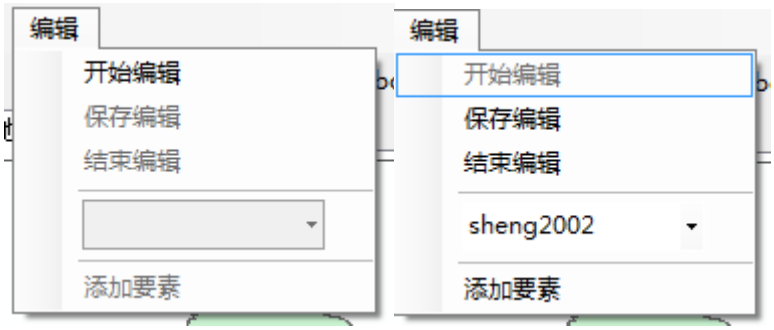
自定义编辑过程，实现“添加要素”功能



1.在主窗体菜单添加“编辑”（ tsmEdit）菜单项，并添加子菜单：开始编辑（tsmStartEdit）、保存编辑（tsmSaveEdit）、结束编辑（tsmEndEdit）、图层选择（cmbSelLayer）、添加要素（tsmAddFeature）
【注：tsm 表示该项为 ToolStripMenuItem，cmb 表示该项为 ToolStripComboBox】

2.定义如下全局变量

```
private IMap pMap = null;
private IActiveView pActiveView = null;
private List<ILayer> plstLayers = null;
private IFeatureLayer pCurrentLyr = null;
private IEngineEditor pEngineEditor = null;
private IEngineEditTask pEngineEditTask = null;
private IEngineEditLayers pEngineEditLayers = null;
```

3.在主窗体的构造函数中，即 public frmMain()中，调用 InitObject()方法，并实现该方法

```
public frmMain()
{
                InitializeComponent();
                InitObject();
}

private void InitObject()
{
     try   {
                ChangeButtonState(false);
                pEngineEditor = new EngineEditorClass();
                MapManager.EngineEditor = pEngineEditor;
                pEngineEditTask = pEngineEditor as IEngineEditTask;
                pEngineEditLayers = pEngineEditor as IEngineEditLayers;
            }
     catch (Exception ex)
          {
                MessageBox.Show(ex.ToString());
          }
 }
```

4.实现“改变按钮状态”方法：ChangeButtonState()

```
 private void ChangeButtonState(bool bEnable)
{
            tsmStartEdit.Enabled = !bEnable;
            tsmSaveEdit.Enabled = bEnable;
            tsmEndEdit.Enabled = bEnable;
            cmbSelLayer.Enabled = bEnable;
            tsmAddFeature.Enabled = bEnable;
 }
```

5.实现“初始化下拉列表框”方法：InitComboBox()

```
 private void InitComboBox(List<ILayer> plstLyr)
{
           cmbSelLayer.Items.Clear();
           for (int i = 0; i < plstLyr.Count; i++)
           {
               if (!cmbSelLayer.Items.Contains(plstLyr[i].Name))
               {  cmbSelLayer.Items.Add(plstLyr[i].Name);  }
           }
           if (cmbSelLayer.Items.Count != 0) cmbSelLayer.SelectedIndex = 0;
 }
```

