

# EngineFramework 扩展 开发文档

作者：我不是小菜(jackey)

QQ:563549269

注：本文档只得用于学习交流，不得用于任何商业用途，违者版权必究。

## EngineFramework2.0 框架简介

### (一) 功能简介

EngineFramework(通用地理信息系统插件框架)是基于.Net 框架和 ArcGis Engine9.x 以“平台+ 插件”模型为设计思想进行设计和开发的, 其是解决企业综合 GIS 应用系统开发和集成的基于构件技术的中间件平台, 是能够满足 GIS 应用需求开发的集成的、可伸缩的产品。EngineFramework 提供基于插件构件技术的 GIS 应用整合框架, 采用可视化拖放和插件动态加载配置方式进行系统功能扩展开发, 其大大提高了系统协同开发的效率, 降低了 GIS 应用系统集成技术的难度和系统维护的成本。注: EngineFramework2.0 已改版 Jackey.Framework 核心库, 其提供了任何基于.NET 的桌面应用程序框架。

### (二) EngineFramework 2.0 提供如下功能扩展点

- 工具条按钮插件定制功能;
- 工具条定制功能;
- 右键菜单定制功能;
- 支持 XML 文件菜单配置功能;
- 支持可撤销操作功能扩展;
- 可浮动窗体定制功能;
- ArcToolBox 工具条插件定制功能;
- 组件服务器插件定制功能;
- 自启动服务扩展功能, 如自定义工具条服务和界面服务;
- 编辑扩展模块扩展开发(编辑内核实现了 ArcMap 的 Editor 类, 支持 IEditSketch、IEditTask、ISnapAgent 等内核接口);
- 地图鹰眼定制功能;
- 添加数据对话框扩展功能;
- 地图数据库管理功能;
- 高级插件应用程序框架定制功能(需要扩展 AbstractApplication 和 AbstractDocument 等内核类)。

### (三) EngineFramework 2.0 具有如下特点和优势

- 通用性强

EngineFramework 的通用性主要指其适合任何基于 ArcGIS Engine 和 .NET 平台的或者任意的桌面 GIS 应用软件开发领域, 其为这些应用程序提供统一的开发框架, 开发人员通过系统提供的扩展接口可以开发丰富的 GIS 桌面应用程序, 利用 EngineFramework 提供的强大类库, 开发人员可以定制高伸缩性的应用程序。

➤ 高可扩展性

EngineFramework 的扩展性主要体现在插件和工具条的定制方面, 在其框架支持下, 开发人员可以将插件设计成内嵌的工具按钮项也可以将其设计成独立的工具条, 还可以将其设计成可浮动的窗体或者一个没有用户界面的组件服务器。

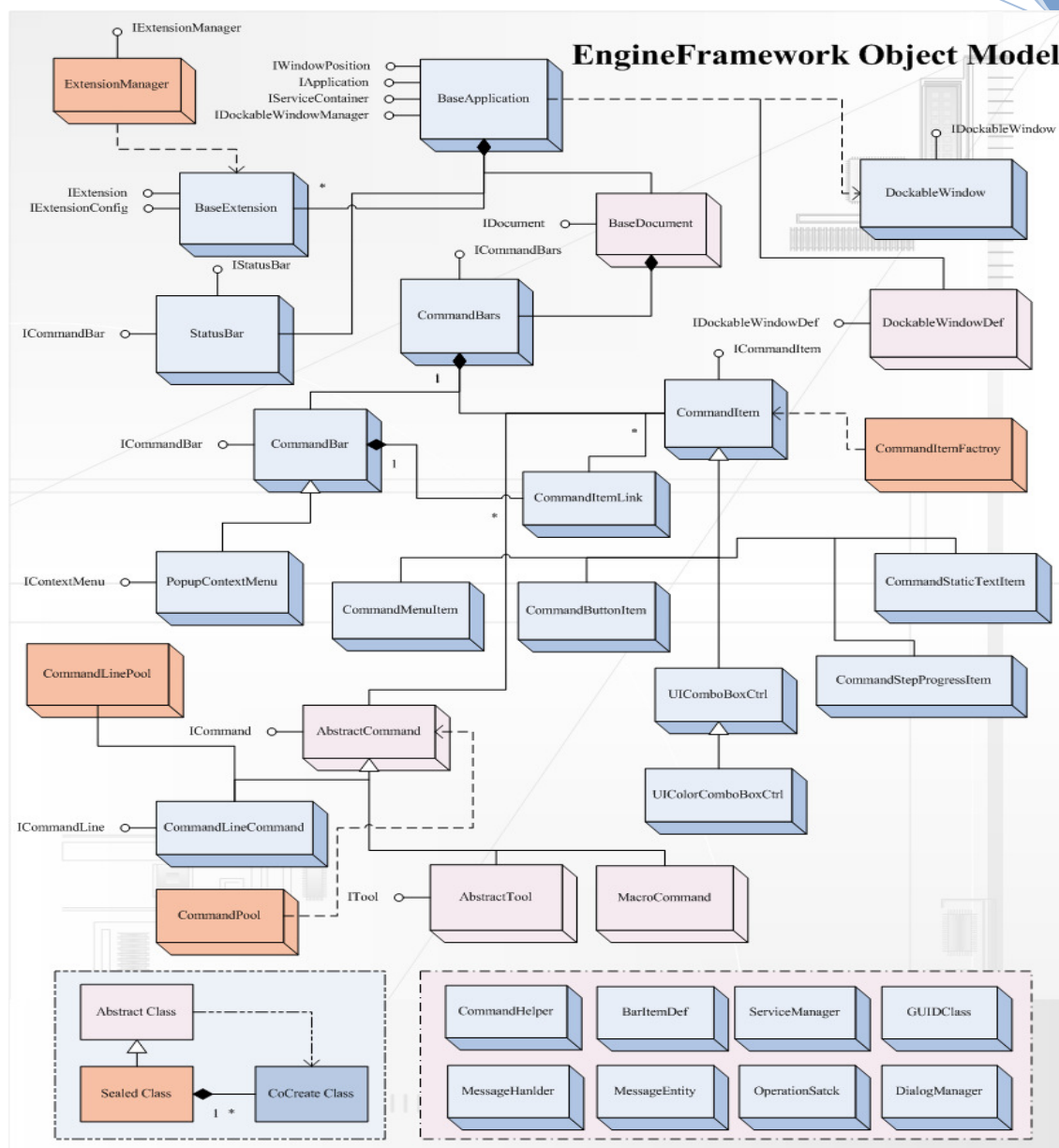
➤ 高可维护性

利用 EngineFramework 的高可扩展性, 应用程序的维护将变得非常简单。例如当用户提出新的需求时, 开发人员可以将新功能模块设计成插件组件, 在保证新插件组件测试完全通过的情况下, 开发人员就可以将此插件组件提交给用户, 用户只需要将其(一个 dll 程序集)放到指定目录即可实现即插即用。在此过程中, 开发人员不需要重新测试整个应用程序和编译整个程序, 更不需要重新打包应用程序, 这样便可以实现系统的远程维护。

