# 1. 项目实现的总流程

## 1.1 项目的建立

在所有程序中，点击“Microsoft Visual Studio 2010”，在子列表中点击“Microsoft Visual Studio 2010”启动Visual Studio 2010，启动后在软件的默认起始页中选择“新建项目...”。



图4-1 启动软件



图4-2 选择新建项目

弹出新建项目对话框，在最近的模板中选择Visual C#，微软.NET架构默认为.NET Framework 4，降序依据为默认值，并选择Windows 窗体应用程序，输入名称“Test0420”,项目保存路径的选择可点击“浏览(B)...”进行自定义，最后点击确定按钮，则项目创建成功。

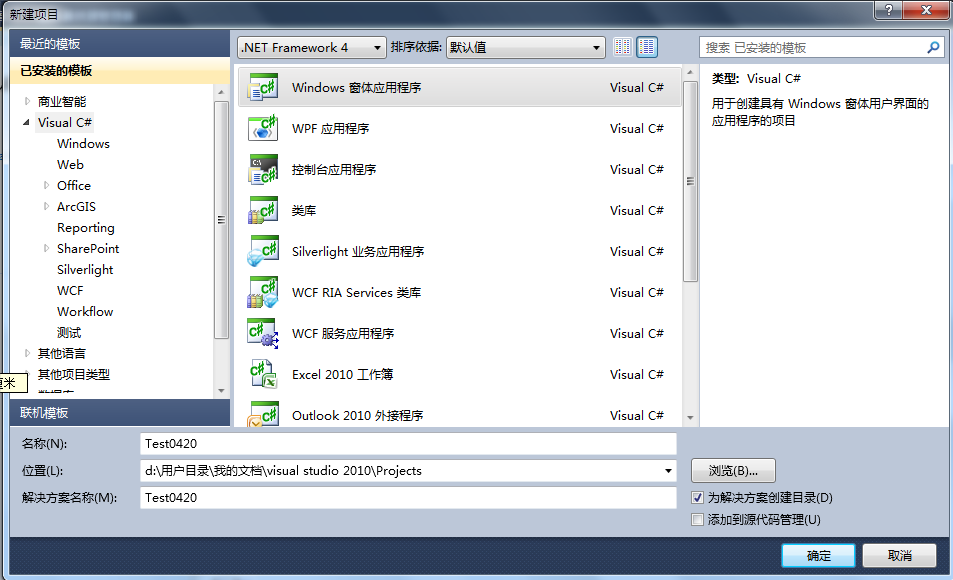


图4-3 新建项目

项目创建成功后，Visual Studio 2010默认会打开，并显示Form1.cs[设计]视图，同时也看到解决方案资源管理器和属性视图。Form1.cs[设计]视图显示的窗体就是整个项目的载体，在窗体上面可以添加各种需要的组件。

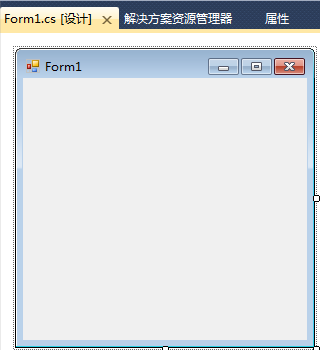


图4-4 项目窗体

鼠标双击窗体后，Form1.cs文件被打开，Form1.cs文件里面对应窗体上各个功能的代码，保证窗体上功能的实现。

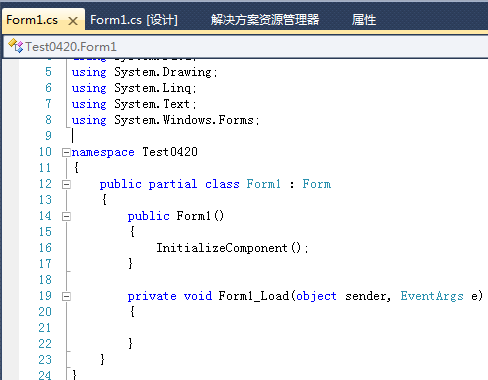


图4-5 Form1.cs文件

最后，手动拉伸一下窗体的宽和高，并在属性视图中，修改下Text属性的值，达到自定义的效果。

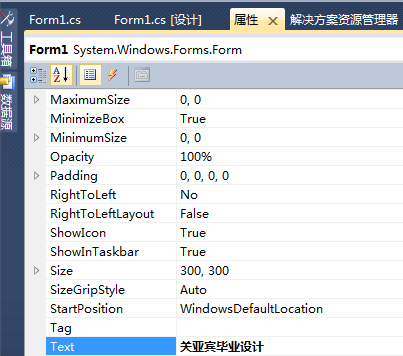


图4-6 修改窗体名称

项目的前期准备完成，窗体最后的效果如下所示。

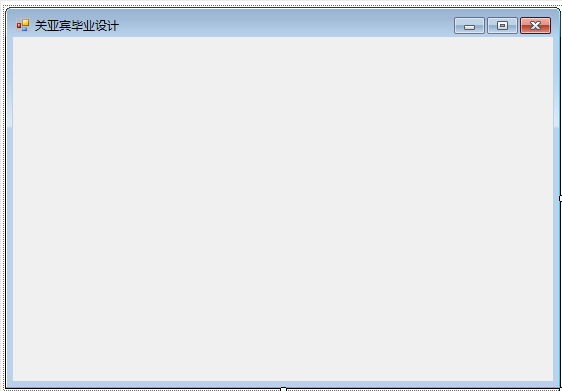


图4-7 窗体效果

## 1.2 菜单栏的添加

首先在窗体最顶部添加菜单栏，在界面左侧点击工具箱，选择“所有Windows窗体|MenuStrip”,并将其拖入窗体。菜单栏需要输入每一个选项的名称，从左到右依次输入“添加shp”、“生成WKT并存入数据库”、“连接数据库”、“查询并还原为shp”和“关闭数据库”，其中“查询并还原为shp”包含“点”、“线”和“面”三个子选项。每一个选项都具有点击事件，对应代码在Form1.cs中。

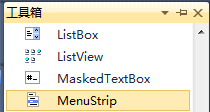


图4-8 添加菜单栏

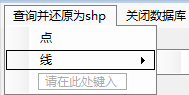


图4-9 添加选项

这是，如果想要查看显示效果，可以点击工具栏的启动调试按键，将项目运行。

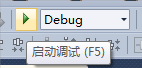


图4-10 启动调试

项目运行起来的就能够看到菜单栏的效果。

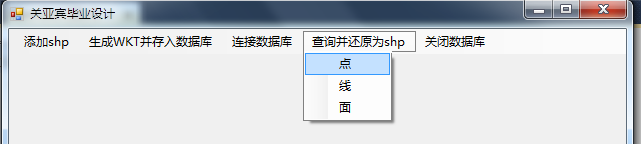


图4-11 菜单栏效果

## 1.3 相关控件的添加和关联

除了菜单栏之外，还需要添加显示shp图层的MapControl控件、显示图层信息和类型的TOCControl控件、具有放大缩小、小手等功能的ToolbarControl控件、显示数据库表信息的DataGridView控件以及退出按钮Button。



图4-12 添加控件

控件的布局存在一定的逻辑顺序，如地图图层一样，会出现重叠覆盖的问题，因此，在拖放控件时必须注意逻辑顺序[11]。



图4-13 控件添加后效果图

控件的位置除了手动调整之外，也可以选中控件，在属性视图中修改Dock属性为Top、Left、Right或者Bottom。将ToolbarControl控件的Dock属性改成Top。

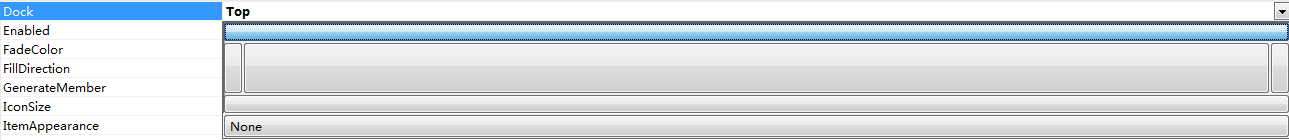


图4-14 设定Dock属性

将按钮中Text修改为退出，调整一下背景色为“Web|DeepSkyBlue”，增加美观性。

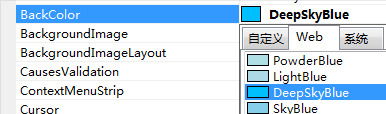


图4-15 修改按钮背景色

ToolbarControl和MapControl的关联。选中ToolbarControl鼠标右键选择属性，在属性对话框中，将Buddy改为axMapControl1，这样就实现了MapControl的绑定，当用户在操作ToolbarControl时，地图文档会在MapControl中实现相应的操作。