6.实现“开始编辑”的 click 事件

```
 private void tsmStartEdit_Click(object sender, EventArgs e)
 {
  try {   pMap = axMapControl1.Map;
          pActiveView = pMap as IActiveView;
          plstLayers = MapManager.GetLayers(pMap);
          if (plstLayers == null || plstLayers.Count == 0)  {
            MessageBox.Show("请加载编辑图层！", "提示", MessageBoxButtons.OK,
                                                    MessageBoxIcon.Information);
            return;  }
           pMap.ClearSelection();       pActiveView.Refresh();
           InitComboBox(plstLayers);    ChangeButtonState(true);
           //如果编辑已经开始，则直接退出
           if (pEngineEditor.EditState != esriEngineEditState.esriEngineStateNotEditing)
                   return;
           if (pCurrentLyr == null) return;
           //获取当前编辑图层工作空间
           IDataset pDataSet = pCurrentLyr.FeatureClass as IDataset;
           IWorkspace pWs = pDataSet.Workspace;
           //设置编辑模式，如果是 ArcSDE 采用版本模式
          if (pWs.Type == esriWorkspaceType.esriRemoteDatabaseWorkspace)
           { pEngineEditor.EditSessionMode =
                       esriEngineEditSessionMode.esriEngineEditSessionModeVersioned;
           } else {
                       pEngineEditor.EditSessionMode =
                       esriEngineEditSessionMode.esriEngineEditSessionModeNonVersioned; }
```

```
            //设置编辑任务
            pEngineEditTask = pEngineEditor.GetTaskByUniqueName
                                        ("ControlToolsEditing_CreateNewFeatureTask");
            pEngineEditor.CurrentTask = pEngineEditTask;// 设置编辑任务
            pEngineEditor.EnableUndoRedo(true); //是否可以进行撤销、恢复操作
            pEngineEditor.StartEditing(pWs, pMap); //开始编辑操作
        }
     catch (Exception ex ){  MessageBox.Show(ex.ToString());   }
   }
```

7.实现 cmbSelectLayer 的 SelectedIndexChanged 事件

```
  private void cmbSelectLayer_SelectedIndexChanged(object sender, EventArgs e)
   {
         try {   string sLyrName = cmbSelLayer.SelectedItem.ToString();
                 pCurrentLyr = MapManager.GetLayerByName(pMap, sLyrName) as IFeatureLayer;
                 //设置编辑目标图层
                 pEngineEditLayers.SetTargetLayer(pCurrentLyr, 0);
             }
        catch (Exception ex) {  MessageBox.Show(ex.ToString());  }
    }

```

8.实现“保存编辑”菜单项的 CLICK 事件：tsmSaveEdit_Click

```
private void tsmSaveEdit_Click(object sender, EventArgs e)
   {  try {
         ICommand m_saveEditCom = new SaveEditCommandClass();
         m_saveEditCom.OnCreate(axMapControl1.Object);
         m_saveEditCom.OnClick();
         axMapControl1.MousePointer = esriControlsMousePointer.esriPointerDefault;
         MessageBox.Show("保存成功");
         }
        catch (Exception ex) {   }
   }
```

9.实现“结束编辑”菜单项的 CLICK 事件：tsmEndEdit_Click

```
private void tsmEndEdit_Click(object sender, EventArgs e)
   { try {
            axMapControl1.CurrentTool = null;
            axMapControl1.MousePointer = esriControlsMousePointer.esriPointerDefault;
            ChangeButtonState(false);
            ICommand m_stopEditCom = new StopEditCommandClass();
            m_stopEditCom.OnCreate(axMapControl1.Object);
            m_stopEditCom.OnClick();
          }
      catch (Exception ex) {   }
}
```

10.实现“添加要素”菜单项的 CLICK 事件：tsmAddFeature_Click

```
  private void tsmAddFeature_Click(object sender, EventArgs e)
  {  try  {
              ICommand m_CreateFeatTool = new CreateFeatureToolClass();
```

```
            m_CreateFeatTool.OnCreate(axMapControl1.Object);
            m_CreateFeatTool.OnClick();
            axMapControl1.CurrentTool = m_CreateFeatTool as ITool;
            axMapControl1.MousePointer = esriControlsMousePointer.esriPointerCrosshair;
           }
     catch (Exception ex) {   }
 }
```

