

用 MapX 与 C#开发地理信息系统

第四章 MAPX 与 C#实例.....	5
4.1 MAPX 图层建立.....	5
4.1.1 MapX 数据与地图的组织结构.....	5
4.1.2 实例 1: 建立/添加一个用户自定义图层.....	5
4.1.3 在 MapX 中使用栅格图层.....	7
4.1.4 实例 2: 栅格图层的建立.....	8
4.2 图元自动标注.....	9
4.2.1 实例 3: 给图层加上自动标注功能.....	9
4.3 MAPX 地图集.....	9
4.3.1 什么是 MapX 地图集 (Geoset)?.....	9
4.3.2 实例 4: 打开已存在的地图集文件.....	10
4.3.3 实例 5: 保存地图集.....	10
4.4 内置工具的使用.....	11
4.4.1 使用标准工具.....	11
4.4.3 实例 6: 内置标准工具的使用.....	11
4.5 自定义工具.....	12
4.5.1 创建自定义工具.....	12
4.5.2 实例 7: 创建测量长度和面积自定义工具.....	13
4.6 MAPX 地图符号样式的定制.....	14
4.7 在图层上添加自定义图元.....	15
4.7.1 实例 8: 鼠标点击向图层上添加图元.....	15
4.7.2 实例 9: 给定坐标向图层上自动添加图元.....	17
4.8 获得图元属性.....	20
4.8.1 实例 10: 获取选定图元的属性.....	20
4.9 图元的选取.....	21
4.9.1 实例 11: 实现 InfoTip 功能.....	21
4.10 图元属性的修改.....	22
4.10.1 实例 12: 修改图元属性.....	22
4.11 实例 13: 图元的查询.....	23
4.12 实例 14: 鹰眼图的实现.....	24
4.13 数据绑定.....	26
4.14 GPS 在 GIS 系统中的应用.....	27
4.14.1 定位信息的接收.....	27
4.14.2 定位信息的提取.....	27
4.14.3 定位信息在 MapX 中的显示.....	28
4.14.4 实例 15: GPS 定位系统的应用.....	28
4.15 多媒体信息在 GIS 系统中的应用.....	33
4.15.1 GIS 中嵌入多媒体的方法.....	33
4.15.2 实例 16: 在 MapX 系统中嵌入多媒体数据.....	33
第五章 MAPX 与 ORACLE 结合.....	35
5.1 ORACLE 数据库对 GIS 的支持.....	35
5.1.1 面向对象的数据库支持.....	35
5.1.2 Oracle spatial 组件的引入.....	35
5.2 循序渐进学习 ORACLE SPATIAL 在 MAPX 中的应用.....	36
5.2.1 oracle 服务器的安装.....	36
5.2.2 准备由 Oracle Spatial 存储的图层文件.....	36
5.2.3 Easyloader 上载工具.....	36
5.2.4 图层信息在 Oracle 中的存储结构.....	38
5.2.5 用程序实现 MapX 图元到 oracle 数据库的上载.....	42
5.2.6 用程序实现 oracle 数据表数据下载至 MapX 中显示.....	44

5.2.7 图元样式的还原.....	46
5.3 在网络环境下实现图层信息共享.....	47
第六章 MAPCTRL 控件的开发方法.....	57
6.1 主要功能.....	57
6.2 开发步骤.....	57
6.3 程序实现.....	58
第七章 分发基于.NET平台的MAPX应用程序.....	91
7.1 .NET FRAMEWORK 概述.....	91
7.2 .NET FRAMEWORK 的主要组件和功能.....	92
7.2.1 公共语言运行库.....	92
7.2.2 .NET Framework 类库.....	92
7.3 安装 .NET FRAMEWORK.....	93
7.4 MAP 客户安装.....	93
7.5 制作安装程序.....	93

第四章 MapX 与 C#实例

这一章我们通过若干专题来介绍用 C#如何开发 MapX 应用程序。

4.1 MapX 图层建立

4.1.1 MapX 数据与地图的组织结构

MapX地图是由一个一个图层合成而来。MapX 将其所有基础信息以 MapInfo表的形式组织起来；每一表都是一组 MapInfo 文件，用来在地图中建立一个图层。

