer.cs（节选）	功能：保存选择到的要素图层

```
private ILayer m_layer = null;
public ILayer lyr
{
    get{    return m_layer;    }
}
```

（5）点击窗体 FrmSelectLayer 空白处添加 FrmSelectLayer_Load()窗体加载事件响应函数，然后添加 CbxLayersAddItems()私有成员函数。

FrmSelectLayer.cs（节选）	功能：将当前地图所有图层添加到组合框

```
private void FrmSelectLayer_Load(object sender, EventArgs e)
{
    CbxLayersAddItems();
}
//获取当前地图所有要素图层，并添加到组合框
private void CbxLayersAddItems()
{
    IEnumLayer layers = m_hookhelper.FocusMap.Layers;
    layers.Reset();
    ILayer layer = layers.Next();
    while (layer != null)
    {
        if (layer is IFeatureLayer)
        {
            cbxLayersChoose.Items.Add(layer.Name);
        }
        layer = layers.Next();
    }
}
```

（6）双击“确定”和“取消”按钮，添加按钮单击事件响应函数；然后添加由名字获得要素图层的私有成员函数 GetFeatureLayer()。

FrmSelectLayer.cs（节选）	功能：获取选中的要素图层

```
private void buttonOk_Click(object sender, EventArgs e)
{
    string strLayer = cbxLayersChoose.SelectedItem.ToString();
    m_layer = GetFeatureLayer(strLayer);
    this.Close();
}
//由名字获得要素图层
private IFeatureLayer GetFeatureLayer(string layerName)
{
    //get the layers from the maps
    IEnumLayer layers = m_hookhelper.FocusMap.Layers;
    layers.Reset();
    ILayer layer = null;
    while ((layer = layers.Next()) != null)
    {
        if (layer.Name == layerName)
            return layer as IFeatureLayer;
    }
    return null;
}
private void buttonCancel_Click(object sender, EventArgs e)
{
```

```
    this. Close () ;
}
```

（7）在解决方案管理器选中项目“ArcEngineClassLibrary”，右键选择【添加】→【新建项】，添加一个 Windows 窗体用于要素查询显示结果的属性信息，类名称为“FrmSelectResult”。然后查看该窗体类代码，添加私有成员变量并修改 FrmSelectResul()构造函数，如下所示。

FrmSelectResult.cs（节选）	功能：修改构造函数

```
namespace ArcEngineClassLibrary
{
    public partial class FrmSelectResult : Form
    {
        private IFeatureLayer m_layer = null;
        private DataTable m_attributeTable = null;
        public FrmSelectResult(IFeatureLayer lyr)
        {
            m_layer = lyr;
            InitializeComponent();
        }
    }
}
```

（8）在窗体上添加一个 DataGridView 控件用于显示要素选择结果的属性信息，修改 DataGridView 控件的 Name 属性为 DataGrdView，如图 2-27 所示。