11. 实现 SaveEditCommandClass 命令

右键项目，添加→类，将类命名为“SaveEditCommandClass”，该类将实现 ICommand 接口，代码如下：

```
using System;   using System.Collections.Generic;       using System.Linq;
using System.Text;      using ESRI.ArcGIS.SystemUI;     using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Controls;     using ESRI.ArcGIS.Geodatabase;
using System.Windows.Forms;     using ESRI.ArcGIS.Display;
namespace WindowsFormsApplication1 {
   class SaveEditCommandClass:ICommand    {
        private IMap m_Map = null;        private bool bEnable = true;
        private IActiveView m_activeView = null;  private IHookHelper m_hookHelper = null;
        private IEngineEditor m_EngineEditor = null;
        #region ICommand 成员
        public int Bitmap        {      get { return -1; }               }
        public string Caption    {      get { return "保存编辑"; }        }
        public string Category   {      get { return "编辑按钮"; }        }
        public bool Checked      {      get { return false; }             }
        public bool Enabled      {      get { return bEnable; }     }
        public int HelpContextID {      get { return -1; }        }
        public string HelpFile   {      get { return ""; }        }
        public string Message    {      get { return "保存编辑过程所做的操作"; }}
        public string Name       {      get { return "SaveEditCommand"; }    }
 public void OnClick()
 {
    m_Map = m_hookHelper.FocusMap;    m_activeView = m_Map as IActiveView;
    m_EngineEditor = MapManager.EngineEditor;
    if (m_EngineEditor == null) return;
    if (m_EngineEditor.EditState != esriEngineEditState.esriEngineStateEditing)
                return;
    IWorkspace pWs = m_EngineEditor.EditWorkspace;
    Boolean bHasEdit = m_EngineEditor.HasEdits();
    if (bHasEdit) {
        if (MessageBox.Show("是否保存所做的编辑？", "提示", MessageBoxButtons.YesNo,
                            MessageBoxIcon.Information) == DialogResult.Yes)
                { m_EngineEditor.StopEditing(true);
                  m_EngineEditor.StartEditing(pWs, m_Map);
                  m_activeView.Refresh();      }
    }
 }
public void OnCreate(object Hook)  {
```

```
      if (Hook == null) return;
      try  {     m_hookHelper = new HookHelperClass();
                 m_hookHelper.Hook = Hook;
                 if (m_hookHelper.ActiveView == null)        m_hookHelper = null;    }
     catch{ m_hookHelper = null;        }
    if (m_hookHelper == null)  bEnable = false;  else   bEnable = true;
}
public string Tooltip  {   get { return "保存编辑过程所做的操作"; }        }
        #endregion
    }
}
```

12. 实现 StopEditCommandClass 命令

右键项目，添加→类，将类命名为“StopEditCommandClass”，该类将实现 ICommand 接口，代码如下：

```
using System;   using System.Collections.Generic;       using System.Linq;
using System.Text;      using ESRI.ArcGIS.SystemUI;     using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Controls; using System.Windows.Forms;  using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Display;
namespace WindowsFormsApplication1
{
    class StopEditCommandClass:ICommand
    {
        private IMap m_Map = null;        private bool bEnable = true;
        private IActiveView m_activeView = null;
        private IHookHelper m_hookHelper = null;
        private IEngineEditor m_EngineEditor = null;
        #region ICommand 成员
        public int Bitmap  {   get { return -1; }     }
        public string Caption   {  get { return "停止编辑"; }    }
        public string Category  {  get { return "编辑按钮"; }    }
        public bool Checked     {  get { return false; }  }
        public bool Enabled     {  get { return bEnable; }     }
        public int HelpContextID  {    get { return -1; }       }
        public string HelpFile    {    get { return ""; }       }
        public string Message     {    get { return "停止编辑"; }    }
        public string Name        {    get { return "StopEditCommand"; }  }
        public void OnClick()     {
            m_Map = m_hookHelper.FocusMap;            m_activeView = m_Map as IActiveView;
            m_EngineEditor = MapManager.EngineEditor;            Boolean bSave = true;
            if (m_EngineEditor == null) return;
            if (m_EngineEditor.EditState != esriEngineEditState.esriEngineStateEditing)
                        return;
            IWorkspaceEdit2 pWsEdit2 = m_EngineEditor.EditWorkspace as IWorkspaceEdit2;
            if (pWsEdit2 == null) return;
            if (pWsEdit2.IsBeingEdited())
            {   Boolean bHasEdit = m_EngineEditor.HasEdits();
             if (bHasEdit)   {
```

```
                if (MessageBox.Show("是否保存所做的编辑？", "提示", MessageBoxButtons.YesNo,
                        MessageBoxIcon.Information) == DialogResult.Yes)
                    {   bSave = true;     }
                else {   bSave = false;    }
                                }
                 m_EngineEditor.StopEditing(bSave);
            }
            m_Map.ClearSelection();            m_activeView.Refresh();
        }
        public void OnCreate(object Hook)    {
            if (Hook == null) return;
            try {   m_hookHelper = new HookHelperClass();     m_hookHelper.Hook = Hook;
                    if (m_hookHelper.ActiveView == null)         m_hookHelper = null;
            }
            catch     {   m_hookHelper = null;    }
            if (m_hookHelper == null)     bEnable = false;    else     bEnable = true;
        }
        public string Tooltip  {   get { return "停止编辑"; }     }
        #endregion
    }
}
```