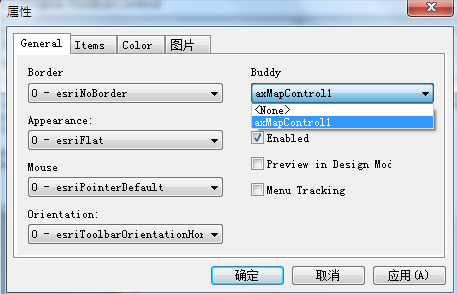


图4-16 绑定控件

选择Items，点击Add，进入Controls Commands对话框,Category下选择Generic，Commands下双击Open，添加成功，同理在Category下选择Map Inquiry，Commands下双击Identify。在Category下选择Map Navigation，Commands下双击Zoom In，Zoom Out，Pan，Full Extent，添加完成后点击Close按钮，在属性对话框点击应用并关闭。

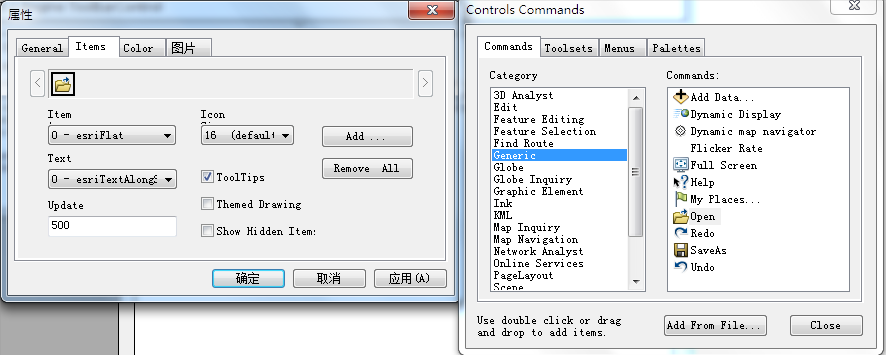


图4-17 添加工具

TOCControl和MapControl的关联操作。鼠标选中TOCControl控件右键属性，把属性对话框中的“General|Buddy”改为axMapControl1进行关联。这样关联完成，最后一步需要添加License许可。在软件的顶部菜单栏，选择“项目|Add ArcGIS License Checking...”。

在ArcGIS License Initializer对话框中，选择“Products|ArcGIS Engine”，点击OK，许可添加成功。

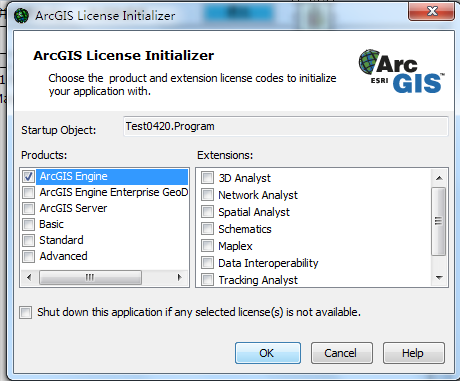


图4-18 添加许可

许可添加成功后，就可以启动调试运行项目，查看界面的情况，如下图。

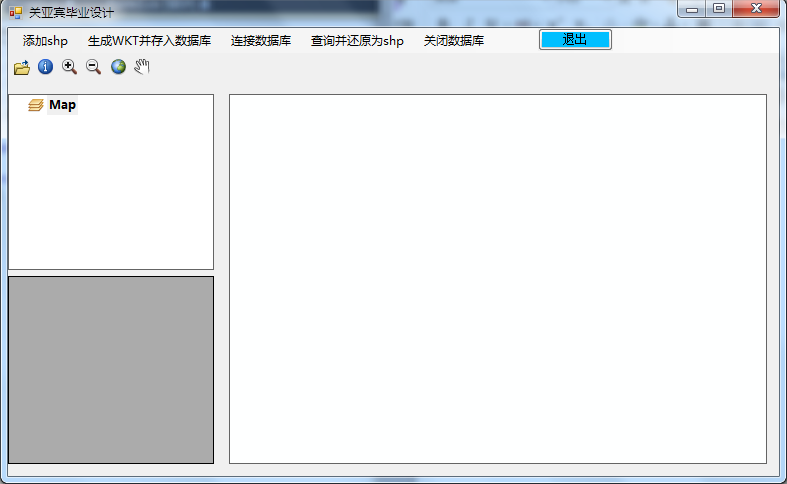


图4-19 界面展示

## 1.4 代码的编写

要实现Form1窗体上对应的功能，就需要一对一进行代码编写，代码全部写在Form1.cs中。在没有开始编写之前，在Form1.cs中已经自动生成了初始代码，最外层是命名空间Test0420，即建立项目时输入的项目名称，命名空间里面包含局部类型Form1。partial是局部类型，表示允许将一个类、结构或者接口分为几个部分，对应在各自的.cs文件中实现，但最后的编译运行仍然会将各个部分的局部类型合并为一个完整的类。

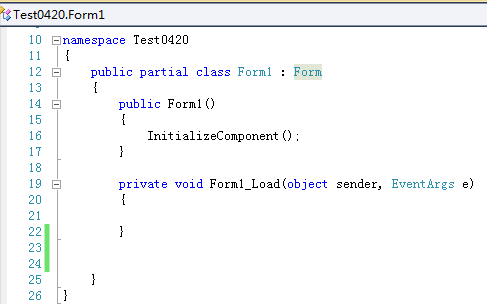


图4-20 初始代码

例如要实现Form窗体中菜单栏第一个选项“添加shp”的效果，就需要为其增加点击事件，只需要在窗体上双击此选项，视图界面自动转换到From1.cs，且鼠标定位在自动生成的函数“添加shpToolStripMenuItem\_Click(object sender,EventArgs e)”中，在这个函数中就能写入代码了。C#中的点击事件直接和控件相关联，且自动生成点击事件的函数框架。

同理，其余菜单栏或者选项都通过点击来增加点击事件。

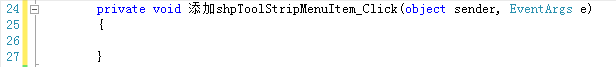


图4-21 点击事件的函数

点击事件对应函数是被包含于“public partial class Form1:Form{}”，所有的变量以及函数不能声明在其之外，否则项目运行会失败。这也就是变量或者函数的生命周期以及存在的范围，不能越界。

另外代码部分最常用的还有注解的使用，例如单行注解，使用“//”开头，多行注解使用“/\*”开头，“\*/”结尾。注解不仅利于代码含义的理解，还利于编程过程中思路的整理。

## 1.5 数据库连接的准备

在顶部菜单栏找到“视图(V)|服务器资源管理器(V)”，进入视图。由于此系统具有连接数据库和操作数据库的功能，所以配置好数据库的连接是前提。右键数据库连接，选择添加连接。

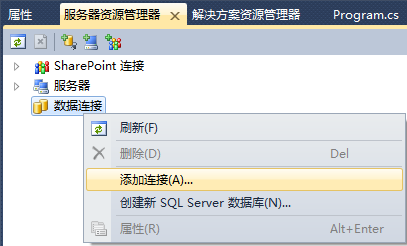


图4-22 连接数据库

进入添加连接对话框，数据源选择Microsoft SQL Server，SQL Server数据库从2008版本开始，支持geometry空间数据类型，可以存储shp图层的坐标信息。

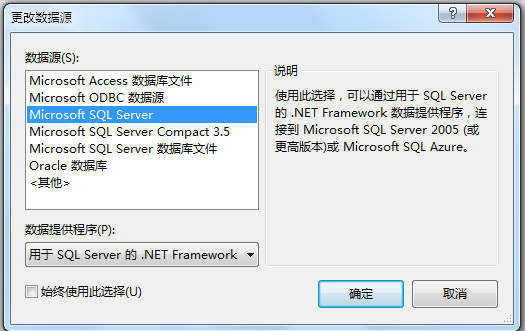


图4-23 更改数据源

下一步，配置服务器名。在安装SQL Server 2012数据库过程中，默认使用计算机名称作为服务器名，即用默认的Windows 身份验证方式进行登录，而没有使用自定义的用户名搭配密码的方式。

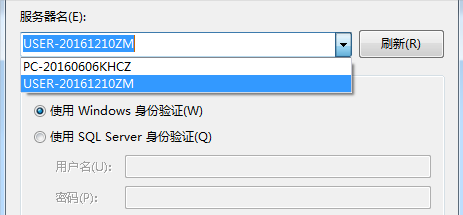


图4-24 配置服务器名

这时，需要打开数据库，打开SQL Server 2012，可以看到启动数据库之前也需要连接到服务器，用默认的Windows 身份验证，点击连接。



图4-25 启动数据库

连接成功后进行新建数据库，这是为之后创建数据库表以及向表中存储数据做准备。在对象资源管理器视图中打开名称为计算机名的服务器，右键数据库选择新建数据库，在新建数据库弹出框，手动输入自定义数据库名称，例如“school”，不需要增加其它的设置，点击确定，则名称为school的数据库建立完成。