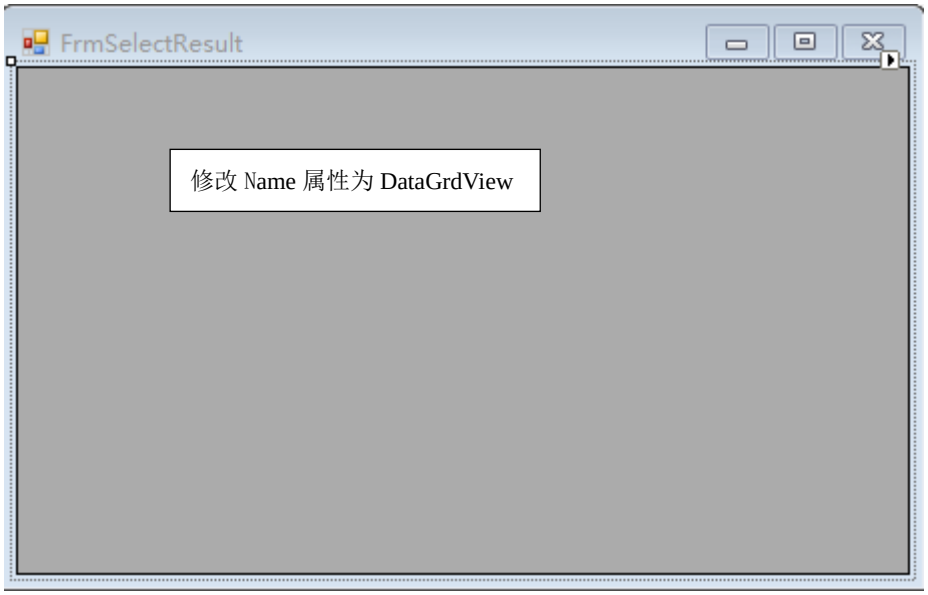

图 2-27 添加 DataGridView 控件

（9）双击 FrmSelectResult 窗体空白处，添加 FrmSelectResult_Load()事件响应函数，创建数据表格并显示；添加私有成员函数 CreateAttributeTable()、getValidFeatureClassName() 、 CreateDataTable() 、 getShapeType() 等 。 在 FrmSelectResult.cs 中添加引用：

```
using ESRI. ArcGIS. Geodatabase;
```

FrmSelectResult.cs（节选）	功能：创建数据表格所需函数

```
private void FrmSelectResult_Load(object sender, EventArgs e)
{
    CreateAttributeTable(m_layer);
}
//创建数据表作为GridView的数据源
private void CreateAttributeTable(ILayer player)
{
    string tableName;
    tableName = getValidFeatureClassName(player.Name);
    m_attributeTable = CreateDataTable(player, tableName);
    //GridView的东西不允许用户进行修改
    m_attributeTable.DefaultView.AllowNew = false;
    //设置数据源
    this.DataGrdView.DataSource = m_attributeTable;
    this.Text = "属性表[" + tableName + "] " + "记录数：" +
        m_attributeTable.Rows.Count.ToString();
}
//将要素类名中的“.”替换为“_”
private string getValidFeatureClassName(string FCname)
{
    int dot = FCname.IndexOf(".");
    if (dot != -1)
    {
        return FCname.Replace(".", "_");
    }
    return FCname;
}
//根据图层选择集来创建数据表
private DataTable CreateDataTable(ILayer pLayer, string tableName)
{
    //创建空DataTable,并确定表头的名称
    DataTable pDataTable = CreateDataTableByLayer(pLayer, tableName);
    //取得图层类型
    string shapeType = getShapeType(pLayer);
    //创建DataTable的行对象
    DataRow pDataRow = null;
    //从ILayer查询到ITable
    IFeatureSelection ipFeatSelect = pLayer as IFeatureSelection;
    ICursor pCursor = null;
    ipFeatSelect.SelectionSet.Refresh();
    ipFeatSelect.SelectionSet.Search(null, true, out pCursor);
    //取得ITable中的行信息
    IRow pRow = pCursor.NextRow();
    int n = 0;
```

```
    while (pRow != null)
    {
        //新建DataTable的行对象
        pDataRow = pDataTable.NewRow();
        for (int i = 0; i < pRow.Fields.FieldCount; i++)
        {
            //如果字段类型为esriFieldTypeGeometry
            if (pRow.Fields.get_Field(i).Type ==
                                esriFieldType.esriFieldTypeGeometry)
            {
                pDataRow[i] = shapeType;
            }
            //如果字段类型为esriFieldTypeBlob类型的数据
            else if (pRow.Fields.get_Field(i).Type ==
                                    esriFieldType.esriFieldTypeBlob)
            {
                pDataRow[i] = "Element";
            }
            else
            {
                pDataRow[i] = pRow.get_Value(i);
            }
        }
        //添加DataRow到DataTable
        pDataTable.Rows.Add(pDataRow);
        pDataRow = null;
        n++;
        //为保证效率，一次只装载最多条记录
        if (n == 2000)
        {
            pRow = null;
        }
        else
        {
            pRow = pCursor.NextRow();
        }
    }
    return pDataTable;
}
//获得图层的几何（点、线、面）类型
private string getShapeType(ILayer pLayer)
{
    IFeatureLayer pFeatLyr = (IFeatureLayer)pLayer;
    switch (pFeatLyr.FeatureClass.ShapeType)
```

```
    {
        case esriGeometryType.esriGeometryPoint:
            return "Point";
        case esriGeometryType.esriGeometryPolyline:
            return "Polyline";
        case esriGeometryType.esriGeometryPolygon:
            return "Polygon";
        default:
            return "";
    }
}
```