这一组 MapInfo 文件包括：

.<Somefile>.tab：图层属性结构定义文件，该文件描述 MapInfo 表的结构。它是描述包含数据文件格式的小文本文件。

.<Somefile>.dat (.mdb、.aid 或 .dbf)：图层属性记录文件，这些文件包含表格数据。可用记事本或相应的数据库管理软件打开浏览数据。

.<Somefile>.map：图层空间记录文件，该文件描述空间图形对象（如果该表没有任何地图对象，则该文件将不存在）。

.<Somefile>.id：图层索引文件，该文件是将数据与空间对象相链接的交叉引用文件（如果该表没有任何地图对象，则该文件将不存在）。

.<Somefile>.ind：它是索引文件。通过该索引文件，您可以使用 Find 对象搜索地图对象。

要创建图层就要了解这些内部机理，方能思路清晰。下面是一个创建自定义层的例子。

4.1.2 实例 1：建立/添加一个用户自定义图层

4.1.2.1 程序功能

在地图上建立一个用户自定义的图层，该图层上的每个图元包括图元编号、图元名称、图元描述、图元坐标等属性，并且生成一个数据集与该图层绑定。

4.1.2.2 程序实现

```
public bool NewUserLayer(string layerName)
//新建自定义图层，若存在则添加到图层集中
{
    MapXLib.Layer layer;
    MapXLib.Fields flds=new MapXLib.FieldsClass();
    flds.AddStringField("source", 50, false);
    flds.AddStringField("name", 50, false);
    flds.AddStringField("identity", 50, false);
    flds.AddStringField("description", 50, false);
    flds.AddStringField("foundTime", 50, false);
    flds.AddFloatField("objX", false);
    flds.AddFloatField("objY", false);
```

```

        MapXLib.LayerInfo layerInfo;
        layerInfo=new MapXLib.LayerInfoClass();
        layerInfo.AddParameter("FileSpec", @appDirectory+"\\ "+layerName+".tab");
        layerInfo.AddParameter("Name", layerName);
        layerInfo.AddParameter("Fields", flds);
        layerInfo.AddParameter("AutoCreateDataset", 1);
        layerInfo.AddParameter("DatasetName", "ds"+layerName);
        if (!File.Exists(@appDirectory+"\\ "+layerName+".tab"))
        {
        layerInfo.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeNewTable;
        }
        else
        {
        layerInfo.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeTab;
        }
        try
        {
            layer = axMap1.Layers.Add(layerInfo, 1);
            axMap1.Refresh();
            return true;
        }
        catch
        {
            return false;
        }
    }
}

```

4.1.2.3 程序说明

(1)flds 是 MapXLib.Fields 对象，即图层的属性字段集，将来会出现在.tab 文件中。用 new MapXLib.FieldsClass() 来实例化一个新的 Fields 对象。在对任何对象进行引用前，必须先实例化该对象。

AddStringField, AddFloatField 是 Fields 字段集对象的两个方法，分别用来定义字符串字段及浮点型字段，并添加到 Fields 字段集对象中。有关 AddStringField(), AddFloatField() 的语法请参考 MapX 文档。

(2)MapXLib.LayerInfo 对象是用来增加新层的一个非常好的方法，在 layerInfo 中具体定义该层的一些参数。

- layerInfo.AddParameter("FileSpec", @appDirectory+"\\userDrawLayer.tab"): 指定该层的存放路径。@为转义字符， appDirectory 为应用程序目录变量，可用 Directory.GetCurrentDirectory() 来得到。 userDrawLayer.tab 为该层的.tab 文件名。
- layerInfo.AddParameter("Name", "userDrawLayer"): 指定该层的名字" userDrawLayer"，名字将会出现在图层控制对话框中。
- layerInfo.AddParameter("Fields", flds): 指定该图层的属性字段集对象，即上面新定义的 MapXLib.Fields 对象。
- layerInfo.AddParameter("AutoCreateDataset", 1): 指定是否自动产生数据集，1 自动产生，0 不

产生。有关数据集的概念，在后续部分再重点作介绍。

- `layerInfo.AddParameter("DatasetName", "dsUserLayer")`：指定数据集的名字“dsUserLayer”。
- `layerInfo.Type` 属性：指定新层的类型。

`MapXLib.LayerInfoTypeConstants.miLayerInfoTypeNewTable`：指定产生一新层。

`MapXLib.LayerInfoTypeConstants.miLayerInfoTypeTab`：指定一存在的图层。

- `axMap2.Layers.Add(layerInfo, 1)` 语句：增加 `layerInfo` 所定义的新层，其中“1”代表增加至图层最上面。

这样就得到一个名为 `userdrawlayer` 的图层，并且得到一个名为 `dsUserLayer` 的数据集与该图层自动绑定。利用这种方法建立数据集非常方便，也非常好用。在一般情况下都可满足系统的需求。该数据集可理解为一张表格，该表格的结构即为上面定义的 `flds` 字段集对象，该表格中的每一个记录对应图层上每一个图元的属性记录。

编号	名称	描述	经度	纬度
01	北京市	中华人民共和国国首都
...