➤ 易学易用

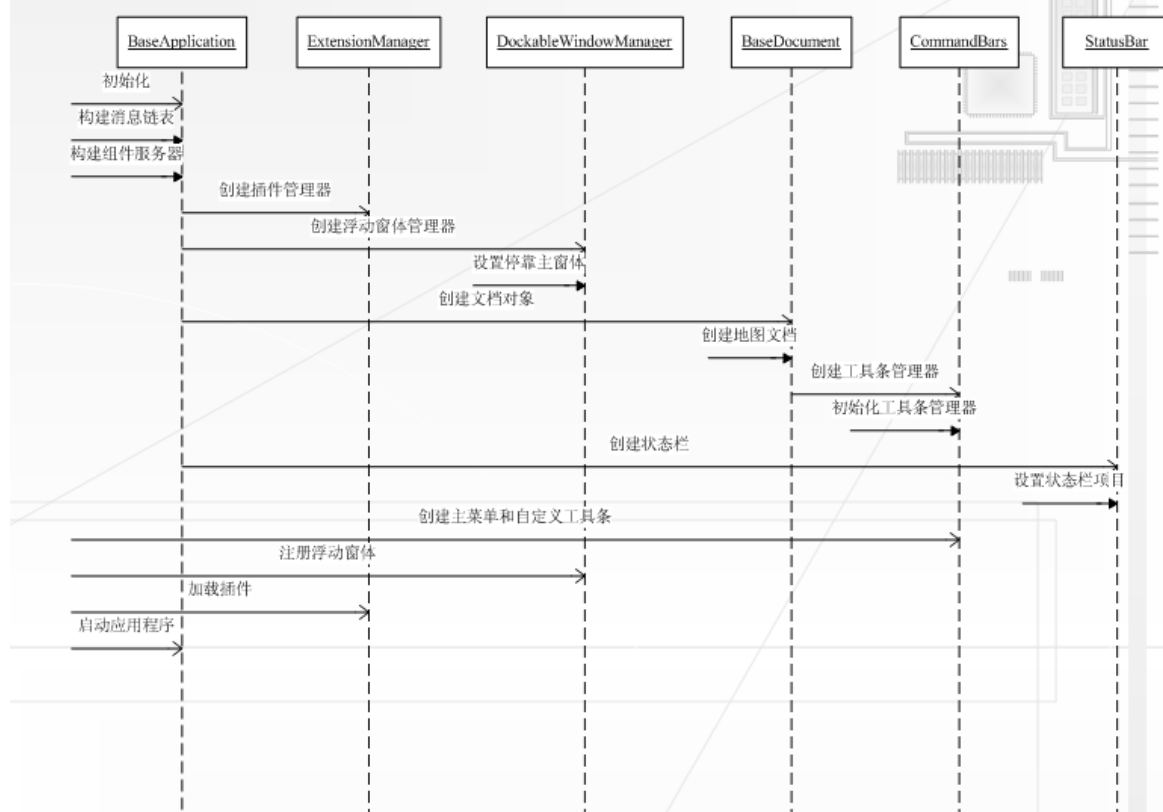
EngineFramework 框架为开发人员提供了一致的编程规范, 这些接口设计简单, 开发人员只需要了解基于接口的编程原则和 OOP 的一些基本概念, 就可以迅速进入开发状态。