13. 实现 CreateFeatureToolClass 命令

右键项目，添加→类，将类命名为“CreateFeatureToolClass”，该类将实现 ICommand 接口，ITool 接口，代码如下：

```
using System;  using System.Collections.Generic;  using System.Linq;
using System.Text;  using System.Windows.Forms;  using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Carto;  using ESRI.ArcGIS.Controls;  using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Display;  using ESRI.ArcGIS.Geodatabase;
namespace WindowsFormsApplication1
{
    public class CreateFeatureToolClass : ICommand, ITool
    {
        private IMap m_Map = null;        private bool bEnable = true;
        private IHookHelper m_hookHelper = null;
        private IActiveView m_activeView = null;
        private IEngineEditor m_EngineEditor = null;
        private IEngineEditLayers m_EngineEditLayers = null;
        private IPointCollection m_pointCollection;
        private INewLineFeedback m_newLineFeedBack;
        private INewPolygonFeedback m_newPolyFeedBack;
        private INewMultiPointFeedback m_newMultPtFeedBack;
        #region ICommand 成员
        public int Bitmap{ get { return -1; }}
        public string Caption{ get { return "添加要素"; }}
        public string Category{ get { return "编辑工具"; }}
        public bool Checked{  get { return false; }}
        public bool Enabled{ get { return bEnable; }}
```

```
        public int HelpContextID{ get { return -1; }}
        public string HelpFile{ get { return ""; }}
        public string Message{ get { return "添加要素"; }}
        public string Name{  get { return "SketchTool"; }}
        public void OnClick()  {
            m_Map = m_hookHelper.FocusMap;      m_activeView = m_Map as IActiveView;
            m_EngineEditor = MapManager.EngineEditor;
            m_EngineEditLayers = MapManager.EngineEditor as IEngineEditLayers;
        }
        public void OnCreate(object Hook)
        {   if (Hook == null) return;
            try  {
                m_hookHelper = new HookHelperClass();       m_hookHelper.Hook = Hook;
                if (m_hookHelper.ActiveView == null)            m_hookHelper = null;
            }
            catch  {   m_hookHelper = null;    }
            if (m_hookHelper == null)    bEnable = false;  else     bEnable = true;
        }
        public string Tooltip     { get { return "添加要素"; }}
        #endregion
        #region ITool 成员
        public int Cursor{ get { return -1; }}
        public bool Deactivate(){return true;}
        public bool OnContextMenu(int x, int y){ return false;}
        public void OnDblClick()
        {
            IGeometry pResultGeometry = null;
            if (m_EngineEditLayers == null) return;
            //获取编辑目标图层
            IFeatureLayer pFeatLyr = m_EngineEditLayers.TargetLayer;
            if (pFeatLyr == null) return;
            IFeatureClass pFeatCls = pFeatLyr.FeatureClass;
            if (pFeatCls == null) return;
            switch (pFeatCls.ShapeType)
            {   case esriGeometryType.esriGeometryMultipoint:
                    m_newMultPtFeedBack.Stop();
                    pResultGeometry = m_pointCollection as IGeometry;
                    m_newMultPtFeedBack = null;
                    break;
                case esriGeometryType.esriGeometryPolyline:
                    IPolyline pPolyline = null;
                    pPolyline = m_newLineFeedBack.Stop();
                    pResultGeometry = pPolyline as IGeometry;
                    m_newLineFeedBack = null;
                    break;
                case esriGeometryType.esriGeometryPolygon:
                    IPolygon pPolygon = null;
```

```
                pPolygon = m_newPolyFeedBack.Stop();
                pResultGeometry = pPolygon as IGeometry;
                m_newPolyFeedBack = null;
                break;
            }
            CreateFeature(pResultGeometry);  //创建新要素
        }
        public void OnKeyDown(int keyCode, int shift)  {   }
        public void OnKeyUp(int keyCode, int shift)    {  }
        public void OnMouseDown(int button, int shift, int x, int y)
        {
         try  {
        IPoint pPt = m_activeView.ScreenDisplay.DisplayTransformation.ToMapPoint(x, y);
            if (m_EngineEditor == null) return;
            if (m_EngineEditor.EditState != esriEngineEditState.esriEngineStateEditing)
                        return;