(3)`LabelProperties` 用来说明如何用数据集中的数据标注图层上每一个图元。

`layer.LabelProperties.Dataset`：指定数据集对象。

`layer.LabelProperties.DataField`：指定用数据集中哪个字段值标注图元。

`layer.LabelProperties.Position`：指定标注位置。

`layer.LabelProperties.Style.TextFont.Size`：指定标注的字体及大小等。

`layer.LabelProperties.Offset`：指定标注离图元中心的距离。

下面我们就可以在刚建立好的图层上描绘自己的内容了。

4.1.3 在 MapX 中使用栅格图层

4.1.3.1 什么是栅格图象？

栅格图象是由多行微小的点（象素）组成的一种计算机化的图象。如果手头有扫描仪及扫描软件，可以通过扫描一幅纸张地图来创建栅格图象。完成地图扫描并将其保存于文件中后，即可在 `MapInfo` 中显示该文件。

有多种不同的栅格图象文件格式。`MapInfo` 能够处理以下格式的栅格图象文件：`JPEG`、`GIF`、`TIFF`、`PCX`、`BMP`、`TGA`（`Targa`）和 `BIL`（`SPOT` 卫星图片）。

4.1.3.2 什么是配准栅格图象？

配准一幅栅格图象时，要输入地图坐标（如经度 / 纬度），并指定栅格图象上与该坐标对应的点。因为栅格图象文件不包含地理坐标信息，所以要在 `MapInfo Professional` 中显示栅格图象前必须进行配准，以使 `MapInfo Professional` 在显示图象时能够完成地理计算，如计算距离和面积等。

在 `MapInfo Professional` 中首次打开一幅栅格图象时，`MapInfo Professional` 显示“配准栅格图象”对话框。填写该对话框以告知 `MapInfo Professional` 如何配准图象。`MapInfo Professional` 将栅格图象配准信息保存在表文件中以供以后使用。下一次打开该栅格图象表时就不必再进行配准了。这样，只须对栅格图象进行一次配准。

栅格图层的应用在地理信息系统中也有重要作用。一般作为背景使用，特别是可作为鹰眼图的背景使用，可防止在改变视图时引起的刷新。

4.1.3.3 配准栅格图像

若还没有在 MapInfo Professional 中显示过某个栅格图像，执行下述步骤配准该图像：

(1) 选择“文件”>“打开”，“打开”对话框出现。

(2) 从“文件类型”下拉列表中选择“栅格图像”。MapInfo Professional 显示栅格图像文件清单。

(3) 选择要打开的栅格图像文件并选择“打开”。弹出一个 MapInfo Professional 对话框，点击“配准”按钮。MapInfo Professional 显示“图像配准”对话框。该栅格图像的一个预览出现在对话框的下半段。

(4) 通过选择“投影”按钮并完成“选择投影”对话框来设定该图像的地图投影。配准栅格文件时，初始的投影方式就是表的缺省投影。

如果通过扫描纸张地图创建栅格图像，该纸张地图应包含所用的地图投影信息。如果不能确定地图投影，使用缺省地图投影（经 / 纬度）。缺省地图投影方式是由“地图窗口参数设置”中的“缺省投影”设置的。如果不清楚的话，可以使用经纬度。

(5) 把鼠标光标移到对话框下半段的预览图像上，并移到一个已知地图坐标（例如经 / 纬度）的点，再单击鼠标按钮。MapInfo Professional 显示“增加控制点”对话框。