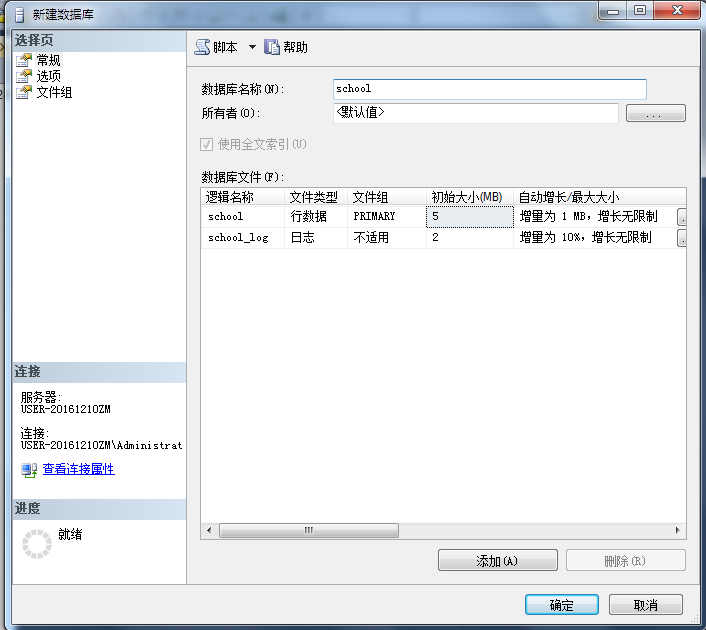


图4-26 新建数据库

打开school数据库，其下有8个子列表，其中表代表数据库表，也是此项目操作的对象，在第五章会详细介绍如何使用代码操作数据库，实现创建数据库表、向表中循环插入数据和读取表中所有的数据这些功能。

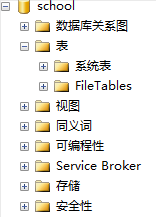


图4-27 数据库结构

数据库建立完成，在添加连接对话框中选择建立好的数据库school，并点击测试连接，查看连接到数据库的状态，若出现弹框显示“测试连接成功”，表明配置正确，没有问题。再次点击确定，完成配置，进入服务器资源管理器视图能看到数据连接子列表中出现的数据库。

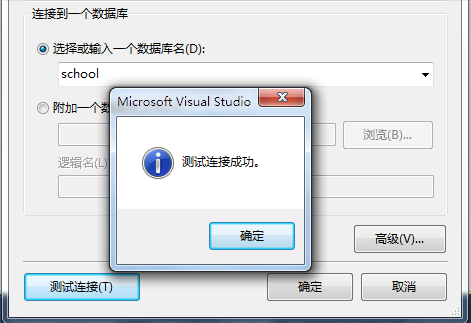


图4-28 连接测试

在项目最终运行时，数据库的操作全部通过SQL语句进行，那么如何得知连接数据库成功与否？这里我特意在窗体上增加了Label控件，显示连接的状态信息，来给出提示，表明数据库是否连接成功。

到这里，Form窗体上所用到的所有控件都已经介绍完毕，为了更加清楚的看到各个控件的作用和关系，特此增加结构图展示。

Form窗体

Label

Button

MenuStrip菜单栏

TOCControl

ToolbarControl

DataGridView

关联

关联

MapControl

图4-29 窗体控件结构图

# 2. 全部的功能

## 2.1 加载shp图层并显示

本课题使用的数据就是shp图层数据，我特意挑选了河南省内的地市作为点图层，高速公路作为线图层，各个直辖市区作为面图层。一般我们查看shp图层可以使用ArcMap直接打开查看。那么如何实现在MapControl控件中显示shp图层？就是这一步需要解决的问题。

在菜单栏第一个选项“添加shp”点击事件的函数中，需要实现的效果有两个：一是打开文件选择框选择shp图层，选择后点击确定；二是把用户选择的图层关联MapControl控件并显示。代码实现的步骤流程从前到后主要是创建一个工作工厂new ShapefileWorkspaceFactory()；文件过滤，指定文件类型ShapeFile；选中文件后获取文件名；创建要素图层new FeatureLayerClass()；关联图层以及要素类pLayer as ILayer；最后一步添加到地图控件中。整个过程的流程示意图如下：

添加到地图控件中

创建工作空间工厂

关联图层和要素类

打开shp文件对应的工作空间

打开要素类

创建要素图层

图5-1 添加shp流程图

关键代码如下：

//创建工作空间工厂

IWorkspaceFactory pWorkspaceFactory=new ShapefileWorkspaceFactor

y();

//文件属性过滤

openFileDialog1.Filter="ShapeFile文件(\*.shp)|\*.shp";

//指定打开文件的文件夹路径

openFileDialog1.InitialDirectory=@"E:\毕业论文资料\河南省交通图";

//得到文件名称

String pFileName=System.IO.Path.GetFileName(pPath);

//创建工作空间和要素类

IWorkspace pWorkspace=pWorksapceFactory.OpenFormFile(pFolder,0);

IFeatureWorkspace pFeatureWorksapce=pWorksapce as IFeatureWorkspa

ce;

IFeatureClass pFc=pFeatureWorksapce.OpenFeatureVlass(pFileName);

//创建要素图层

IFeatureLayer pFlayer=new FeatureLayerClass();

pFLayer.FeatureClass=pFc;

//关联图层和要素类

ILayer pLayer=pFlayer as ILayer;

IMap pMap=axMapControl1.Map;

//添加图层到地图控件中。

pMap.AddLayer(pLayer);

//刷新图层

axMapControl1.ActiveView.Refresh();

保证代码的正确使用，需要引用四个类库，分别是ESRI.ArcGIS下的DataSourcesFile、Geodatabase、Carto、Geometry。

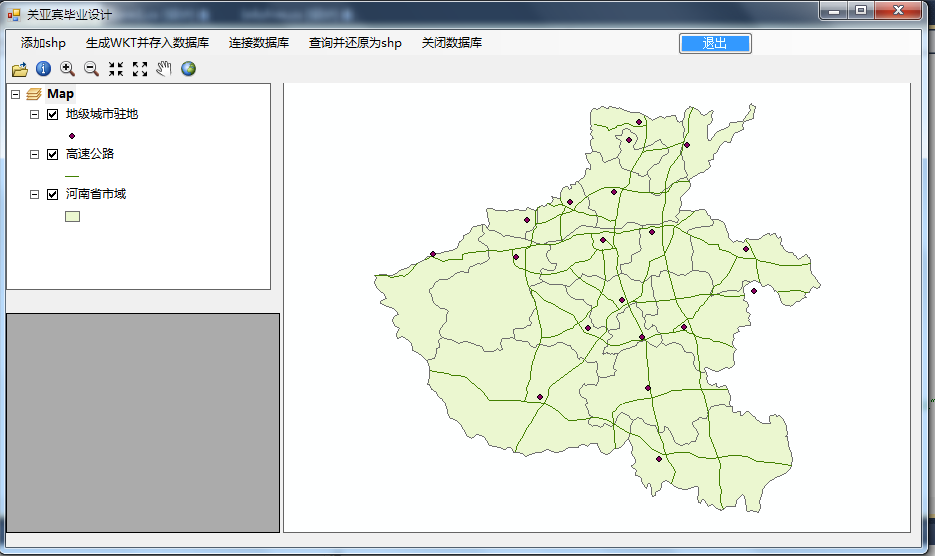


图5-2 添加shp效果

从效果图上可以看到，添加的点、线和面图层在MapControl控件上正常显示了。左侧的TOCControl控件则显示图层名称和类型，这就是控件关联的效果。

## 2.2 图层的浏览、放大缩小和平移

操作TOCControl控件上的放大、缩小以及小手功能，如下图就是放大功能，在图层上鼠标可以画框进行放大。同理，缩小也是通过鼠标画框进行缩小。点击小手，在MapControl控件上就能移动图层了。点击地球图标，能将所有的图层以最大显示范围展现在MapControl控件中，类似于全图显示的功能。