#### (四) 内核框架对象模型图 (obsolete)

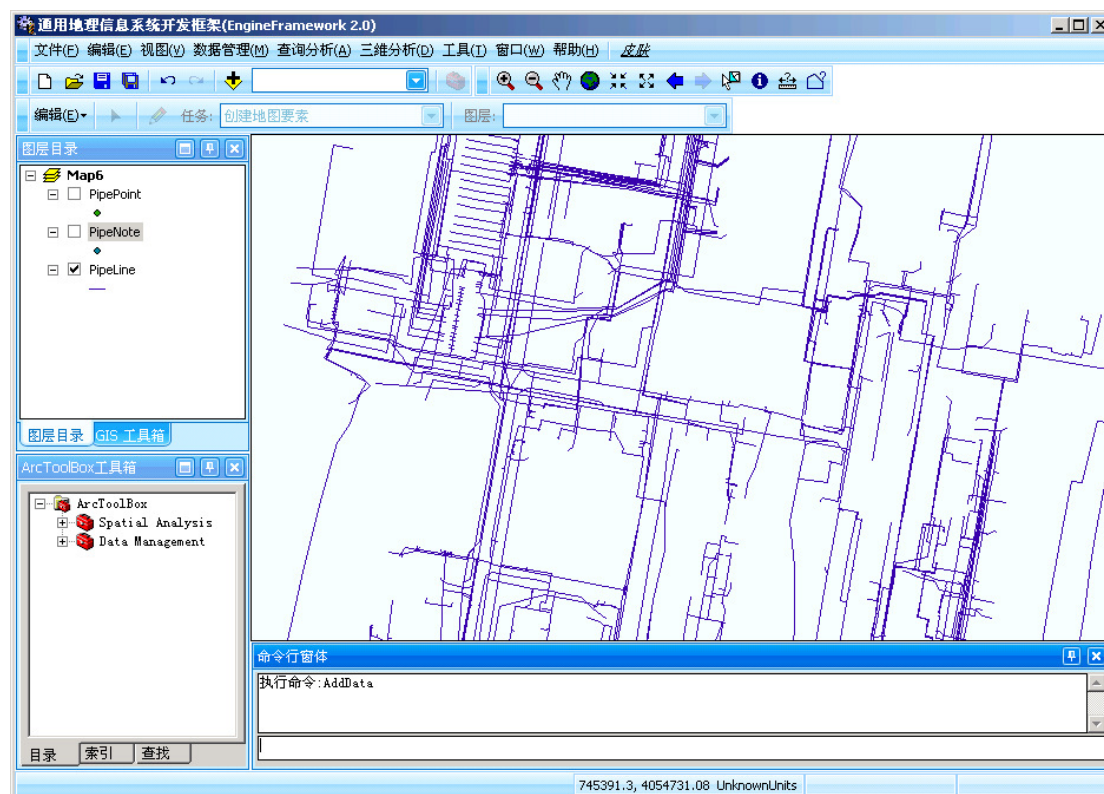


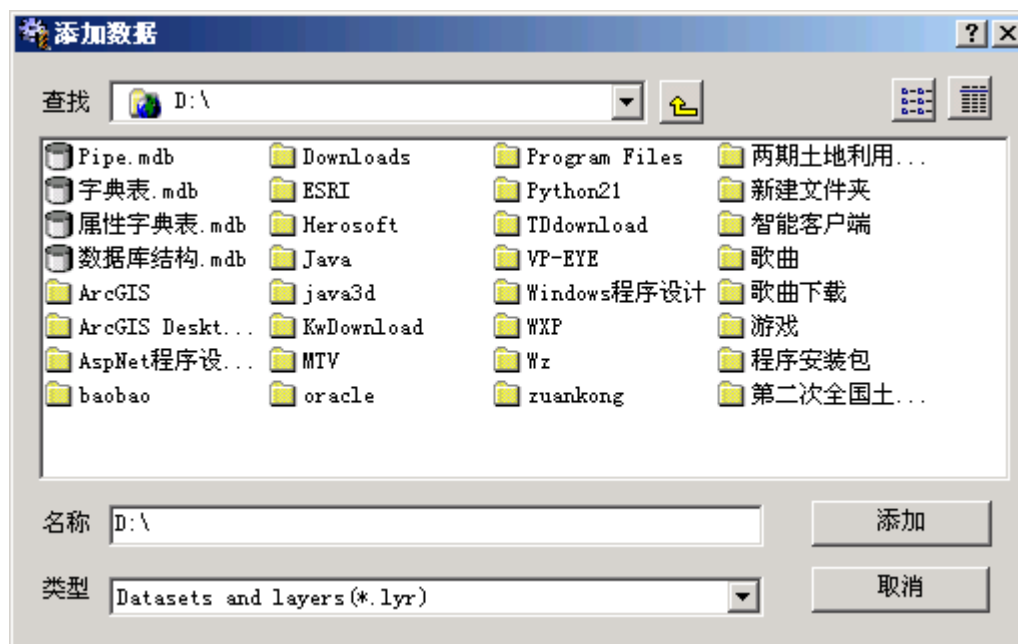
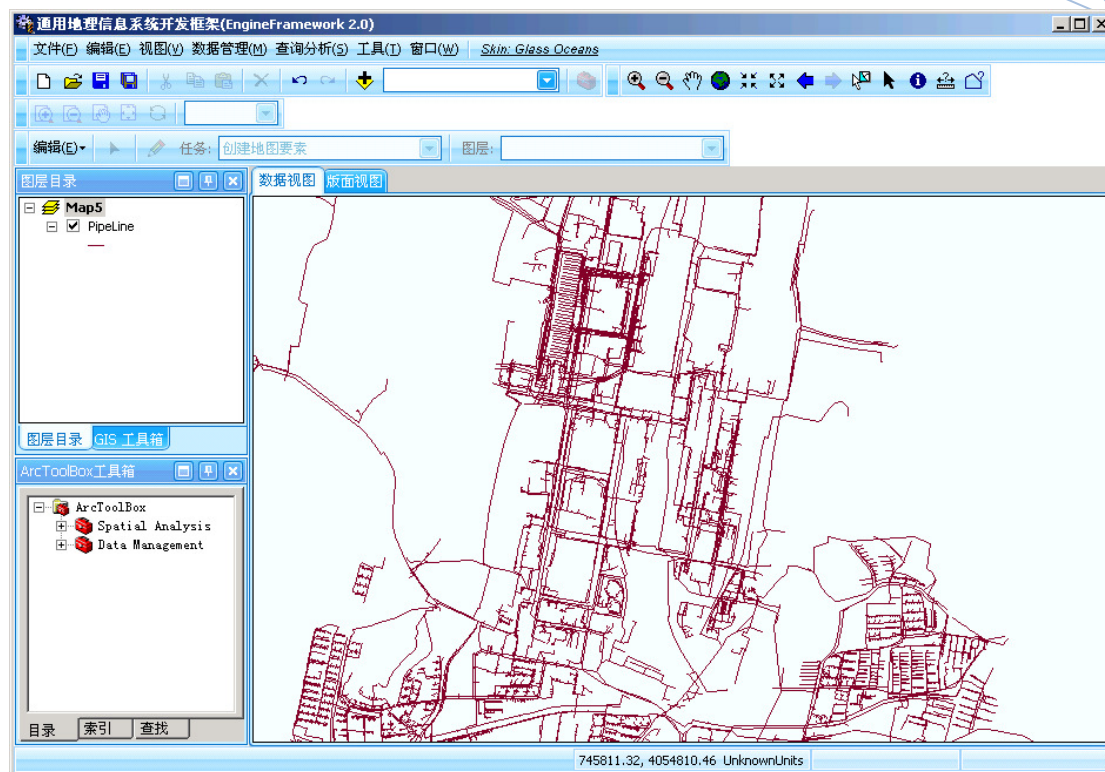
(五) 内核框架启动序列图 (obsolete)

## EngineFramework 框架启动序列图



## (六) 系统定制的 ArcMap 应用程序框架界面





## EngineFramework 框架扩展开发详细文档

本文档主要适用于基于 Jackey.Framework 框架开发的桌面应用程序, 我院以下四个系统都将构建于该框架下, 分别是 SharpMap 应用程序、GoogleMap 应用程序、EngineMap 应用程序和一个 DMS(院设备管理系统)。

Jackey.Framework 是一个封装 DevExpress 控件提供程序统一入口点的一个可扩展的插件框架。利用该框架可以随意扩展应用程序, 因此可以极大的提高系统的可扩展性和维护性。

本文档主要介绍基于 Jackey.Framework 的 EngineMap 应用程序开发示例, 其他应用程序扩展开发可以参考该文档。

### (1) 工具条按钮定制示例:

Jackey.Framework 中要求所有工具按钮都实现 AbstractCommand 抽象类(该类适配了 Jackey.Framework 的 ICommandObject 接口和 ArcGIS Engine 的 ICommand 接口), 对于和 MapControl 或者 PageLayoutControl 交互的按钮还必须实现 AbstractTool 抽象类。这两个抽象类位于 ARSC.ArcEngine.Command.dll 中。

下面分别以两个示例来说明按钮的定制过程。第一个示例的功能是点击一个按钮项时, 将地图窗口放大到原来的两倍大小; 第二个示例的功能是实现和地图控件交互的放大功能(需要和地图控件的交互)。

#### 1) 点击一个按钮项实现地图窗口自动放大两倍效果:

```
/// <summary>
/// 固定放大地图窗口到原来的两倍大小。
/// </summary>
[GlobleID("ARSC.ArcEngine.Command.ArcMap.CommandControlFixedZoomIn")]
public class CommandControlFixedZoomIn : AbstractCommand
{
    private IMxDocument m_mxDoc;

    public CommandControlFixedZoomIn()
    {
        base.m_caption = "中心放大";
        base.m_category = "PanZoomTool";
        base.m_checked = false;
        base.m_enabled = true;
        base.m_message = "中心放大";
    }
}
```



```
base.m_tooltip = "中心放大";
base.m_name = "FixedZoomIn";

//打开资源文件
ResourceManager rm = ResourceServer.GetManager(
    AssemblyInformation.ResourcePath,
    this.GetType().Assembly);

//获取图标
base.m_bitmap =
(System.Drawing.Bitmap)(rm.GetObject("PanZoom_FixedZoomIn"));
}

public override void OnCreate(object hook)
{
    base.OnCreate(hook);
    m_mxDoc = base.m_app.Document as IMxDocument;
}

public override void OnClick()
{
    IActiveView activeView = m_mxDoc.ActiveView;
    IPoint centerPnt = new PointClass();
    centerPnt.PutCoords((activeView.Extent.XMax +
activeView.Extent.XMin) / 2,
        (activeView.Extent.YMax + activeView.Extent.YMin) / 2);

    IEnvelope envelop = activeView.Extent;
    envelop.Expand(0.5, 0.5, true);
    activeView.Extent.CenterAt(centerPnt);
    activeView.Extent = envelop;
    activeView.Refresh();
}
}
```