（10）添加私有成员函数 CreateDataTableByLayer()和 ParseFieldType()，并在 FrmSelectResult.cs 中添加引用：

```
using ESRI.ArcGIS.Geometry;
```

FrmSelectResult.cs（节选）	功能：根据要读图层字段组初始化数据表表头

```
private DataTable CreateDataTableByLayer(ILayer pLayer, string tableName)
{
    //创建一个DataTable表
    DataTable pDataTable = new DataTable(tableName);
    //取得ITable接口
    ITable pTable = pLayer as ITable;
    IField pField = null;
    DataColumn pDataColumn;
    //根据每个字段的属性建立DataColumn对象
    for (int i = 0; i < pTable.Fields.FieldCount; i++)
    {
        pField = pTable.Fields.get_Field(i);
        //新建一个DataColumn并设置其属性
        pDataColumn = new DataColumn(pField.Name);
        if (pField.Name == pTable.OIDFieldName)
        {
            pDataColumn.Unique = true;//字段值是否唯一
        }
        //字段值是否允许为空
        pDataColumn.AllowDBNull = pField.IsNullable;
        //字段别名
        pDataColumn.Caption = pField.AliasName;
        //字段数据类型
        pDataColumn.DataType =
                    System.Type.GetType(ParseFieldType(pField.Type));
        //字段默认值
        pDataColumn.DefaultValue = pField.DefaultValue;
        //当字段为String类型是设置字段长度
        if (pField.VarType == 8)
```

```
            {
                pDataColumn. MaxLength = pField. Length;
            }
            //字段添加到表中
            pDataTable. Columns. Add(pDataColumn);
            pField = null;
            pDataColumn = null;
        }
        return pDataTable;
}
//将字段类型转换为字符串
private string ParseFieldType(esriFieldType fieldType)
{
        switch (fieldType)
        {
            case esriFieldType. esriFieldTypeBlob:
                return "System. String";
            case esriFieldType. esriFieldTypeDate:
                return "System. DateTime";
            case esriFieldType. esriFieldTypeDouble:
                return "System. Double";
            case esriFieldType. esriFieldTypeGeometry:
                return "System. String";
            case esriFieldType. esriFieldTypeGlobalID:
                return "System. String";
            case esriFieldType. esriFieldTypeGUID:
                return "System. String";
            case esriFieldType. esriFieldTypeInteger:
                return "System. Int32";
            case esriFieldType. esriFieldTypeOID:
                return "System. String";
            case esriFieldType. esriFieldTypeRaster:
                return "System. String";
            case esriFieldType. esriFieldTypeSingle:
                return "System. Single";
            case esriFieldType. esriFieldTypeSmallInteger:
                return "System. Int32";
            case esriFieldType. esriFieldTypeString:
                return "System. String";
            default:
                return "System. String";
        }
}
```