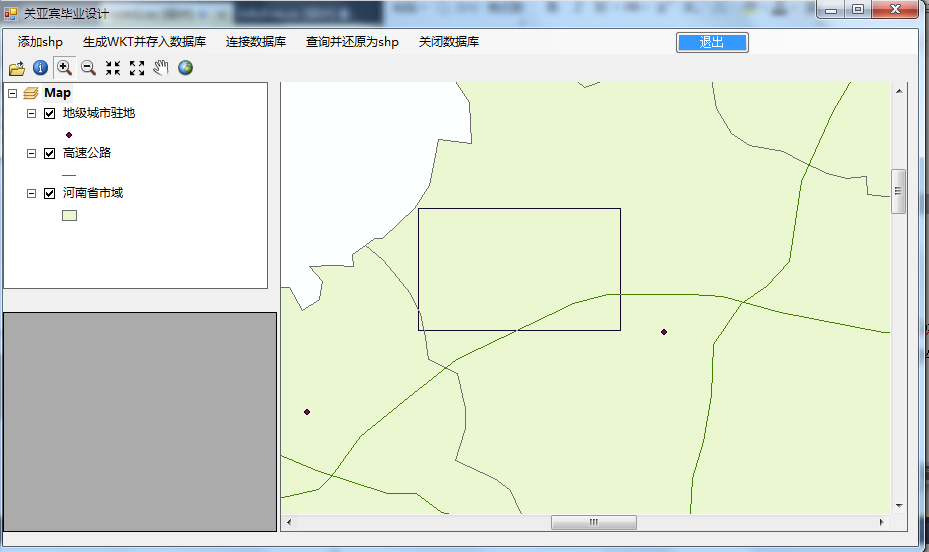


图5-3 放大镜功能

## 2.3 连接和断开SQL Server 2012数据库

C#中连接数据库需要使用到的类是SqlConnection，并实例化一个对象，使用其Open方法，就可实现数据库的连接。但是用SqlConnection的前提是在Form1.cs增加引用的类库。和SQL语句相关的类库有Microsoft.SqlServer、System.Data.SqlClient、System.Data.SqlTypes和Microsoft.SqlServer.Typ

es，都需要通过using添加进来，才可以使用其中的类。重要代码如下。

//产生数据库连接对象的实例，参数中school之前我自定义的数据库的名字

SqlConnetion myconnection=new SqlConnection("Data Source=(local);

Initial Catalog=school;Integrated Security=True");

//调用open方法打开数据库，如果文本控件显示提示信息表明操作成功

myconnrction.Open();

label1.Text="数据库连接成功";

在数据库连接成功之后，才能正确的进行数据库操作，如本章第五节对应的创建数据库表并插入数据，第七节的读取数据库表中的所有数据，都是在数据库连接成功的前提下进行操作的。而断开数据库只需要调用SqlConnection的Close()方法即可。

## 2.4 读取shp图层的坐标信息并转换为WKT

shp文件中存储的是图层的坐标信息，例如一个点，那么对应点的shp文件中存储的就是点坐标X值和Y值。一条线由多个点组成，则线的shp文件中存储的是点的集合，集合的第一个点就是线的起始点坐标，集合的最后一个点是线的终点坐标。对于面，举个最简单的例子，正方形由四个点组成，在第四个点没有连接到第一个点时还是线，当第四个点和第一个点连接后组成了闭合的多边形，所以多边形中的坐标信息也是点坐标的集合。

读取shp文件中的WKT信息，即读取坐标信息，获取坐标信息。我使用了开源的GDAL类库，步骤简单，获取的WKT信息以字符串的形式输出。

首先，借助GDAL需要添加引用，进入解决方案资源管理器，右键引用，选择添加引用，需要引入的引用有gdal\_csharp.dll、gdalconst\_csharp.dll、ogr\_csharp.dll和osr\_csharp.dll，引入成功后如下图。

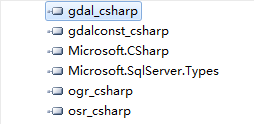


图5-4 GDAL的引用

由于最后输出的WKT是字符串形式，可以事先定义好字符串变量方便之后使用。

//定义WKT信息对应的字符串

private string wktString;

使用private表示私有属性，表示在此Form.cs中独有，不能被继承。

获取到shp图层的WKT信息最关键的就是如何使用GDAL类库。重要代码如下：

//初始化类库

OSGeo.OGR.Ogr.RegisterAll();

//调用GetDriverByName获得Driver对象，参数指定是shapefile

OSGeo.OGR.Driver dr=OSGeo.OGR.Ogr.GetDriverByName("ESRIshapefil

e");

//path表示shp文件的路径

OSGeo.OGR.DataSource ds=dr.Open(path,0);

//得到图层对象

OSGeo.OGR.Layer layer=ds.GetLayerByIndex(0);

//调用图层GetSpatialRef方法得到投影信息

OSGeo.OSR.SpatialReference coord=layer.GetSpatialRef();

String coordString;

//字符串变量接收坐标信息

coord.ExportToWkt(out coordString);

//开始循环读取坐标信息

String wkt;

String strwkt=string.Empty;

OSGeo.OGR.Feature feat

While((feat =layer.GetNextfeature())!=null){

OSGeo.OGR.Geometry geometry=feat.GetGeometryRef();

geometry.ExportToWkt(out wkt);

//例如点坐标，wkt输出的格式为POINT (40.646160125732422 64.520668029785156)

//加上换行符，一个点数据、一条线数据或者一个面数据读取完成后换行，并且使用字符串追加，得到所有的坐标信息

strwkt+=wkt+"\n";

}

//返回值就是当前图层的所有WKT信息

return strwkt;

在循环结束后就得到了shp图层的wkt信息。

## 2.5 将WKT信息存入数据库对应的表中

根据功能需求，对应点、线和面的shp图层，在读取完成后得到了WKT信息，就要存入数据库中，但存入数据库的前提有两个。一是判断当前图层是点、线还是面，二是根据判断的结果创建一个新的数据库表，创建完成数据库表之后，才能循环插入数据。

当前图层的数据属于点、线或是面的判断取决于数据的类型。

在获取wkt信息过程中得到的OSGeo.OGR.Layer其对象layer在这里就发挥了关键作用，调用layer的GetGeomType()就可以得到空间数据类型。

String typeName=layer.GetGeomType().ToString();

typeName代表的含义是空间数据类型，那么它有三种可能的值，分别是wkbPoint、wkbLineString和wkbPolygon。所以根据情况进行判断，将代码封装在if(){}中，就能分开操作，对症下药。

例如当前图层是点图层，那么对应代码如下：

if(typeName.Equals("wkbPoint")){

if(num1==1){

//创建数据库表，但表只需要一张，所以创建表的SQL语句只能执行一次

string createSQL= "create table point(id int IDENTITY(1,1),lo

cation geometry);";

//运行SQL语句需要SqlCommand类,两个参数，一个是string类型的SQL语句，一个是SQLConnection对象

SqlCommand sqlCommand=new SqlCommand(createSQL,myconnectio

n);

//执行SQL语句

sqlCommand.ExecteNonQuery();

//执行成功，弹框显示提示信息

MessageBox.show("点，对应数据库表已经创建！");

//创建表后num1自增加1变为2，也就不符合if的条件了，保证了创建表只能执行一次

num1++;

}

//point表创建完成，但表中没有数据，下面就是循环插入数据

//循环的次数就是shp图层中点的数目，通过layer.GetFeatureCount方法得到坐标点的总数

if(num<=layer.GetFeatureCount(0)){

//定义插入数据的SQL语句。使用geometry::STGeomFromText()函数来处理WKT信息，并添加了SRID空间引用标识

insertSQL="insert into point(location) values(geometry::STG

eomFromText('"+wkt+"',0));";

//执行SQL语句，这里插入数据依旧处于循环之中

SqlCommand insertCommand=new SqlCommand(insertSQL,myconnec

tion);

//调用ExecuteNonQuery方法执行SQL语句

insertCommand.ExecuteNonQuery();

num++;

}

//循环插入point数据库表数据完成

}//整个while循环结束

读取某一个图层的所有点坐标数据都在循环中进行，到这里需要解决两个问题：一是读取到的数据后的存放位置，而是存储数据的方式。

这两个问题同样在整个循环结束后也已经解决，循环时根据空间数据类型判断当前图层是点、线或者面，判断结束后立即创建数据库表，当然，创建数据库表只需要执行一次，也只能执行一次。数据库表创建完成后就能顺利进行循环插入数据的操作了，插入数据同样放在循环里面，将一个点、一条线或者一个面的点数据一次又一次的插入数据库中，整个大循环结束了，数据库表也创建好了，并且表中已经存入了这个图层所有的数据。

这里只展示了点图层的处理方式，对于线和面同样适用，只不过需要修改一下数据库表的名字即可，我这里创建的3张数据库表分别是point、polyline和polygon。为了更清晰的了解读取WKT信息并存入数据库的全过程，特此画出程序流程图如下：

开始

是wkbPolygon

建polygon表，循环插入数据

WKT数据存储完成

是wkbLineString

建polyline表，循环插入数据

判断typeName

GDAL读取图层坐标数据

建point表，循环插入数据

是wkbPoint

结束

图5-5 存储WKT过程

选择高速公路.shp图层作为演示的例子，在加载高速公路.shp之后点击菜单栏的“生成WKT并存入数据库”选项，触发后台的生成wkt数据ToolStripMenuItem\_Click()函数，进而开始读取图层的WKT信息，并根据typeName的值创建数据库，创建完成后出现弹框，提示创建成功的信息。