## 2) 实现和地图控件交互放大效果

```
/// <summary>
/// 拉框放大地图窗口。演示和地图控件的交互。
/// </summary>
[GlobbleID("ARSC.ArcEngine.Command.ArcMap.CommandControlZoomIn")]
public class CommandControlZoomIn : AbstractTool
{
    protected IMxDocument m_mxDoc;
    protected INewEnvelopeFeedback m_envelopeFeedback;
```



```
//拉框起点
protected IPoint m_startPoint = null;

/// <summary>
/// 构造函数执行初始化操作,主要执行图标生成
/// </summary>
public CommandControlZoomIn()
{
    #region 私有成员初始化

    base.m_caption = "放大";
    base.m_category = "PanZoomTool";
    base.m_checked = false;
    base.m_enabled = true;
    base.m_message = "拉框放大, 点击放大倍";
    base.m_tooltip = "放大地图窗口";
    base.m_name = "ZoomIn";
    base.m_IsMouseDown = false;

    //打开资源文件
    ResourceManager rm = ResourceServer.GetManager(
        AssemblyInformation.ResourcePath,
        this.GetType().Assembly);

    //获取图标
    base.m_bitmap =
        (System.Drawing.Bitmap)(rm.GetObject("PanZoom_ZoomIn"));

    base.m_moveCursor = new
        Cursor(Assembly.GetExecutingAssembly().GetManifestResourceStream(
            AssemblyInformation.CursorPath+".PanZoom_Zoomining_Handle.cur"));
    base.m_hMoveCursor = base.m_moveCursor.Handle;

    base.m_defaultCursor = new
        Cursor(Assembly.GetExecutingAssembly().GetManifestResourceStream(
            AssemblyInformation.CursorPath+".PanZoom_Zoomin_Handle.cur"));
    base.m_hDefaultCursor = base.m_defaultCursor.Handle;

    #endregion
}
```

```
public override void OnCreate(object hook)
{
    base.OnCreate(hook);

    m_mxDoc = base.m_app.Document as IMxDocument;
}

#region ITool 成员

protected override void Reset()
{
    base.Reset();
    m_envelopeFeedback = null;
    m_startPoint = null;
}

public override void OnMouseDown(int button, int shift, int x, int
y)
{
    //右键退出
    if (button != 1) return;

    //清除上一次的状态
    if (IsMouseDown())
        Reset();

    IActiveView activeView = this.m_mxDoc.ActiveView;

    if (activeView is IPageLayout)
    {
        IPoint pnt =
activeView.ScreenDisplay.DisplayTransformation.ToMapPoint(x, y);
        IMap map = activeView.HitTestMap(pnt);
        if (map == null) return;

        if (map != activeView.FocusMap)
        {
            activeView.FocusMap = map;

            activeView.PartialRefresh(esriViewDrawPhase.esriViewGraphics, null,
null);
        }
    }
    m_startPoint =
```

```
activeView.ScreenDisplay.DisplayTransformation.ToMapPoint(x, y);

    base.m_IsMouseDown = true;
}

public override void OnMouseMove(int button, int shift, int x, int
y)
{
    //右键退出
    if (button != 1)
        return;

    IActiveView activeView = this.m_mxDoc.ActiveView;

    if (IsMouseDown() && m_startPoint != null)
    {
        //开始拉框
        if (m_envelopeFeedback == null)
        {
            m_envelopeFeedback = new NewEnvelopeFeedback();
            m_envelopeFeedback.Display =
activeView.ScreenDisplay;
            m_envelopeFeedback.Start(m_startPoint);
        }

        //更新矩形框
        IPoint pntMoveTo =
activeView.ScreenDisplay.DisplayTransformation.ToMapPoint(x, y);
        m_envelopeFeedback.MoveTo(pntMoveTo);
    }
}

public override void OnMouseUp(int button, int shift, int x, int
y)
{
    //右键退出
    if (button != 1) return;

    IActiveView activeView = this.m_mxDoc.ActiveView;

    if (IsMouseDown() && m_startPoint != null)
    {
        IEnvelope envelope;
```

```
//只是鼠标点击,没有拉框,则以点击点为中心放大
if (m_envelopeFeedback == null)
{
    envelope = activeView.Extent;
    envelope.CenterAt(m_startPoint);
    envelope.Expand(0.5, 0.5, true);
}
else//拉框放大
    envelope = m_envelopeFeedback.Stop();

double width, height;
width = envelope.Width;
height = envelope.Height;

//如果范围有效,则将其设置为当前地图窗口
if (width != 0 && height != 0)
{
    activeView.Extent = envelope;
    activeView.Refresh();
}
}

//重置状态
Reset();
}

public override void OnKeyDown(int keyCode, int shift)
{
    if (IsMouseDown())
    {
        //按下Esc键取消操作
        if (keyCode == 27)
        {
            Reset();
        }
    }

    this.m_mxDoc.ActiveView.PartialRefresh(esriViewDrawPhase.esriViewForeground, null, null);
}

}

public override bool Enabled
{

```

```
get
{
    //当地图窗口具有数据时,才可以使用该按钮项
    IActiveView activeView = this.m_mxDoc.ActiveView;
    if (activeView.FocusMap.LayerCount > 0)
        return true;

    return false;
}

}

#endregion
}
```

### 3) 按钮面板开发示例

```
/// <summary>
/// 绘制图形。
/// </summary>

[GlobleID("ARSC.ArcEngine.Command.ArcMap.CommandControlDrawElement", CommandTypeEnum.PaletteItem)]
public class CommandControlDrawElement :
AbstractCommand , ISupportInitialize
{
    private IMxApplication _mxApp;
    public CommandControlDrawElement()
    {
        base.m_message = "绘制图形要素.可以绘制点、线、面、曲线等要素";
    }

    void OnActiveCommandChangedHandler(object sender,
        EventArgs e)
    {
        OnClick();
    }

    #region 重写的成员

    public override void OnCreate(object hook)
    {
        base.OnCreate(hook);
        _mxApp = base.m_app as IMxApplication;
    }
}
```

```
public override void OnClick()
{
    CommandPaletteItem cmdItem = this.EventSenderObject
as CommandPaletteItem;
    if (cmdItem.ActiveCommand != null){
        ITool tool = cmdItem.ActiveCommand as ITool;
        if (tool != null)
            _mxApp.CurrentTool = tool;
        else
            cmdItem.ActiveCommand.OnClick();
    }
}

public override string Message
{
    get
    {
        CommandPaletteItem cmdItem =
this.EventSenderObject as CommandPaletteItem;
        if (cmdItem.ActiveCommand !=null)
            return cmdItem.ActiveCommand.Message;
        return base.Message;
    }
}

#endregion
```