（11）在 ArcMap 中选择【自定义】→【自定义模式】，加载客户化组件，选

择多边形查询要素图层，如图 2-28 所示。

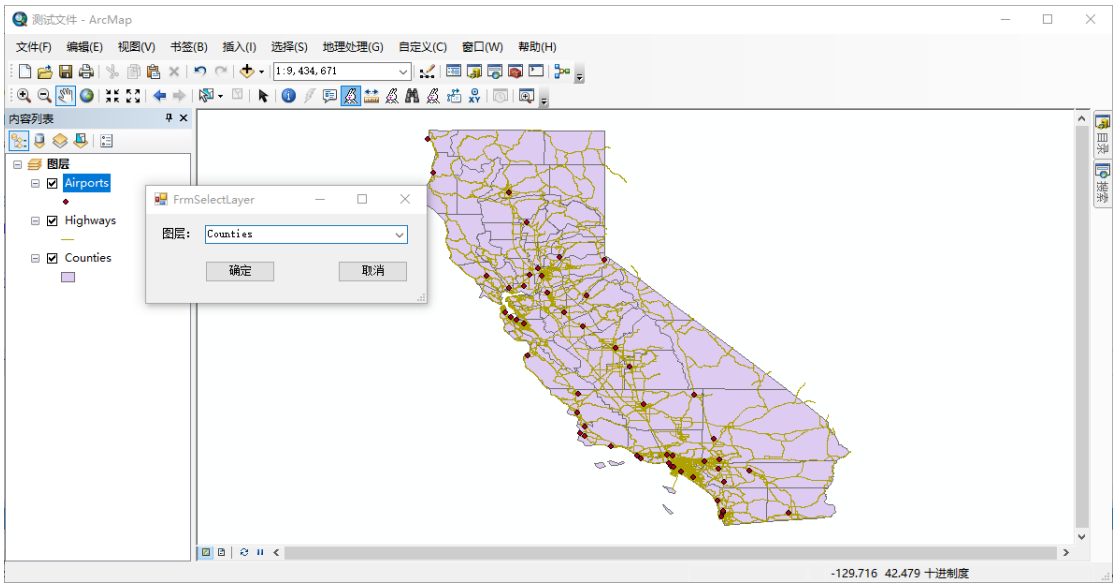

图 2-28 加载客户化组件

在 ArcMap 中的运行多边形交互选择要素并查看效果，如图 2-29 所示。

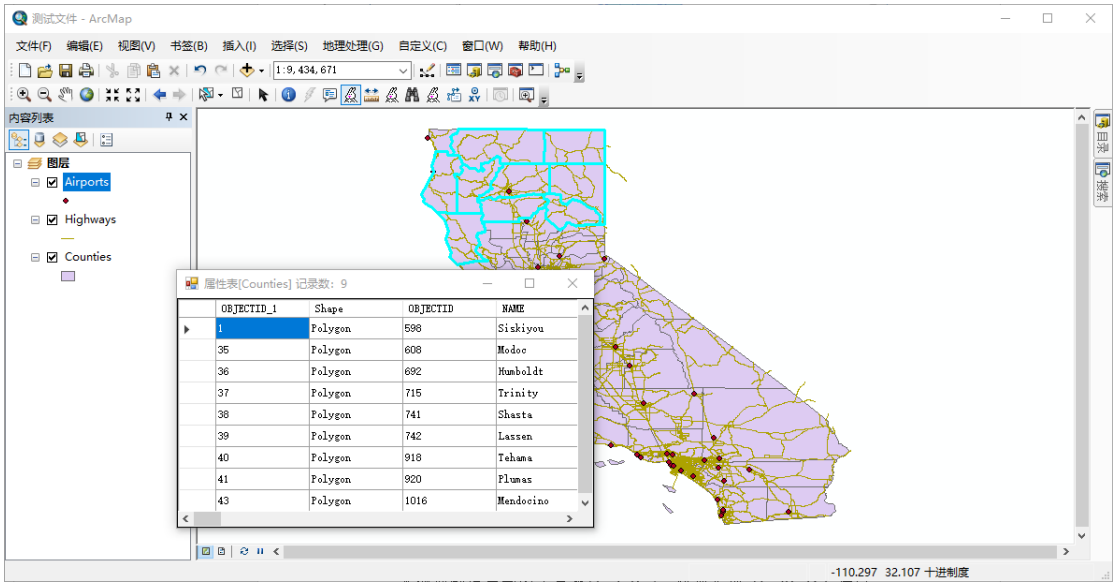

图 2-29 要素选择效果

6.3 创建定制的工具条

本例要创建定制的工具条(Tool Bar)，必须实现 IToolBarDef 接口。IToolBarDef 接口包括 Caption、ItemCount 及 Name 三个属性和 GetItemInfo 方法。ItemCount 属性表示 ToolBar 显示的条目(Button、Tool 或其它控件)数。GetItemInfo 方法定义工具条上各条目的 CLSID，其中，参数 pos 表示条目在 ToolBar 中的位置，itemDef 是定义相应位置条目的 IItemDef 对象。具体步骤如下：