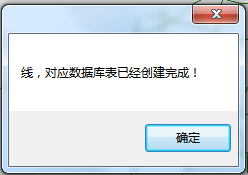


图5-6 建表成功

点击确定之后，后台会循环读取每一条线上的点集合，将每一条线的点集合作为一个完整的数据插入数据库，插入完成也就等于循环完成，最后出现InfoFrm窗体显示全部的WKT信息。

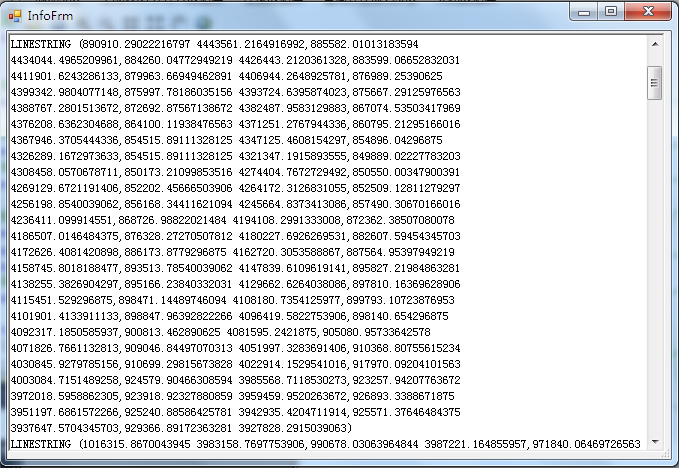


图5-7 WKT信息

关闭InfoFrm窗体之后，进入SQL Server查看是否生成了polyline数据库表。打开数据库，展开使用的school数据库，在表坐在文件夹右键进行刷新操作，即等同于更新操作，就能看到新生成的数据库表polyline。

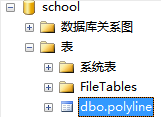


图5-8 查看数据库表

为了更详细的查看数据库的字段以及值，选中dbo.polyline鼠标右键选择编辑其 200行，打开数据库表。数据库表中除了主键id之外还有一个location字段，location字段的数据类型是geometry，这是SQL Server数据库独有的。Location值的格式是数据类型名称之后括号括起来的是一对又一对的坐标点，从线的第一点起点直到最后的终点。

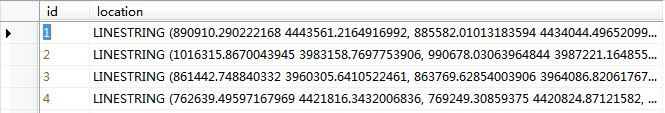


图5-9 表的字段值

## 2.6 将WKT信息在新窗体中显示

为了更加直观的查看到WKT数据信息，这里我把WKT信息作为参数，显示在一个新的窗体中。

创建这个新的窗体需要进入解决方案资源管理器中，选中项目名称，右键“添加(D)|新建项(W)...”，添加新项对话框中点击“Visual C#项|Windows窗体”，名称输入InfoFrm.cs，点击添加。之后InfoFrm.cs[设计]视图打开，可以看到InfoFrm窗体。

InfoFrm窗体出现的时机是在读取WKT信息结束后显示，即此时SQL Server 2012数据库中已经创建表完成并循环插入数据结束。InfoFrm窗体出现的目的是为了显示WKT信息，所以要增加RichTextBox，显示文本信息，其具有滚动条的功能。当文本信息过大，但显示区域固定时，滚动条出现，用户能够滑动鼠标或者移动滚动条查看全部的数据，这一点类似于CSS3中的overflow:auto，当一个固定宽高的DIV中显示的文本大大超过了其本身所占面积，那么增加此样式后就会自动出现滚动条。

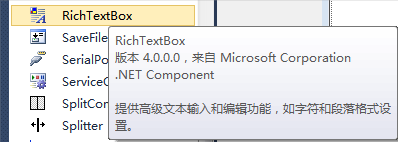


图5-10 添加RichTextBox

将RichTextBox控件拖拽到InfoFrm窗体，并调整下各自宽度高度一个合适的长度，最终效果图如下。

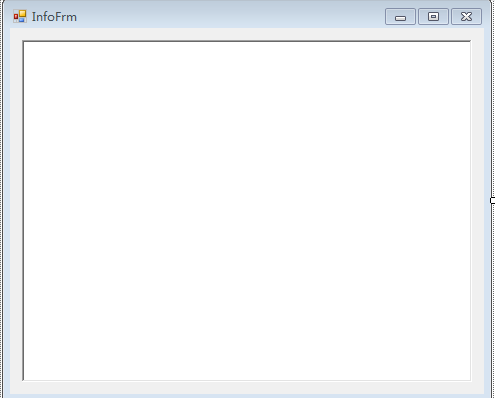


图5-11 InfoFrm效果图

此时InfoFrm还是单独的存在，没有和系统其他部分发生关联，而我要实现的效果是在WKT存储完成后，InfoFrm窗体弹出后，WKT文本在RichTextBox控件中。实现这些效果的关键代码如下：

//调用读取WKT信息的函数，参数是shp文件的完整路径,函数返回值是字符串，也就是WKT信息

String wktString=ReadShp(pPath);

//调用InfoFrm.cs中的public InfoFrm(string str)方法实例化InfoFrm对象，传入参数WKT信息

InfoFrm info=new InfoFrm(wktString);

//窗体对象显示

info.ShowDialog();

注意一点，这里使用了InfoFrm.cs中的公共方法，等同于java中常说的构造方法，而且是有参数的构造方法，构造方法是无条件执行的，且只会执行一次。构造方法的完整代码为：

Public InfoFrm(string str){

InitializeComponent();

//传递过来的字符串参数，赋值给rtxtInfo的文本。

rtxtInfo.Text=str;

}

有参数的构造方法的优点就是接收参数并为自己所用。rtxtInfo.Text=str是赋值操作，类似于java中的int a=1。而rtxtInfo就是RichTextBox控件的Name属性值，这就实现了InfoFrm窗体中显示文本信息的效果了。

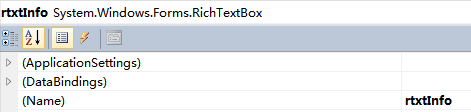


图5-12 RichTextBox的属性

如下图，InfoFrm窗体中显示的文本框具有滚动条效果，浏览数据变得更加方便。

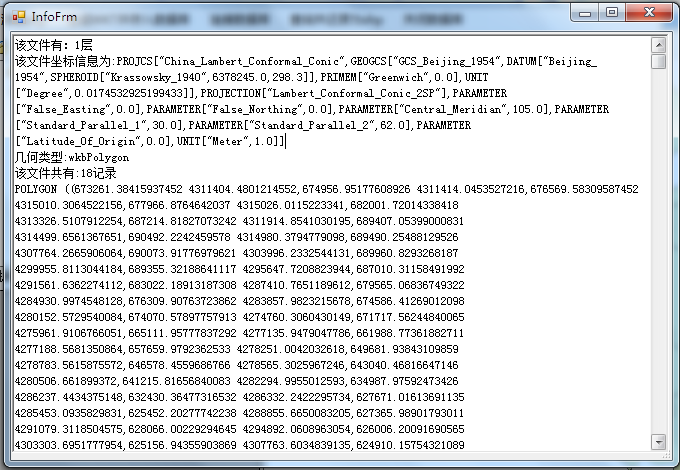


图5-13 InfoFrm窗体

## 2.7 从数据库表中读取WKT信息

从数据库表中读取WKT信息，转换为字符串，这一步是为还原为shp图层做的准备，也是整个系统的分水岭。从最开始的加载shp图层并显示，以及读取坐标信息并存入数据库只是实现了数据存储的功能，是从上往下的过程，从文件到数据库表的转移过程，转移的对象是shp文件中的坐标信息。反过来，使用逆向思维，能不能再一次从数据库表中读取这些数据呢？读取后将坐标信息存入新的图层后输入，实现文件的还原呢？这是从下往上的过程，从数据到文件。来回颠倒数据的意义不仅仅是要你学会C#中关于地理空间数据的类、方法和接口，更是理论和实践的结合，有了数据处理的思维模式，推而广之，处理其它类型的数据也就不在话下了。

数据库表中的数据在一个项目或者系统中可以看做是最底层的，类似于根和树的关系，无非就是数据的增删改查操作。一般情况下，需要查询某些数据会直接在数据库中进行查询，本系统使用了SQL Server 2010，那么可以先在数据库中进行查询。打开数据库，在界面顶部菜单栏点击“新建查询(N)”，SQLQuery1.sql视图被打开。