            if (m_EngineEditLayers == null) return;
            IFeatureLayer pFeatLyr = m_EngineEditLayers.TargetLayer;
            if (pFeatLyr == null) return;  IFeatureClass pFeatCls = pFeatLyr.FeatureClass;
            if (pFeatCls == null) return;             object missing = Type.Missing;
            m_Map.ClearSelection();
            switch (pFeatCls.ShapeType)   {
                    case esriGeometryType.esriGeometryPoint:
                        //当为点层时，直接创建要素
                        CreateFeature(pPt as IGeometry);    break;
                    case esriGeometryType.esriGeometryMultipoint:
                        //点集的处理方式
                        if (m_pointCollection == null)    {
                            m_pointCollection = new MultipointClass(); }
                        else  {
                            m_pointCollection.AddPoint(pPt, ref missing, ref missing);
                        }
                        if (m_newMultPtFeedBack == null)
                        {   m_newMultPtFeedBack = new NewMultiPointFeedbackClass();
                            m_newMultPtFeedBack.Display = m_activeView.ScreenDisplay;
                            m_newMultPtFeedBack.Start(m_pointCollection, pPt);
                        }   break;
                    case esriGeometryType.esriGeometryPolyline: //多义线处理方式
                        if (m_newLineFeedBack == null)   {
                            m_newLineFeedBack = new NewLineFeedbackClass();
                            m_newLineFeedBack.Display = m_activeView.ScreenDisplay;
                            m_newLineFeedBack.Start(pPt);}
                        else { m_newLineFeedBack.AddPoint(pPt);}
                        break;
                    case esriGeometryType.esriGeometryPolygon: //多边形处理方式
                        if (m_newPolyFeedBack == null)  {
                            m_newPolyFeedBack = new NewPolygonFeedbackClass();
```

```
                        m_newPolyFeedBack.Display = m_activeView.ScreenDisplay;
                        m_newPolyFeedBack.Start(pPt);}
                    else { m_newPolyFeedBack.AddPoint(pPt);}
                    break;
            }
        }
        catch (Exception ex) { }
    }
    public void OnMouseMove(int button, int shift, int x, int y)
    {
       IPoint pPt = m_activeView.ScreenDisplay.DisplayTransformation.ToMapPoint(x, y);
        if (m_EngineEditLayers == null) return;
        //获取编辑目标图层
        IFeatureLayer pFeatLyr = m_EngineEditLayers.TargetLayer;
        if (pFeatLyr == null) return; IFeatureClass pFeatCls = pFeatLyr.FeatureClass;
        if (pFeatCls == null) return;
        switch (pFeatCls.ShapeType)   {
            case esriGeometryType.esriGeometryPolyline:
                if (m_newLineFeedBack != null)
                    m_newLineFeedBack.MoveTo(pPt);
                break;
            case esriGeometryType.esriGeometryPolygon:
                if (m_newPolyFeedBack != null)
                    m_newPolyFeedBack.MoveTo(pPt);
                break;     }
    }
    public void OnMouseUp(int button, int shift, int x, int y) {        }
    public void Refresh(int hdc) {        }
    #endregion
    #region 操作函数
     // 创建要素
    private void CreateFeature(IGeometry pGeometry)
    {  try {
            if (m_EngineEditLayers == null) return;
            IFeatureLayer pFeatLyr = m_EngineEditLayers.TargetLayer;
            if (pFeatLyr == null) return;
            IFeatureClass pFeatCls = pFeatLyr.FeatureClass;
            if (pFeatCls == null) return;      if (m_EngineEditor == null) return;
            if (pGeometry == null) return;
            ITopologicalOperator pTop = pGeometry as ITopologicalOperator;
            pTop.Simplify();
            IGeoDataset pGeoDataset = pFeatCls as IGeoDataset;
            if (pGeoDataset.SpatialReference != null)    {
                pGeometry.Project(pGeoDataset.SpatialReference);     }
            m_EngineEditor.StartOperation();       IFeature pFeature = null;
            pFeature = pFeatCls.CreateFeature();    pFeature.Shape = pGeometry;
```

```
                pFeature.Store();     m_EngineEditor.StopOperation("添加要素");
                m_Map.SelectFeature(pFeatLyr, pFeature);      m_activeView.Refresh();
            }
            catch (Exception ex)      {  }
        }
        #endregion
    }
}
```