**#region ISupportInitialize 成员**

```
public void Initialize()
{
    CommandPaletteItem cmdItem = this.EventSenderObject as
    CommandPaletteItem;
    cmdItem.AddCommand(new
        CommandControlDrawRectangleElement(), true);
    cmdItem.AddCommand(new
        CommandControlDrawPolygonElement());
    cmdItem.AddCommand(new
        CommandControlDrawCircleElement());
    cmdItem.AddCommand(new
        CommandControlDrawLineElement());
    cmdItem.AddCommand(new
        CommandControlDrawMarkerElement());
}
```

```
//这里是直接包装ArcGIS Engine的命令
//这允许我们可以在Jackey.Framework中可以直接利用已有的
//ArcGIS Engine的内置按钮功能，可以极大的提高开发速度。
cmdItem.AddCommand(CommandProxy.CreateProxy(new
    ESRI.Controls.ControlsAlignMiddleCommand()));

cmdItem.ReCalculateLayout();
cmdItem.ActiveCommandChanged += new
    EventHandler(OnActiveCommandChangedHandler);
cmdItem.OnCreate();
}

#endregion
}
}
```

从以上三个示例中可以看出，定制工具项按钮功能就是如何实现 AbstractCommand 和 AbstractTool 的问题。在实现时需要注意以下几点问题：

- 1) 所有定制按钮都必须定义 GlobeID 特性，该特性将用于该按钮项的注册和查找，默认的按钮项目都被注册为 CommandButtonItem 类型，如果需要注册为其他类型请显示设置 GlobeID 的 CommandType 属性。其值可以为 CommandButtonItem\MenuItem\StaticTextItem\ComboBoxItem\ColorComboBoxItem\EditBoxItem\StepProgressItem\PopupWindowItem\PaletteItem。
- 2) 定制按钮必须在构造函数中填写该项的一些界面信息，如按钮标题和图标等。
- 3) 如果该按钮项具有非托管资源，需要重写 Dispose(bool disposing) 方法，以便顺利清除资源，对于和 ArcGIS Engine 控件有事件交互的必须在 Dispose 显示注销该事件，以防资源泄露。
- 4) 如果需要和事件触发者进行通讯，可以利用 base.



EventSenderObject 来获取事件触发者对象。如  
CommandButtonItem sender=base.EventSenderObject as  
CommandButtonItem。

- 5) 对于需要实现初始化的按钮，如面板按钮，可以实现  
ISupportInitialize 接口，在 Initialize 方法中写入初始化方  
法。

定制功能开发完成后，就需要将其注册到系统中，可以通过  
CommandPool.RegisterCommandItme 方法在运行时对按钮进行注册。这样  
注册按钮具有一定的弊端，因为必须在程序启动或者插件加载前显示调用该方法。  
Jackey.Framework 提供了基于配置文件的注册方法，等按钮项开发完成后，  
将其添加到应用程序配置文件中，即可实现自动注册。该方法的原理是当主程序  
启动时，系统自动读取配置文件，并且对其中配置的按钮项执行注册。下面是注  
册后的配置表内容(红色部分)。

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="commandSetup" type="Jackey.Framework.CommandSection,
Jackey.Framework, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null" />
  </configSections>
  <commandSetup>
    <commands>
      <add
CommandID="ARSC.ArcEngine.Command.ArcMap.CommandControlFixedZoomIn"
AssemblyPath="\ARSC.ArcEngine.Command.dll" />
      <add
CommandID="ARSC.ArcEngine.Command.ArcMap.CommandControlZoomIn"
AssemblyPath="\ARSC.ArcEngine.Command.dll" />
    </commands>
  </commandSetup>
</configuration>
```

```
</commands>

</commandSetup>

<configuration>
```

## (2) 工具条定制示例:

工具条是基于工具按钮项进行定制的, 因此, 预进行工具条的定制, 必须首先定制工具按钮功能项。下面给出一个工具条定制示例:

```
/// <summary>
/// 视图工具栏。
/// </summary>
public class ViewBarDef:IBarDef
{
    #region IBarDef 成员

    public string Caption
    {
        get { return "视图工具栏"; }
    }

    public int ItemCount
    {
        get { return 4; }
    }

    public string Name
    {
        get { return "ViewToolBar"; }
    }

    public void SetItemInfo(int pos, ref IBarItemDef barItemDef)
    {
        GuidClass uid = new GuidClass();

        switch (pos)
        {
            case 0://放大
                uid.Value =
"ARSC.ArcEngine.Command.ArcMap.CommandControlZoomIn";
                barItemDef.CommandID = uid;
                break;
            case 1://缩小
                uid.Value =
```

```
"ARSC.ArcEngine.Command.ArcMap.CommandControlZoomOut";
        barItemDef.CommandID = uid;
        break;
    case 2://漫游
        uid.Value =
"ARSC.ArcEngine.Command.ArcMap.CommandControlPan";
        barItemDef.CommandID = uid;
        break;
    case 3://要素选择
        uid.Value =
"ARSC.ArcEngine.Command.ArcMap.CommandControlFeatureSelect";
        barItemDef.CommandID = uid;
        barItemDef.BeginGroup = true;
        break;
    }
}

#endregion
}
```

工具条定制开发完成后就可以在程序启动时或者插件加载时将其添加到主程序中，可以通过如下方法进行按钮和工具条的加载：

### 1) 加载单独工具按钮项：

```
IcommandBars cmdBars=_app.Document.CommandBars;

IcommandBar viewToolBar=cmdBars.CreateBar("ViewToolBar", "视图工具栏",
CommandBarTypeEnum.CommonToolbar, BarDockStyleEnum.Top);

//添加放大工具到工具条
GuidClass uid = new GuidClass();
uid.Value = "ARSC.ArcEngine.Command.ArcMap.CommandControlZoomIn";
viewToolBar.AddCommandItem(uid);

//添加缩小工具到工具条
uid.Value = "ARSC.ArcEngine.Command.ArcMap.CommandControlZoomOut";
viewToolBar.AddCommandItem(uid);
```

### 2) 加载定制工具条：

```
IcommandBars cmdBars=_app.Document.CommandBars;
IcommandBar viewToolBar= cmdBars.CreateBar(new ViewBarDef());
```

### (3) 浮动窗体定制示例:

系统所有的浮动窗体都是由 Jackey.Framework 统一进行管理的, 系统通过 IDockableWindowManager 接口用于浮动窗体的注册查找以及卸载功能。浮动窗体首先必须由 IDockableWindowDef 接口进行定义, 该接口中有一个 DockableControl 属性, 在定制开发时, 通过将实际的控件(UserControl)或者窗体通过该属性返回即可。建议定制窗体界面元素利用 UserControl 进行定制。下面以一个例子进行说明。该例子主要定制了一个 ArcToolBox 的工具箱浮动窗体。

示例代码如下:

```
/// <summary>
/// 定义ArcToolBox工具箱窗体。
/// </summary>
[GlobbleID("ARSC.ArcEngine.ArcToolBox.ArcToolBoxWindowDef")]
public sealed class ArcToolBoxWindowDef:IDockableWindowDef
{
    //自定义UserControl控件类型。
    private ToolBoxControl _toolBoxCtrl;