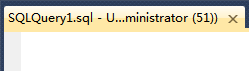


图5-14 SQL Server查询操作

在SQLQuery1.sql界面，输入任意SQL语句，再点击顶部菜单栏的“执行(X)”，SQL语句就被运行了。

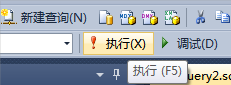


图5-15 执行操作

以point表为例子，进行查询操作，首先打开point表，point表中有主键id和geometry类型的location字段，location字段存放的是一个个点的坐标值。

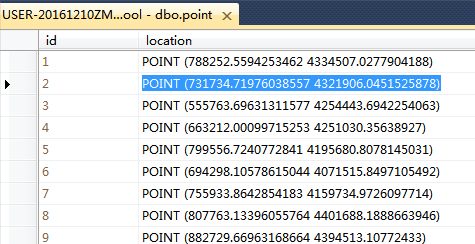


图5-16 point表数据

首先使用最常见的查询方法进行查询，输入SQL语句“select \* from point;”，点击“执行”，查询虽然成功了，但出现了很奇怪的现象，location的值发生了变化，变为类似于二进制的形式。原因也很简单，表中的location字段是特殊的geometry空间数据类型，不适合普通的查询。那么如何实现查询后的location的值依旧是“POINT（x,y）”，这里引入了进行geometry空间数据类型查询特用的方法，借助系统函数STAsText()或者ToString()将WKT信息转换为字符串类型，便可正常显示。

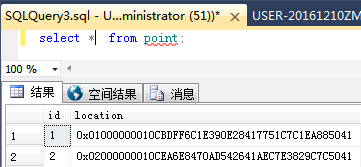


图5-17 普通查询

使用系统函数进行查询，执行语句“select location.STAsText() from point;”,查询结果正常显示，或者使用“select location.ToString() from point;”，结果是一致的。查看查询结果，会发现查询的这一列没有列名，原因是SQL语句中没有将查询结果给定一个名字。如何给定？只需要改变SQL语句为“select location.AtAsText() as location from point;”，as后面就是这一列指定的名字，我这里给定的名字和原来的列名一致。

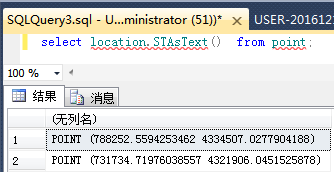


图5-18 查询操作

这是point表的查询方法，同样，polyline表和polygon表也适用系统函数STAsText()或ToString()进行查询，得到最初的WKT文本信息。

但point表的数据隐含着很特殊的一点，每一条数据是一个坐标点，每一个点都是唯一存在的，且点与点之间相互独立，没有关系。在接下来的实现功能中需要分别得到点的X和Y坐标数据，不需要整个点的类似于POINT(x,y)这样的数据。那么系统函数中是否存在这样的函数，可以将点数据中的坐标值剥离出来呢？答案是有的，执行SQL语句“select location.STX as x,location.STY as y from point;”，可以成功得到点的X坐标和Y坐标。单独得到X和Y坐标将在还原为点图层过程中发挥关键作用，使用到的系统函数是STX和STY。

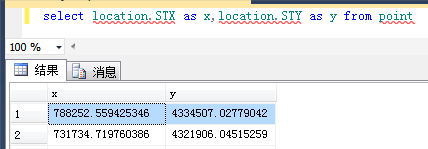


图5-19 查询XY坐标

明确了SQL语句，下面就要在From1.cs中编写代码了。在MenuStrip菜单栏的第四个选项“查询并还原为shp”有三个子列表，即点、线和面。对应要实现的功能分别是在数据库中对point、polyline或polygon进行查询操作，并将查询得到的数据在DataGridView控件中展示，同时对查询出的数据进行格式上的修改，最终得到的是文本的格式，并为接下来的还原为shp图层做准备，所以这一步的操作将会得到格式上有些不同的WKT信息。



图5-20 菜单子列表

子列表中点对应的点击函数为private void 点ToolStripMenuItem\_Click

(object sender,EventArgs e){}，函数中有两个部分，一部分是普通查询并关联DataGridView控件，第二部分是使用系统函数查询，得到每一条点数据的X坐标和Y坐标，并创建IPoint对象。重要代码如下。

//定义SQL语句，字符串类型

String selectSQL="select \* from point;";

//实例化SqlDataAdapter对象，联合DataSet来读取数据

SqlDataAdpter oDataAdpter=new SqlDataAdapter();

//与sql命令对象绑定

oDataAdpter.SelectCommand=new SqlCommand(selelctSQL,myconnectio);

DataSet ds=new DataSet();

//填充数据

oDataAdpter.Fill(ds,"城市表");

//填充完成后，绑定控件显示信息

dataGridView1.DataSource=ds.Tables[0];

到这里第一部分完成，效果如下图。对于polyline表和polygon表的查询方法并关联实现完全和point表一致。

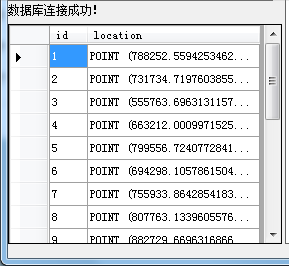


图5-21 dataGridView效果

下面为WKT还原为shp的过程。在本节一开始我就详细介绍了在数据库中通过执行SQL语句进行查询，查询point表对应的SQL语句是“select location.STX as x,location.STY as y from point;”，每执行一次查询便得到一条数据的X坐标和Y坐标，并将坐标赋值给Double类型变量，为转换为IPoint对象做准备。实现这个过程的代码是：

//进行SQL操作的类

SqlCommand cmd=myconntion.CreateCommand();

cmd.CommandText= "select location.STX as x,location.STY as y from point;";

//执行查询，并将查询到的值以流的形式返回给SqlDataReader对象

SqlDataReader readerPoint=cmd.ExecuteReader();

//在循环之前，定义一个整型变量记录循环的次数，也就是点的个数

Int a=0;

//开始循环读取，读取的结果是两列，分别是X坐标和Y坐标，其索引下标从0开始

While(readerPoinr.Read()){

Double douX=readerPoint.GetDouble(0);

Double douY=readerPoint.GetDouble(1);  
 if(addPoint(ly,douX,douY)){

a++;

}

}

//循环结束，关闭输出流

readerPoint.Close();

在循环中使用了自定义的addPoint(ly,douX,douY){}函数，三个参数分别是ILayer对象，双精度类型的X和Y坐标，这个函数的作用将在5.9节中详细讲解。

以上就是从point数据库表中读取数据的过程，最终读取的结果是每一个点的X坐标和Y坐标。由于点数据的特殊性，并且可以直接使用系统函数得到坐标值，所以整个读取功能难度不大。但是对于线和面，情况就不同了。

一条线段除了起点和终点之外还有多个中间点，而面就更为复杂，例如正方形是四个点组成的封闭图形，在众多的点中判断哪些点属于同一个面则是重点，且每次循环操作只针对一个面上的点，这都是需要解决的问题。

先来解决线的问题，打开数据库polyline表，查看线数据的格式LINESTRING (x y, x y, x y, x y)，有一对括号，括号前有个空格，括号里面使用逗号作为每一对坐标的分割符，逗号后面接着又有一个空格，所以如何像点一样得到一条线上的多个点坐标，这就是难点。我的解决方法是字符串的截取和剥离方法，在每次循环读取一条线时，修改字符串，最后得到点的坐标。

和点的操作一样，使用SqlDataReader读取数据，对应的SQL语句是“select location.STAsText() as location from polyline”,操作字符串的代码部分如下。

//由于会多次操作字符串，首先在循环之前需要定义6个字符串对象，其实这是比较笨的方法，但能达到想要的效果

String str1="";

//其余5个字符串不在写了，再定义一个整型变量记录读取次数，也就是多少条线

int num=0;

while(readerPolyline.Read()){

//得到第一条线数据，转换为字符串，值是STRINGLINE (x y, x y)

String lineText=readerPolyline.GetString(0);

//首先去除字符串中的所有的英文字母,Regex类的使用需要添加引用，using System.Text.RegularExpressions;

str1=Regex.Replace(lineText,"[a-zA-Z]","");

//str1变为 (x y, x y)，括号前有空格，下面去除首尾的空格

str2=str1.Trim();

//将括号代替为空字符

str3=str2.Replace("(","");

str4=str3.Replace(")","");

//str4为x y, x y，下面修改逗号和逗号后面的空格，统一进行换行，达到一个点坐标在一行，方便之后的按行读取坐标点

str5=str4.Replace(", ","\n");

//str5为为一行都是x y的格式

str6=str5.Replace(" ",",");

//str6将空格换位逗号，最终的格式时每一行都是一个点的坐标，且X和Y坐标之间使用逗号隔开，方便下面的按行读取，参数就是str6

Using(StringReader sr=new StringReader(str6)){

//字符串line在循环中会接受每一行的值，整型a作为记录数

String line="";

int a=0;