（1）新建工具条类，解决方案管理器选中“ArcEngineClassLibrary”项目，

单击鼠标右键菜单，选择【添加】→【新建项】，选择【Visual C#】→【ArcGIS】→【Extending ArcObjects】→【Base Toolbar】模板，类命名为“ToolbarFeatureLayer”，如图 2-30 所示。

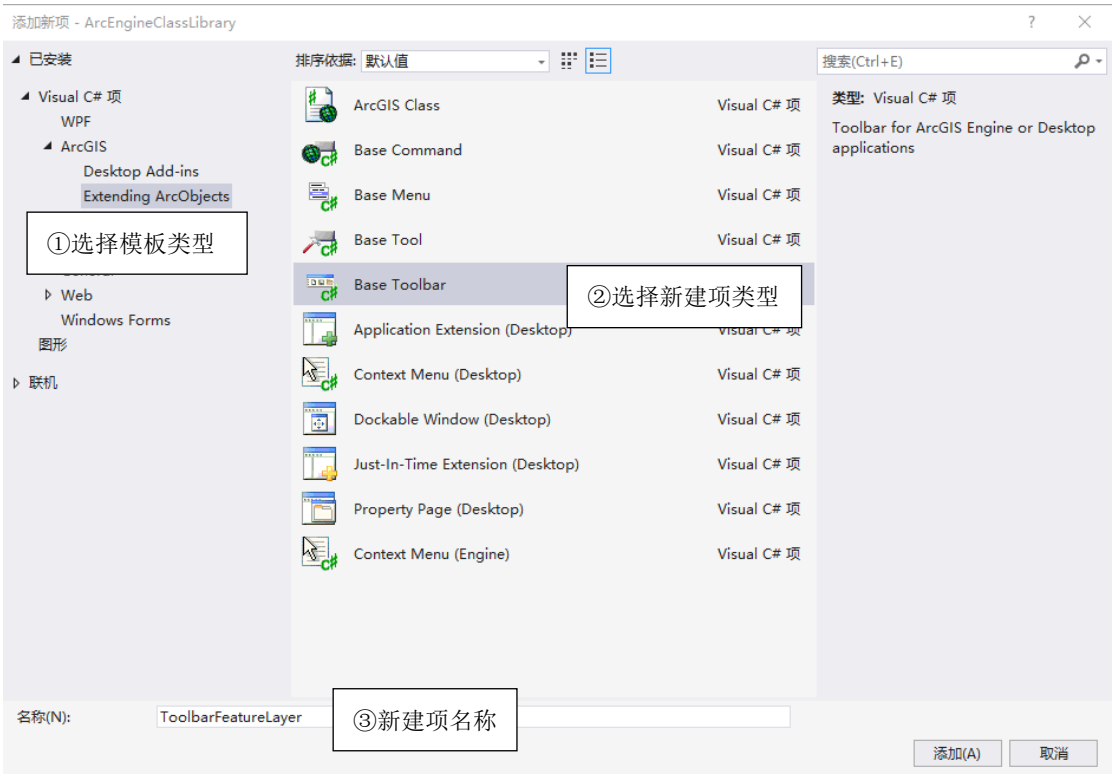

图 2-30 新建定制工具条类

（2）在弹出的新建项向导选项对话框中选择“Desktop ArcMap”，如图 2-31 所示。

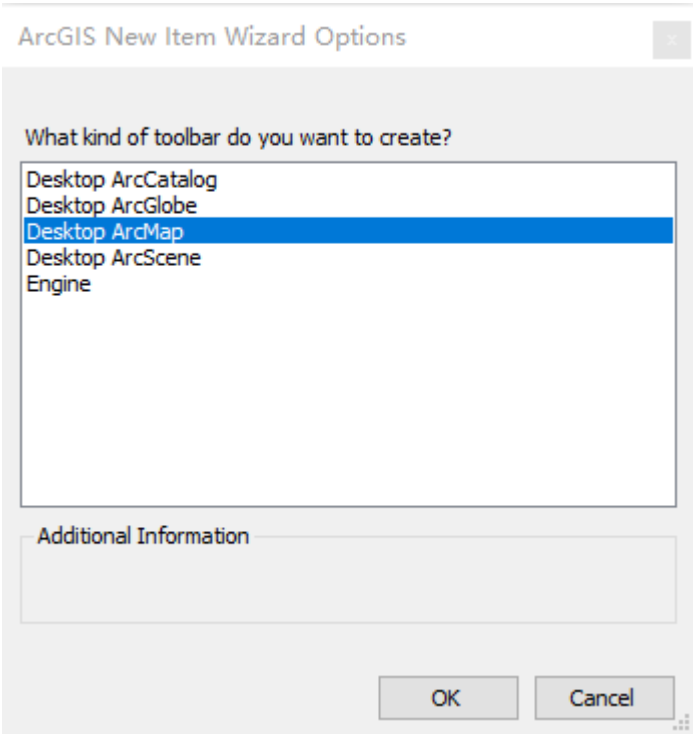

图 2-31 ArcGIS 新建项向导选项对话框

（3）工具条条目的 CLSID 分为两种：（a）系统 CLSID，代表 ArcGIS 的一个功能，如"esriArcMapUI.ZoomInTool"、"esriArcMapUI.ZoomOutTool"等；（b）用户定制 CLSID，表示用户自己定义的功能，其引用方式为“项目名称.定制功能类名称”，即"ProjectName.ClassName"，必须注意，这里“项目名称 ProjectName”即为当前项目的名称，“定制功能类名称 ClassName”是项目中实现的一个功能类的名称。在工具条类“ToolbarFeatureLayer”的构造函数中写入添加工具条的项的代码。

ToolbarFeatureLayer.cs（节选）	功能：添加工具条上的项

```
public ToolbarFeatureLayer()
{
    // TODO: Define your toolbar here by adding items
    AddItem("esriArcMapUI. ZoomInTool");
    BeginGroup(); //Separator
    AddItem("{FBF8C3FB-0480-11D2-8D21-080009EE4E51}", 1);
    AddItem(new Guid("FBF8C3FB-0480-11D2-8D21-080009EE4E51"), 2);
    BeginGroup();
    // 添加用户自定制的按钮和工具
    AddItem("ArcEngineClassLibrary. CmdUVRenderer");
    AddItem("ArcEngineClassLibrary. ToolPolygonSelect");
}
```