(6) 通过输入对应于在地图图像上单击位置的地图坐标，完成“增加控制点”对话框。

记住，本初子午线以西的任何位置有负的经度，赤道以南的任何位置有负的纬度。因此，西经 73 度对应于 X 值-73。

如果以度为单位输入坐标，必须输入小数度而不是度 / 分 / 秒。

(7) 重复步骤 5 和 6，直到输入最少三个控制点。要保证精确结果，输入五或六个控制点。所增加的每个控制点有助于 MapInfo Professional 把地球坐标同栅格图像上的位置联系起来。理想地，在图像的每个角至少有一个控制点。

所需的控制点数依赖于栅格图像的性质。如果不能确定地图投影或正在使用没有实际地图投影的图像，例如航空照片，也许要输入二十或更多控制点。

(8) 完成增加控制点后选择“确定”。MapInfo Professional 把该栅格图像显示在地图窗口中。

完成“图像配准”对话框后，MapInfo Professional 把配准信息保存到一个表文件（.tab）中。这样我们就可以象利用普通 Mapinfo 表文件一样来用这个栅格图层了。

4.1.4 实例 2：栅格图层的建立

4.1.4.1 程序功能

装载一栅格图层到图层集合的最低层。

4.1.4.2 程序实现

```
public bool LoadRasterLayer(string layerName)
{
    int layerNumber=axMap1.Layers.Count;
    try
    {
        axMap1.Layers.Add(layerName+".tab", layerNumber+1);
        return true;
    }
    catch
    {
        return false;
    }
}
```



```
}
```

4.1.4.3 程序说明

装载栅格图层与装载普通图层一样，用同样的语法结构 `Layers.Add()`。

4.2 图元自动标注

对一个图元加上标注可直观的给用户识别地图上的每一个地理对象，在 **MapX** 中可设置自动标注图元和手工标注图元。

4.2.1 实例 3：给图层加上自动标注功能

4.2.1.1 程序实现

```
private void autoLabel(string layerName)
{
    MapXLib.Layer layer=axMap1.Layers._Item(layerName);
    MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName);
    layer.LabelProperties.Dataset =ds;
    layer.LabelProperties.DataField =ds.Fields._Item("name");
    layer.LabelProperties.Position=MapXLib.PositionConstants.miPositionBC;
    layer.LabelProperties.Style.TextFont.Size=10;
    layer.LabelProperties.Offset=4;
    layer.AutoLabel =true;
}
```

4.2.1.2 程序说明

(1)程序中假定 `layer` 层在生成时已生成与之绑定的数据集，其名称为 `"ds"+layerName`。具体作法请参见实例新建自定义图层。

(2)用 `LabelProperties` 对象来定义标注的内容，字体，位置等。例子中用数据集的 `name` 字段的内容来标注图元，标注位置在图元下方偏移 4 个单位，字体大小为 10。

(3)`layer.AutoLabel =true` 打开自动标注功能，若要关闭所有标注可简单地置标注的 `visible` 属性为 `false`。

4.3 MapX 地图集

4.3.1 什么是 MapX 地图集(Geoset)?

地图集即图层的集合，**MapX** 控件的 `Geoset` 属性即为要打开的地图集对象。地图集对象的扩展名为 `.gst`。它可由 **MapX** 所带的工具 `Geoset Manager` 来生成，请读者自己一试。

`Geoset` 保留地图图层及其设置的集合，以便于您使用。`Geoset` 是由同一地理区域的标准 `MapInfo` 格式地图文件（`.tab`）组成的数据集，因此命名为 `Geoset`。`Geoset` 可以帮助您避免在每次要作为示例地图处理图层时要分别打开和显示这些图层的耗时的工作。

`.gst` 是包含若干元数据关键字的文本文件，告诉 **MapX** 显示哪些表以及如何显示它们。

在打开一个 `Geoset` 时，它自动默认显示打开在 `Geoset` 中包括的所有文件。开发人员可以更改该默认显示以满足自身的要求。`Geoset` 的设置包括投影、默认缩放、对象的自动加标签、缩放图层以及在打

开时表是否可见。MapX 也将打开开发人员指定的任何单个 (. tab) 地图文件。Geoset 是出于方便目的提供的, 不是 MapX 行使功能所必需的。

4.3.2 实例 4: 打开已存在的地图集文件

4.3.3.1 程序功能

在硬盘中选择一个地图集文件.gst, 然后在 MapX 中打开, 并保存地图的初始属性如初始时的缩放比例及中心位置等。

4.3.3.2 程序实现

```
openFileDialog1.DefaultExt = "*.gst";
openFileDialog1.Filter="geoset file (*.gst)|*.gst";
openFileDialog1.ShowDialog();
if (openFileDialog1.FileName=="")
    return;
axMap1.GeoSet=openFileDialog1.FileName;
mapZoom=axMap1.Zoom;
mapCenterX=axMap1.CenterX;
mapCenterY=axMap1.CenterY;
```