    /// <summary>
    /// 构造函数。
    /// </summary>
    public ArcToolBoxWindowDef()
    {
        _toolBoxCtrl = new ToolBoxControl();
    }

    /// <summary>
    /// 利用toolBoxPath目录中的程序集文件生成ArcToolBox目录树。
    /// </summary>
    /// <param name="toolBoxPath">插件工具箱文件夹路径</param>
    public void InitializeArcToolBox(string toolBoxPath)
    {
        if (toolBoxPath == null || toolBoxPath.Length == 0)
            throw new ArgumentNullException("toolBoxPath can't be null!");
    }

    //代码已经省略
}
```

```
}

#region IDockableWindowDef 成员

/// <summary>
/// 获取工具箱标题。
/// </summary>
public string Caption
{
    get { return "ArcToolBox工具箱"; }
}

/// <summary>
/// 获取工具箱控件。
/// </summary>
public Control DockableControl
{
    get { return this._toolBoxCtrl; }
}

/// <summary>
/// 获取工具箱名称。
/// </summary>
public string Name
{
    get { return "ArcToolBox"; }
}

/// <summary>
/// 工具箱初始化时执行, 传递钩子。
/// </summary>
/// <param name="hook">钩子对象</param>
public void OnCreate(object hook)
{
    //
    //hook 是一个IApplication 接口类型
    //
}

/// <summary>
/// 工具箱销毁时执行该操作。
/// </summary>
public void OnDestroy()
{

```

```
        this._toolBoxCtrl = null;
    }

    /// <summary>
    /// 获取用户自定义数据，默认返回空。
    /// </summary>
    public object UserData
    {
        get { return null; }
    }

    #endregion
}
```

浮动窗体定制完成后需要进行注册，注册是通过运行时代码进行注册的，注册方法如下：

```
IDockableWindowManager dockWinManager
    = this.m_app as IDockableWindowManager;
ArcToolBoxWindowDef toolBoxDef = new ArcToolBoxWindowDef();

//注册ArcToolBox浮动窗体
IDockableWindow arcToolBoxWindow
=dockWinManager.RegisterDockableWindow(toolBoxDef);

//初始化工具箱
toolBoxDef.InitializeArcToolBox(Application.StartupPath +
"\\ArcToolBox");

//浮动为Tab模式
arcToolBoxWindow.DockTo(tocWindow, DockFlagsEnum.DockAsTab);
```

**注意：**开发浮动窗体时，也需要定义一个 GlobeID 特性用于唯一标示该浮动窗体。

#### (4) 目录内容窗体定制示例：

目录内容窗体(ContentsView)大都实现为浮动窗体，因此其实现和浮动窗体基本一致，由于其和 IActiveView 进行交互，所以其必须实现 IContentsView 接口，这里不再举例，框架中 Toc 视图即实现为目录内容窗体。

#### (5) ArcToolBox 工具箱定制开发示例：

ArcToolBox 内部也实现为插件机制。因此，可以对其进行插件扩展，从而丰富应用程序的功能。扩展 ArcToolBox 工具箱非常简单，首先从

BaseToolBoxItem 基类型定制用户界面, 重写其中 Execute() 方法, 实现具体的功能。定制开发后, 需要配置一个 XML 文件, 该文件描述定制后其在 ArcToolBox 树结点中的显示内容。下面是一个配置例子和界面示例。

```
<?xml version="1.0" encoding="utf-8"?>

<ArcToolBox>

  <ToolSet caption="Spatial Analysis">

    <ToolItem caption ="Buffer Analysis"
class="ARSC.ArcEngine.ToolBox.Analysis.BufferAnalysisToolItem"/>

    <ToolItem caption ="Clip Analysis"
class="ARSC.ArcEngine.ToolBox.Analysis.ClipAnalysisToolItem"/>

  </ToolSet>

  <ToolSet caption="Data Management">

    <ToolSet caption="subset">

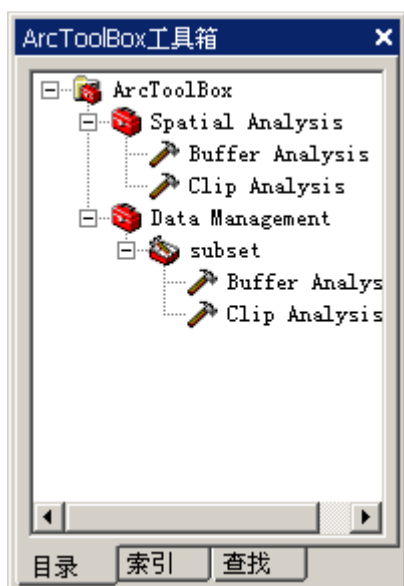
      <ToolItem caption ="Buffer Analysis2"
class="ARSC.ArcEngine.ToolBox.Analysis.BufferAnalysisToolItem"/>

      <ToolItem caption ="Clip Analysis"
class="ARSC.ArcEngine.ToolBox.Analysis.ClipAnalysisToolItem"/>

    </ToolSet>

  </ToolSet>

</ArcToolBox>
```





该模式扩展开发比较简单,需要注意的地方就是配置文件的中 class 项,该项应该是类型全名。此外要求配置文件必须的命名必须是其程序集名称(去掉 dll)+ “ToolBox.xml” 的文件。

### (6) ToolBox 工具箱定制开发示例:

ToolBox 工具箱和 ArcToolBox 工具箱实现原理基本一致,只不过其表现为不同的界面风格。开发时需要从 ToolBoxItem 抽象类型进行继承,重写其中 Execute 方法即可。同样,其开发完成后必须保证在其程序集目录下具有一个配置文件,配置文件命名和 ArcToolBox 配置文件命名相同。下面给出配置文件和界面:

```
<?xml version="1.0" encoding="utf-8"?>

<!--GIS 插件工具箱-->

<ToolBoxSet ResourceName="ToolBox.CartographyLib.myResource">

  <ToolBox GroupName="CartoGraphy" GroupCaption="专题制图"
GroupLargeImage="" GroupSmallImage="CartoGraphy"
GroupStyle="LargeIconsList" Expanded="true">

    <ToolBoxItem ItemName="SimpleRender" ItemCaption="单一专题地图制作"
ItemClass="ToolBox.CartographyLib.SimpleRenderToolItem"
ItemLargeImage="SimpleRender"/>

    <ToolBoxItem ItemName="UniqueValueRender" ItemCaption="分类专题地图制作"
ItemClass="ToolBox.CartographyLib.UniqueValueRenderToolItem"
ItemLargeImage="UniqueValueRender"/>

    <ToolBoxItem ItemName="ClassBreaksRender" ItemCaption="分级专题地图制作"
ItemClass="ToolBox.CartographyLib.ClassBreakRenderToolItem"
ItemLargeImage="ClassBreaksRender"/>

  </ToolBox>

  <ToolBox GroupName="QueryUtil" GroupCaption="统计查询"
```

```
GroupLargeImage="" GroupSmallImage="QueryGroup"