//调用ReadLine()方法返回值是每一行的字符串对象，判断不为空是循环是否继续执行的条件，为空就停止循环

while((line=sr.ReadLine()!=null)){

//定义字符数组存放X和Y坐标,数组长度是2

String[] xy=line.Split(',');

//获取索引下标为0和1的X坐标值和Y坐标值，并转换为double类型，使用了Convert.ToDouble(str)方法，参数是字符串，返回值是double类型

String x=xy[0];

String y=xy[1];

double douX=Convert.ToDouble(x);

Double douY=Convert.ToDouble(y);

//这样就得到了线中点的X和Y坐标值

}

}

}

//循环结束，关闭输出流

readerPolyline.Close();

这就是获取线数据中点坐标的方法，在真正得到坐标数据之前是花了大量功夫处理字符串，直到最后数据变成每一对坐标都独占一行的效果。面数据的处理和线一样，但增加了一个name属性，作为点的标识符。

在面数据的读取和还原过程中，出现了当前多边形的最后一个点和下一个多边形的第一个点连在一起的情况，造成这种情况的原因就是没有对点进行整合，没有将当前多边形的最后一个点和下一个多边形的第一个点进行分离的操作，故为每一个点增加了name属性，作为唯一标识。

str5=str4.Replace(", ",",多边形"+num+"\n");

在让每一对坐标换行之前，在Y后增加一个逗号，逗号后面紧跟name属性值，标识当前多边形是第几个多边形。例如第一次循环，得到第一个多边形中所有的点坐标，转换字符串格式效果如下图。

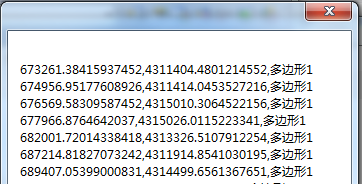


图5-22 多边形处理WKT信息

在循环读取过程中，和线不同的一点是增加了List集合，泛型为string。

List<string> pColumns=new List<string>();

//进入循环读取，将所有的数据存入集合中，方便在生成集合是增加name 字段

Using(StringReader sr=new StringReader(str6)){

String line;

//单行读取，有数据就会进入循环

While((line=sr.ReadLine())!=null){

String[] xyAndName=line.Split(',');

for(int j=0;j<xyAndName.length;j++){

//读取一个点，集合存入3个对象，X和Y坐标，以及点对应的name属性

pColumns.Add(xyAndName[j]);

}

String x=xyAndName[0];

Double douX=Convert.ToDouble(x);

String y=xyAndName[1];

Double douY=Convert.ToDouble(y);

String name=xyAndName[2];

//得到了坐标值和当前坐标所属的多边形名称

}

}

到这里，从数据库表中读取WKT信息已经完成，下面两节将介绍还原为shp图层的详细过程，即生成点、线和面图层，并向图层中循环增加数据，循环结束后MapControl控件关联生成的图层，使其显示。

## 2.8 创建点、线和面图层

创建点、线和面图层对应的函数分别是CreatePointShapeFile()、CreateLineShapeFile()和CreateShpFromPoint(string shapeFileFullName)。点和线使用的方法是一样的，只创建图层，不添加数据，添加数据放在循环中添加，而面图层对应函数把创建图层和插入数据结合在一起了。

先展示点图层的创建过程。

//返回值是ILayer对象

Private ILayer CreatePointShapeFile(){

//首先打开文本框，用户选择文件保存的位置

SaveFileDialog sFileDialog=new SaveFileDialog();

//指定文件保存类型是能为shp

sFileDialog.Filter="Shape文件(\*.shp)|\*.shp";

DialogResult dResult=sFileDialog.ShowDialog();

//定义文件保存完整路径，以及文件名称，为之后再次加载数据做准备

String fileFullPath;

String fileName;

String filePath;

//文件保存对话框，用户选择“确定”之后，获取文件名和路径

if(dResult==DialogResult.OK){

fileFullPath=sFileDialog.FileName;

int index=fileFullPath.LastIndexOf("\\");

fileName=fileFullPath.Substring(index+1);

filePath=fileFullPath.Substring(0,index);

}

//实例化一个字段集合对象

IFields fields=new FieldsClass();

IFieldsEdit fieldsEdit=fields as IFieldsEdit;

//实例化单个字段集合对象

IField field=new FieldClass();

IFieldEdit fieldEdit=field as IFieldEdit;

//为单个字段对象赋值，并添加入字段集合中

fieldEdit.Name\_2="Shape";

fieldEdit.Type\_2=esriFieldType.esriFieldTypeGeometry;

//IGeometry是用来设计几何字段的

IGeometryDef geometryDef=new GeometryDef();

IGeometryDefEdit geometryDefEdit=geometryDef as IGeometryDefE

dit;

//指定图层类型是点

geometryDefEdit.GeometryType\_2=esriGeometryType.esriGeometryPoint;

geometryDefEdit.SpatialReference\_2=axMapControl1.SpatialReference;

fieldEdit.GeometryDef\_2=pGeometryDef;

//集合字段添加完成

fieldsEdit.AddFiels(field);

//管理矢量数据

IFeatureWorkspace pFeatureWorkspace=pWorksapceFactory.OpenFrom

File(filePath,0) as IFeatureWorksapce;

//最终，创建图层

pFeatureWorkspace.CreateFeatureClass(fileName+".shp",fields,null,null,esriFeatureType.esriFTSimple,"shape","");

//控件关联图层并显示

axMapControl1.AddShapeFile(filePath,fileName);

//返回索引下标是0的图层，但没有数据

return this.axMapControl1.get\_Layer(0);

}

这就是创建一个图层的全部过程，最开始弹出的另存为对话框，用户选择文件的保存位置和文件名称，点击确定后获取到保存路径和文件名称，开始正式的创建图层。首先定义字段，字段赋值后加入字段集合中，并将指定类型后的地理几何对象赋值给字段集合，最后创建图层类。

对于线和面图层，只是在指定esriGeometryType上不同，对应是esriGeometryPolyline和esriGeometryPolygon。

## 2.9 从WKT还原为shp并展示

还原为shp图层重要的两部分别是：创建图层和向图层中循环添加数据。

创建图层已经在上一节介绍到了，这一节主要介绍如何正确的循环插入图层中数据。

//点图层在循环读取数据前进行创建

ILayer ly=CreatePointShapeFile();

While(readerPoint.Read()){

//循环箱图层整增加点，图层只有一个，点坐标有多个

if（addPoint(ly,douX,douY)）{

a++;

}

}

这里就引出了循环增加数据的addPoint函数，其重要代码如下：

Private bool addPoint(ILayer ly,double x,double y){

//转换类型

IFeatureLayer layer=ly as IFeatureLayer;

//由下往上转换

IFeatureClass iFc=layer.FeatureClass;

IFeatureClassWrite fr=iFc as IFeatureClassWrite;

IWorkspaceEdit workSpace=(iFc as IDataset).Workspace as IWorkspaceEdit;

workSpace.StartEditing(true);

workSpace.StratEditOperation();

//定义点集合和图层类

IFeature feature;

IPointCollection pPointColl=new PolygonClass();

IPoint pt=new PointClass();

pt.X=x;

pt.Y=y;

//向点集合中增加点对象

pPointColl.AddPoint(pt);

//转换为shape图层

feature.Shape=pt;

fr.WriteFeature(feature);

return true;