4.3.2.3 程序说明:

例程中由 openFileDialog 可在运行中任意打开所需的地图集对象, 然后把该地图集文件名赋给地图的 GeoSet 属性即可。后面三行是用于保存地图集的初始缩放比例及中心位置, 以便能够在运行过程中随时恢复到起始视图状态下。

4.3.3 实例 5: 保存地图集

4.3.3.1 程序功能

在新建或添加一个图层后, 地图集改变, 在此时应保存改变后的地图集, 以方便下次一次性打开所有图层, 发挥 geoset 的作用。下面的程序实现这一功能。

4.3.3.2 程序实现

```
string geosetFileName="";
saveFileDialog1.InitialDirectory=@appDirectory;
saveFileDialog1.Filter = "geoset files (*.gst)|*.gst";
if(saveFileDialog1.ShowDialog()==DialogResult.OK)
    geosetFileName=saveFileDialog2.FileName;
else
    return;
axMap1.SaveMapAsGeoset(axMap1.Title.ToString(), @geosetFileName);
```

4.3.3.3 程序说明

- (1)appDirectory 为应用程序目录变量, 可用 Directory.GetCurrentDirectory() 来得到;
- (2)saveFileDialog 对话框来得到地图集的文件名及路径;
- (3)saveMapAsGeoset() 方法来保存地图集, 其语法为void **SaveMapAsGeoset** (**System.String** name , **System.String** fileSpec)

4.4 内置工具的使用

大多数地图绘制应用程序提供各种工具来协助完成常见的绘图任务（例如在地图上绘制线条）和导航任务（例如放大）。MapX 提供若干常见地图绘制工具，并且您还可以创建自己的定制工具。

4.4.1 使用标准工具

使用 MapX，您可以很容易地将常见工具栏按钮并入应用程序中。MapX 提供对若干常见地图绘制工具的内置支持，可实现大部分地图功能，包括：

- 令用户更改地图的比例和/或位置的导航工具（放大、缩小、平移、居中）。
- 让用户单击地图图元以给它加标签的加标签工具。
- 为用户提供各种方法来选择地图图元的一组选择工具。
- 对象创建工具，用于创建新的地图图元。

提供对修改键（SHIFT 键、CTRL 键）的内置支持的选择工具：使用选择工具的同时按住 SHIFT 键；该工具将取消选择图元；使用选择工具的同时按住 CTRL，该工具会将图元添加到选择中。只要按下修改键 MapX 就会自动显示不同的光标（加号或减号出现在该光标旁），以便用户可以理解该键的用途。