GroupStyle="LargeIconsList" Expanded="true">

    <ToolBoxItem ItemName="SqlQuery" ItemCaption="SQL 属性查询"

ItemClass="ToolBox.CartographyLib.SqlQueryToolItem"

ItemLargeImage="SqlQuery"/>

    <ToolBoxItem ItemName="StringQuery" ItemCaption="地名数据查询"

ItemClass="ToolBox.CartographyLib.StringQueryToolItem"

ItemLargeImage="StringQuery"/>

</ToolBox>

</ToolBoxSet>
```



### (7) 高级插件开发示例:

在 Jackey.Framework 框架下, 可以将插件设计成工具条也可以设计成可浮动的窗体, 甚至还可以设计成一个没有用户界面的组件服务对象。如设计一个 TimerService 服务类, 该类主要为 Jackey.Framework 提供状态栏时间显示功能。插件开发是 Jackey.Framework 功能扩展的支撑点, 为了使插件开发变得简单, Jackey.Framework 提供了抽象基类 BaseExtension, 该类主要维护插件内部状态

和负责插件的初始化操作。如果查看对象模型图，下面给出一个示例，该示例利用插件实现一个工具栏，即当插件加载时，其为主框架生成一个工具栏。

```
/// <summary>
/// 三维分析扩展模块。
/// </summary>
[GlobbleID("ARSC.ArcEngine.DDDAnalysis.DDDEExtension")]
[LicenseProvider(typeof(ComplexLicenseProvider))]
public class DDDEExtension : BaseExtension, INameParserService
{
    /// <summary>
    /// when the extension is loaded,the application framework will
    call this method.
    /// and you should call initialization method here.e.g,register
    command item,register
    /// dockable window ,or create command bar and any thing you like.
    /// </summary>
    /// <remarks>when the application is loading this extension,this
    method will be called automatically.</remarks>
    /// <param name="initializationData">the initialize
    data,commonly it will be the the interface of IApplication.</param>
    public override void Startup(object initializationData)
    {
        //you must call the base.Startup method first.
        base.Startup(initializationData);

        //create the specified command bar.
        ICommandBars cmdBars = base._app.Document.CommandBars;
        cmdBars.CreateBar(new DDDBarDef());
    }

    /// <summary>
    /// shutdown the extension,it will be called by the application
    framework when the application is shutdown.
    /// </summary>
    public override void Shutdown()
    {
        try
        {
            //find the specified command bar.
            ICommandBars cmdBars = base._app.Document.CommandBars;
            ICommandBar cmdBar = cmdBars.FindBar("DDDAnalysisToolBar");

            //you must ask cmdBar if it is null.
        }
    }
}
```

```
        if (cmdBar != null)
        {
            //remove and destroy the specified command bar.
            cmdBars.RemoveBar(cmdBar);
            cmdBar.DestroyBar();
        }
    }
    finally
    {
        //you must call the base.Shutdown lastly.
        base.Shutdown();
    }
}

#region INameParserService 成员

public string ParseName(string firstName, string lastName)
{
    return firstName + lastName + " service provided by DDExtension
extension";
}

#endregion
}
```

从代码中可以看出，定制一个插件非常简单，其方便性主要体现在框架的兼容性和扩展性。需要注意的是，在开发插件工具条时，同样需要注册工具条按钮项，其注册方法前面已经介绍过，这里不再赘述。此外，同样需要在插件顶部定义GlobeID特性，用于唯一标示插件。如果需要插件支持许可可以利用[LicenseProvider特性进行配置](#)。

系统提供了插件管理器主要用于插件的加载，对于插件的加载可以根据用户的需求进行灵活的定制。如从配置文件进行插件的加载或者从指定文件夹下加载其中所有插件程序集。如果想实现自定义插件加载方法，可以继承IExtensionSearchStrategy接口，实现其中的Load方法即可。插件加载的代码如下：

```
////提取插件路径
string extPath =
ConfigurationManager.AppSettings["ExtensionPath"];
IExtensionManager manager = _app.ExtensionManager;
```

```
//从文件夹装载插件，这里可以通过实例化不同的加载算法进行插件加载
IExtensionSearchStrategy strategy = new
    FolderSearchStrategy(extPath);

//实例化插件上下文
ExtensionContext context=new ExtensionContext (strategy);

//加载插件
manager.LoadExtension(context);
```

系统建议提供基于配置文件的插件加载，配置方式如下：

配置后配置文件如下 (部分)：

```
<configSections>

    <section name="extensionsSetup"
type="Jackey.Framework.ExtensionSection, Jackey.Framework,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />

</configSections>

<extensionsSetup>
```

```
<extensions>

  <add Name="DDDDAnalsis"

GUID="ARSC.ArcEngine.DDDAnalysis.DDDEExtension"

      PluginSort="1" ProductName="三维分析工具栏" Category="高级分析扩展模块" Description="提供高级三维功能，如登高线生成。"

      LargeImage="LargeImage" SmallImage="SmallIamge" Developer="张学宝"

      Copyright="www.arsc.com" UpdateURL="www.arsc.com"

ResourceName="MyResource"

      LoadImmediately="true" SupportService="true"

TypeName="ARSC.ArcEngine.DDDAnalysis.DDDEExtension"

      AssemblyPath="\Extensions\ARSC.ArcEngine.DDDAnalysis.dll"

/>

</extensions>

</extensionsSetup>
```

为了增强Jackey.Framework的高可扩展性，其又设计为基于Service的架构模型，Service架构模式的思想是将每一个组件都设计成一个组件服务器，该服务器可以为其他组件提供额外功能。Service架构模式的实现可以方便的实现插件之间的通讯和消息传递，该模式的运用，可以极大的提高系统的可扩展性。为了能使插件支持组件服务，必须将配置文件中的SupportService设置为true才可以。如果想访问该组件服务，可以使用如下方法：

```
InameParserService
parserService=base._app.GetService(typeof(DDDEExtension)) as
InameParserService;

If(parserService != null)
{
    String fullName= parserService.Parse("zhang", "xuebao");
}
```

插件管理器主要用于管理系统中配置的各种类型插件：



### (8) 应用程序级框架扩展开发示例:

利用Jackey.Framework可以开发行如ArcMap的应用程序，也可以扩展成ArcScene的应用程序，ArcCatalog的应用程序和ArcGlobe的应用程序。Jackey.Framework也是为所有ArcGIS桌面应用程序和基于.Net框架的桌面系统提供统一的开发框架模型，在该框架下支持下进行系统功能定制。目前Jackey.Framework提供了行如ArcMap的桌面框架模型和SharpMap桌面框架，其他几个产品框架预进行后续开发。该级别的开发需要很强的软件设计能力，并且要求熟悉Jackey.Framework核心模块和ArcGIS框架模型。

下面以定制ArcMap应用程序为例，进行框架扩展开发，(注：这里只给出示意性代码)

```
/// <summary>
/// ArcMap应用程序核心对象。
/// </summary>
public class MxApplication : BaseApplication, IMxApplication,
    IWindowPosition
{
    //
    //这里定义私有成员变量
    //