}

效果图如下。首先弹出“另存为”对话框，选择保存的位置，我这里选择保存在F盘，文件名称为0420Point，点击保存。

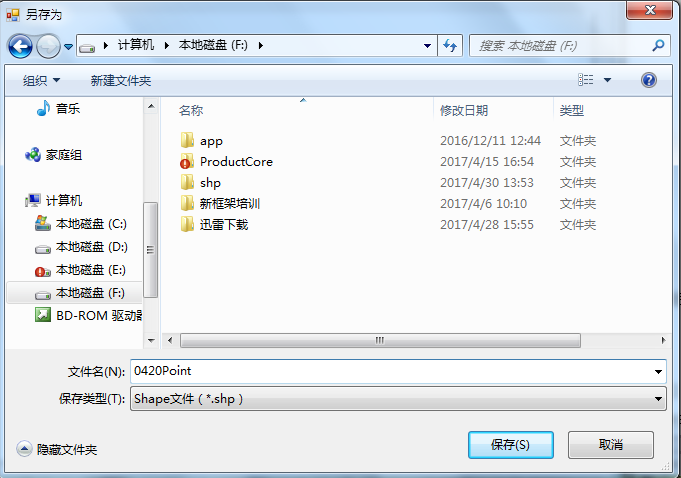


图5-23 保存shp文件

保存成功后，进入循环，循环结束会有弹框提示图层是否创建成功。

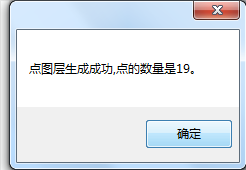


图5-24 点图层生成

图层生成后，获取其文件路径和文件名，就会自动加载到MapControl控件中并显示。

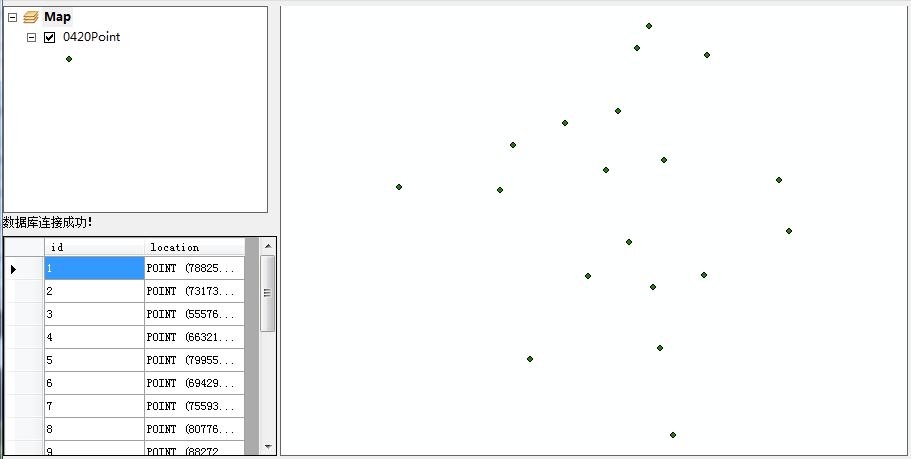


图5-25 点图层显示

创建线图层和点大体一样，在循环之前创建图层对象和空间数据集合对象，间数据集合对象是为存放线数据上多个点的集合特意设计的。

ILayer ly=CreateLineShapeFile();

IGeometryCollection geometryCollection=new PolylineClass();

//在循环中将一条线上所有点放入点集合，这里的点集合使用接口实现类PathClass实现，线段是由路径，即path组成

//在真是进入单行的读取前，需要创建线对象和object

ILine pLine=new LineClass();

//线对象是每次读取一条数据是创建一次，在大循环中，单行读取循环之外

Object missing=Type.Missing;

IPointCollection pointColl=new PathClass();

while((line=sr.readLine（）！=null)){

IPoint pPoinr=new PointClass();

pPoint.X=douX;

pPoint.Y=douY;

//循环加入点集合

pointColl.AddPoint(pPoint,ref missing,ref missing);

}

//一条线上所有的点都加入点集合了，将点集合转换为空间数据集合

geometryCollection.AddGeometry(pointCollection as IGeometry,ref missing,ref missing );

//将空间数据集合转换为线对象,调用调价线函数，在线图层中加入线

IPolyline pLine=geometryCollectio as IPolyline;

addLine(ly,pLine);

这里引出了addLine(ILayer ly,IPolyline pline)函数，它和点对应的的addPoint(ILayer,double x,double y)十分类似。

feature.Shape=pLine;

//只有这一处的不同，将线对象转换为了图层

线图层的效果如下图。

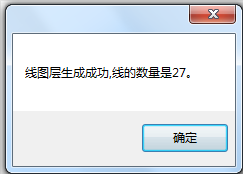


图5-26 线图层生成

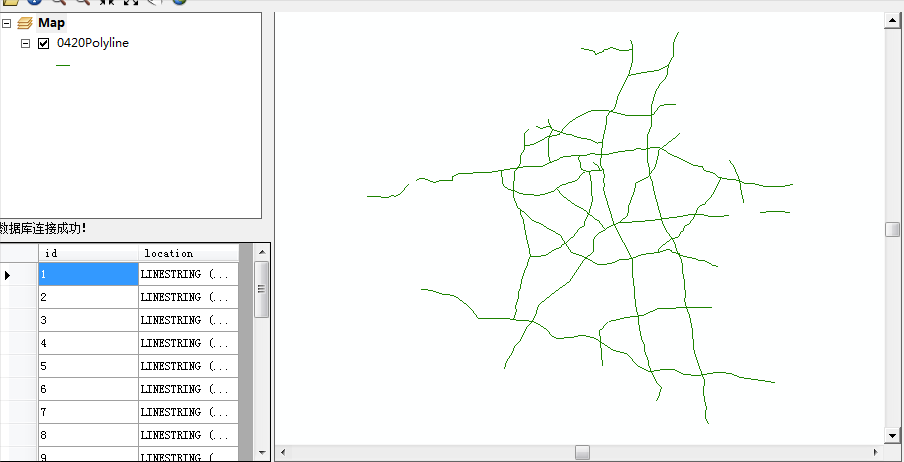


图5-27 线图层显示

最后到了面的还原，在一开始我就遇到了在一个多边形上的所有点没有变成闭合图层的情况，当前多边形的最后一个点和下一个多边形的第一个点有连线。问题的原因是在循环时没有判断当前点属于哪一个多边形。为解决这个问题，特别增加了一个自定义类CPoint和泛型是CPoint的列表List。

Struct CPoint{

public double px;

public double py;

public string pname;

}

List<CPoint> cpointList=new List<CPoint>();

CPoint类除了具有点的X和Y坐标值属性之外，还具有name属性，根据每一个点的name属性判断其属于哪一个多边形，在循环时直接根据name属性循环，就能把同属于一个多边形的所有点全部加进来。

在循环读取所有的多边形中点的过程中，产生CPoint对象，加入列表集合，循环完成，集合存储完成，进入生产面图层这至关重要的一步，在这个过程中，会根据点的name属性值进行循环，name属性值一样的点会依次进入循环，并放置在纯点集合IPointCollection中。

//获取所有的多边形名称的集合

List<string> pNameList=new List<string>();

for(int i=0;i<cpointList.Count;i++){

//循环的次数是坐标的数量，并将不同的pname属性放入pNameList

if(pNameList.Contains(cpointList[i].pname.Trim())==false){

pNameList.Add(cpCointList[i].pname.Trim());

//pNameList中只存在多边形的名称，且不重复

}

}

for(int i=0;i<pNameList.Count;i++){

IPointCollection pointColl=new PolygonClass();

Object o=Type.Missing;

//循环点集合，循环的结构是数量多的放在循环里面，数量小的包含数量大的

for(int j=0;j<cpointColl.Count;j++){

if(cpointList[i].pname.Trim()==pNameList[i].Trim){

//pname属性相同的进入次循环得到一个点集合

IPoint pPoint=new PointClass();

pointColl.AddPoint(pPoint,ref o,ref o);

}

}

//点集合最后再追加一个点，即起止点在集合最后又出现了一次，保证了闭合图形的形成

if(pointColl.PointCount>0){

IClone pClone=pointColl.get\_Point(0) as IClone;

IPoint endPoint=pClone.Clone() as IPoint;

pointColl.AddPoint(endPoint,ref o,ref o);

}

//一个多边形循环结束，创建图层

IFeature feature = pFeatureClass.CreateFeature();

feature.Shape=pointColl as IPolygon;

}

面图层的还原过程中，主要使用了一个自定义的CPoint类，并通过name属性是否一致进行循环，特别巧妙的一点就是在集合最后追加起点，保证了闭合图形的形成。

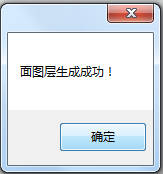


图5-28 面图层生成

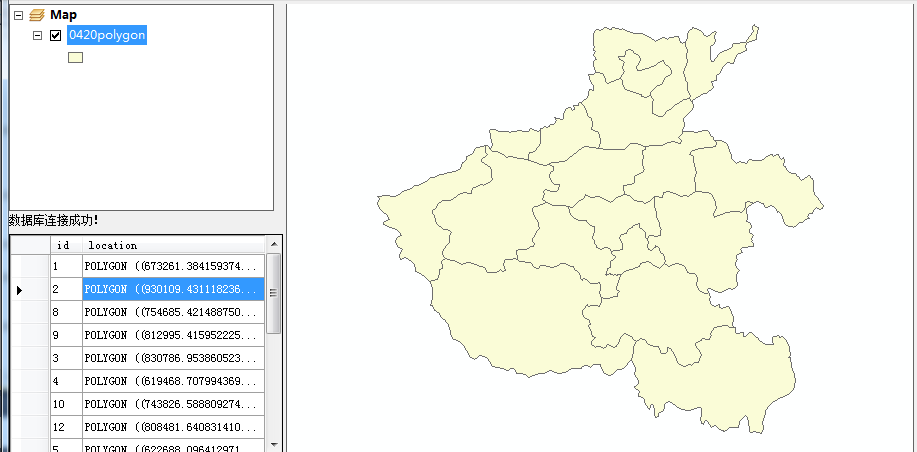


图5-29 面图层显示

## 2.10 退出系统

退出按钮的效果等同于点击窗体上的叉号，代码则使用this.Dispose();实现，代表关闭整个窗体，退出程序。