14 实现 MapManager 类

右键项目，添加→类，将类命名为“MapManager”，代码如下：

```
using System;  using System.Collections.Generic;  using System.Linq;
using System.Text;  using ESRI.ArcGIS.Carto;  using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Display;  using ESRI.ArcGIS.Geodatabase; using System.Windows.Forms;
using ESRI.ArcGIS.Controls;
namespace WindowsFormsApplication1
{
    class MapManager
    {
        public MapManager(){}
        private static IEngineEditor _engineEditor;
        public static IEngineEditor EngineEditor
        { get { return MapManager._engineEditor; }
          set { MapManager._engineEditor = value; } }
        //根据图层名获取图层
        public static ILayer GetLayerByName(IMap pMap, string sLyrName)
        {   ILayer pLyr = null;            ILayer pLayer = null;
            try{
                for (int i = 0; i < pMap.LayerCount; i++)
                {   pLyr = pMap.get_Layer(i);
                    if (pLyr.Name.ToUpper() == sLyrName.ToUpper())
                    {   pLayer = pLyr;       break;         }
                }
            } catch (Exception ex){}
            return pLayer;
        }
        //获取当前地图文档所有图层集合
        public static List<ILayer> GetLayers(IMap pMap)
        {
            ILayer plyr = null;     List<ILayer> pLstLayers = null;
            try{
                pLstLayers = new List<ILayer>();
                for (int i = 0; i < pMap.LayerCount; i++) {
                    plyr = pMap.get_Layer(i);
                    if (!pLstLayers.Contains(plyr))   {  pLstLayers.Add(plyr);  }
                }
            }
```

```
            catch (Exception ex) { }
            return pLstLayers;
        }
}
```