    //
    //这里定义构造函数代码
    //
```



```
//
// 这里重写BaseApplication 成员
protected override void CreateDocument()
{
    Return new MxDocument();
}
//

//
// 这里重写IMxApplication成员
//

//
// 这里重写IWindowPosition成员
//

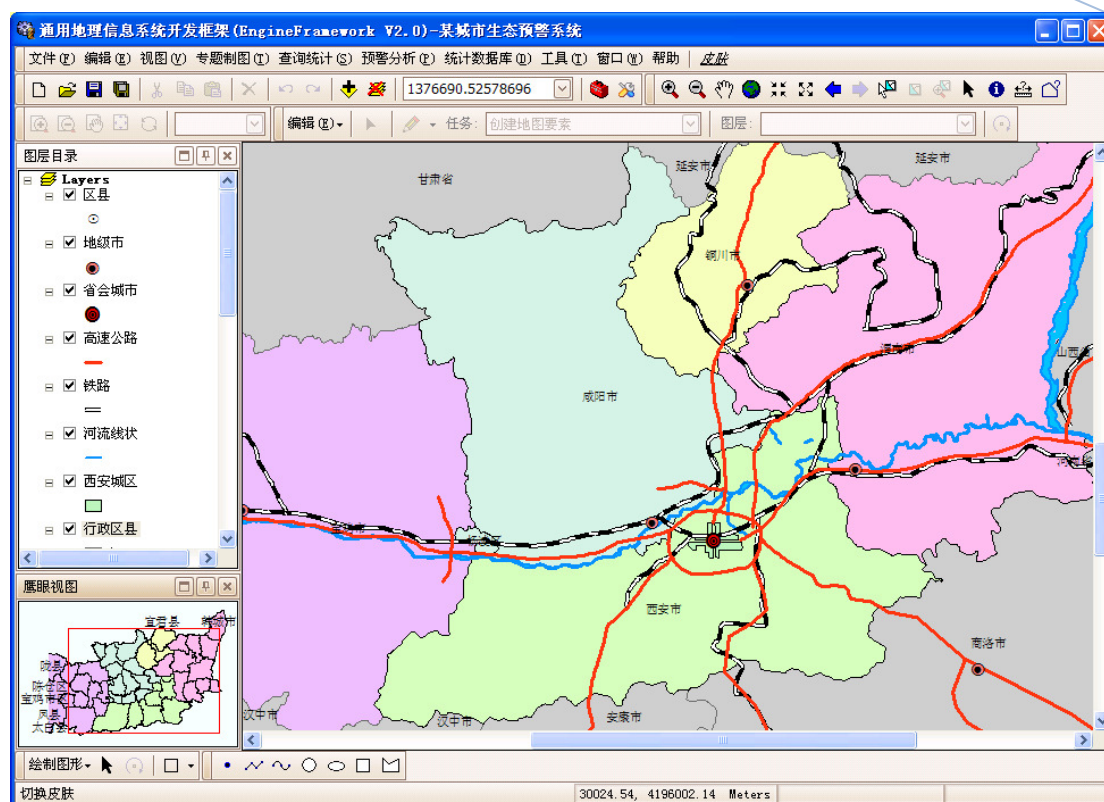
//
// 这里重写IChangeDataView成员
//
}

/// <summary>
/// 地图文档核心对象。
/// </summary>
public class MxDocument : BaseDocument, IMxDocument, IDocumentDirty,
    IContentsViewEdit, IDocumentEvents
{
    ///全部省略
}
```

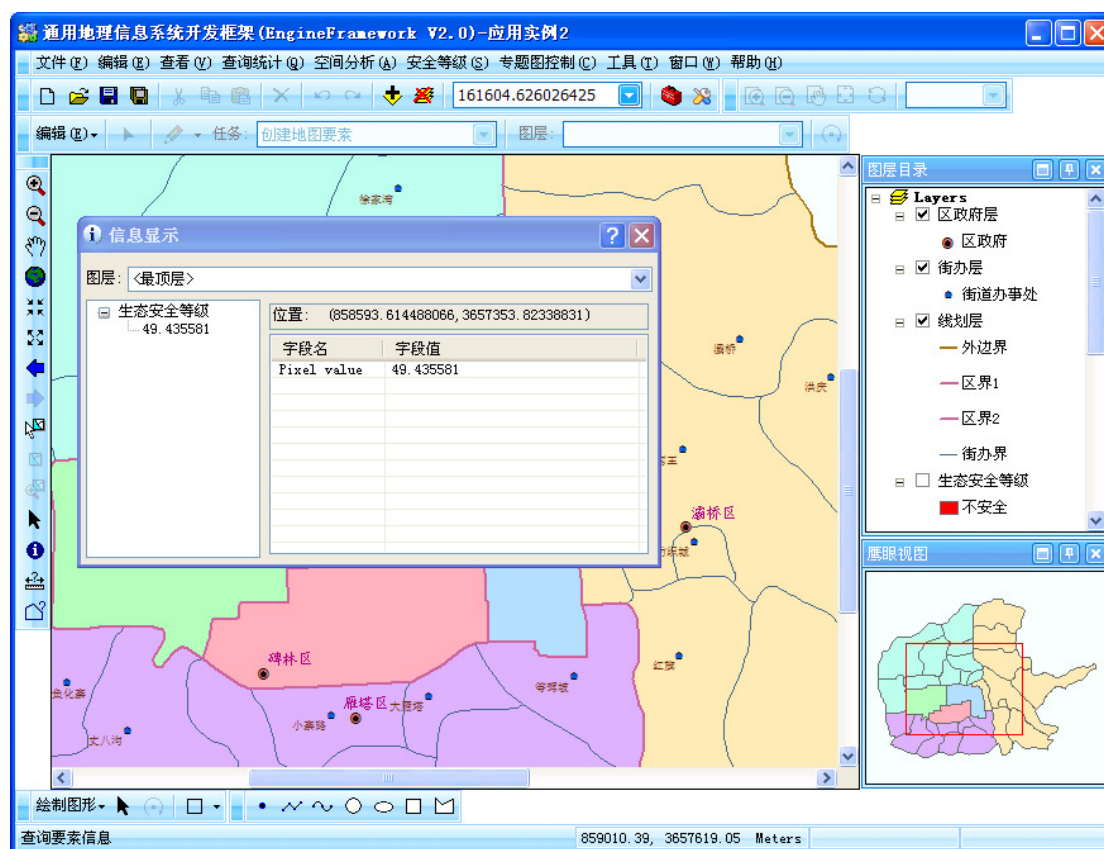
从例子中可以看出，如果想开发ArcGlobe应用程序，只需要继承BaseApplication抽象类型，重写其中的CreateDocument()方法，该方法返回GlobeDocument类型，并且使GlobeDocument类型继承BaseDocument类型即可实现扩展框架。

下面是系统开发实现的几个典型应用实例：

### 1) 利用 EngineFramework 构建的应用实例 1



## 2) 利用 EngineFramework 构建的应用实例 2



## 3) 利用 Jackey.Framework+SharpMap+GDAL\OGR 构建的简单 GIS 框架