（4）编译程序调出 ArcMap，点击【自定义】→【自定义模式】，在 ArcMap 中加载工具条，如图 2-32 所示。

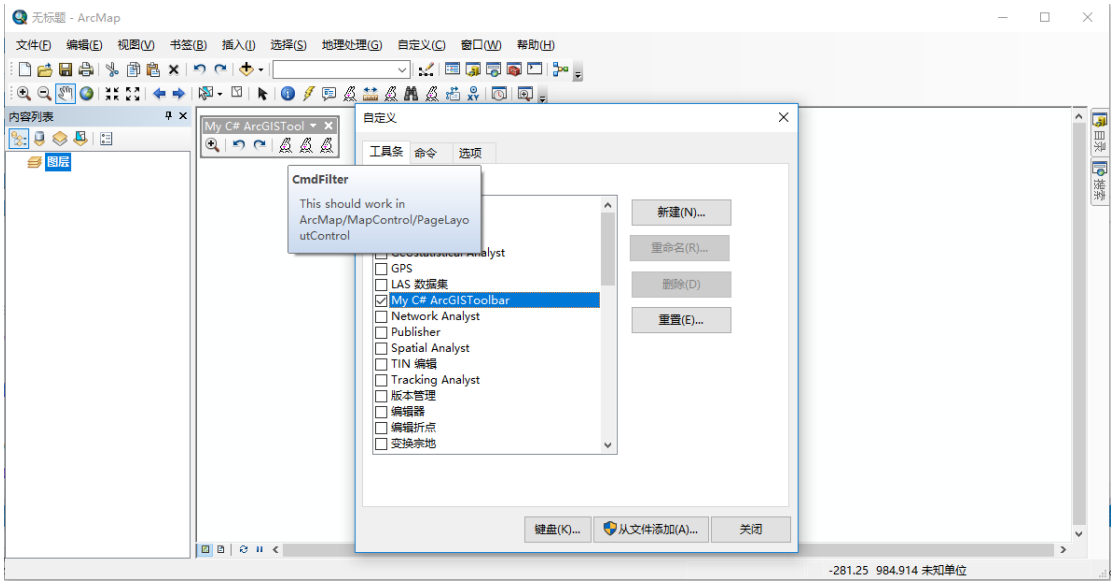

图 2-32 ArcMap 中添加自定义工具条

6.4 阅读实例代码

安装目录“…\DeveloperKit10.6\Samples\”有名为 arcobjects-sdk-community-samples-master.zip 的压缩包，解压缩后主要的按钮和工具都在 Controls 目录下以

ControlsCommand...开头的文件夹中（如图 2-33 所示），按钮或工具的类要从 BaseCommand 或 BaseTool 派生，或者直接从 ICommand 或 ITool 派生。选取二次开发包提供的程序实例中 ControlsCommandsPanZoomCommands 和 ControlsCommandsSelectCommands 阅读并解释代码，在 ArcMap 应用程序中加载按钮或工具并执行，说明按钮或工具的功能。

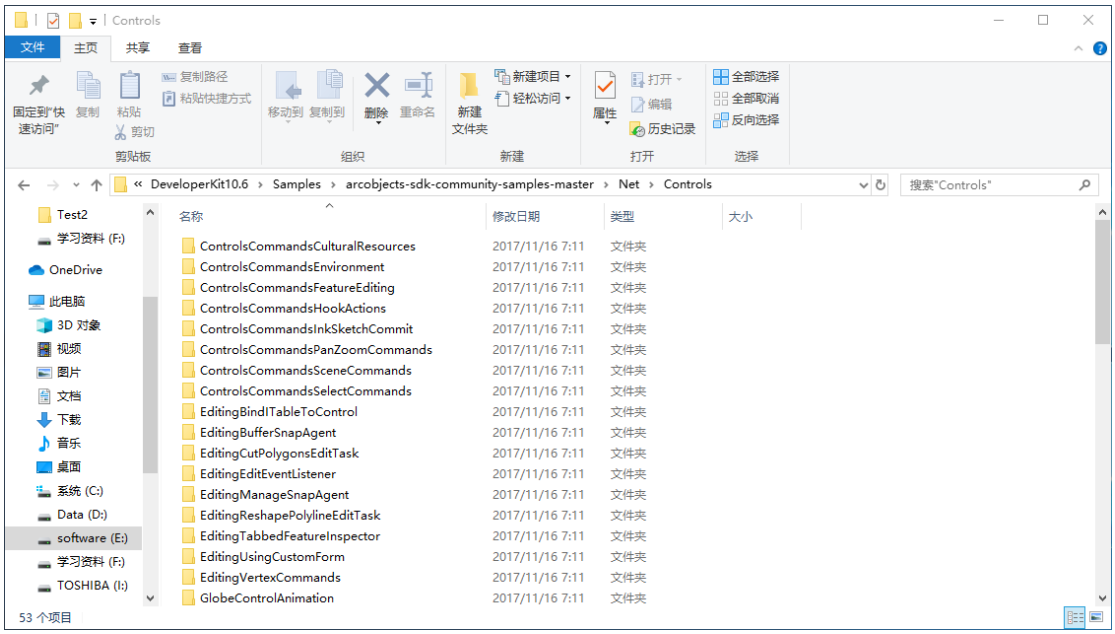

图 2-33 AO 开发程序实例目录