下面列出内置工具常量：

miArrowTool = 1000：单击标题或注释此外，在可编辑图层中移动选定图元或调整选定图元的大小。

miPanTool = 1001：漫游工具

miCenterTool = 1002：使...成为中心工具

miZoomInTool = 1003：地图放大工具

miZoomOutTool = 1004：地图缩小工具

miSymbolTool = 1005：符号注释工具

miTextTool = 1006：文本注释工具

miSelectTool = 1007：图元选择工具

miRadiusSelectTool = 1008：扇形选择工具

miRectSelectTool = 1009：矩形选择工具

miPolygonSelectTool = 1010：多边形选择工具

miLabelTool = 1011：标注工具

miAddLineTool = 1012：画线工具

miAddPolylineTool = 1013：画折线工具

miAddRegionTool = 1014：画区域工具

miAddPointTool = 1015：画点图元工具

4.4.3 实例 6：内置标准工具的使用

4.4.3.1 程序功能

实现地理信息系统中常见的工具栏按钮功能，并入应用程序中。

4.4.3.2 程序实现

//放大按钮

```
axMap2.CurrentTool=ToolConstants.miZoomInTool;
```

```

//恢复到初始视图按钮
axMap2.ZoomTo(this.mapZoom, this.mapCenterX, this.mapCenterY);
//漫游按钮
axMap2.CurrentTool=MapXLib.ToolConstants.miPanTool;
//增加点图元按钮
MapXLib.Style style=new MapXLib.StyleClass();
style.PickSymbol();
axMap2.DefaultStyle =style;
layerInsertion.Editable=true;
axMap2.Layers.InsertionLayer=layerInsertion;
axMap2.CurrentTool=MapXLib.ToolConstants.miAddPointTool;
//增加折线图元按钮
MapXLib.Style stylePolyLine=new MapXLib.StyleClass();
stylePolyLine.PickLine();
axMap2.DefaultStyle =stylePolyLine;
layerInsertion.Editable=true;
axMap2.Layers.InsertionLayer=layerInsertion;
axMap2.CurrentTool=MapXLib.ToolConstants.miAddPolylineTool;
//增加区域图元按钮
MapXLib.Style styleRegion=new MapXLib.StyleClass();
styleRegion.PickRegion();
axMap2.DefaultStyle =styleRegion;
layerInsertion.Editable=true;
axMap2.Layers.InsertionLayer=layerInsertion;
axMap2.CurrentTool=MapXLib.ToolConstants.miAddRegionTool;

```

4.4.2.3 程序说明

- (1)currentTool 属性用来选择当前工具;
- (2)在上面增加图元的例程中, style.PickSymbol ()方法用来取得新图元的样式, 包括大小. 线型. 颜色. 填充方案等, 以增加程序的灵活性;
- (3)特别注意在增加图元时, 不仅要设置当前图层为可编辑, 而且要把当前层指定为图层集合的可插入层, 即要指定图层集的 insertionLayer 属性。

4.5 自定义工具

MapX 内置工具给我们编程提供了很大的方便, 但在一些情况下还远远不能满足用户需求, 在这种情况下, 我们就要进行自定义工具了。

4.5.1 创建自定义工具

在您为任何应用程序创建一个定制工具时, 通常要执行三步:

- (1) 创建工具。
- (2) 编写工具处理程序 (工具实际执行功能的代码)。
- (3) 使用该工具 (令用户开始使用工具)。

4.5.2 实例 7：创建测量长度和面积自定义工具

在 GIS 系统中，长度及面积的测量是一项重要功能，它也是典型的自定义工具的使用的例子，各种书中都有相应介绍，在这里就在 c#下创建自定义长度测量及面积测量工具的例程介绍如下。

4.5.3.1 程序功能

创建两个自定义工具：测量长度及测量面积。

4.5.3.2 程序实现

//首先声明长度测量及面积测量工具常量

```
private const int miGetLength =100; //自定义用户工具：测量长度
```

```
private const int miGetArea =101; //自定义用户工具：测量面积
```

//然后创建长度测量及面积测量工具

```
axMap2.CreateCustomTool(miGetLength, MapXLib.ToolTypeConstants.miToolTypePoly,
MapXLib.CursorConstants.miCrossCursor, MapXLib.CursorConstants.miCrossCursor,
MapXLib.CursorConstants.miCrossCursor, false);
```

```
axMap2.CreateCustomTool(miGetArea, MapXLib.ToolTypeConstants.miToolTypePolygon,
MapXLib.CursorConstants.miCrossCursor, MapXLib.CursorConstants.miCrossCursor,
MapXLib.CursorConstants.miCrossCursor, false);
```

//最后在 PolyToolUsed 实现两个工具

```
private void axMap1_PolyToolUsed(object sender, AxMapXLib.CMapXEvents_PolyToolUsedEvent e)
{
    if(e.toolNum==miGetLength)
    {
        MapXLib.Points pts=(MapXLib.Points)e.points;
        MapXLib.Point pt1, pt2;
        double d=0.0;
        for(int i=1; i<pts.Count; i++)
        {
            pt1=pts._Item(i);
            pt2=pts._Item(i+1);
            d+=axMap2.Distance(pt1.X, pt1.Y, pt2.X, pt2.Y);
        }
        statusBarPanel2.Text="距离: "+d.ToString();
    }
    else if(e.toolNum==miGetArea)
    {
        MapXLib.Points pts=(MapXLib.Points)e.points;
        MapXLib.FeatureFactory dd=axMap2.FeatureFactory;
        MapXLib.Style style=axMap2.DefaultStyle;
        statusBarPanel2.Text="面积: "+dd.CreateRegion(pts, style).Area.ToString();
    }
}
```

4.5.2.3 程序说明

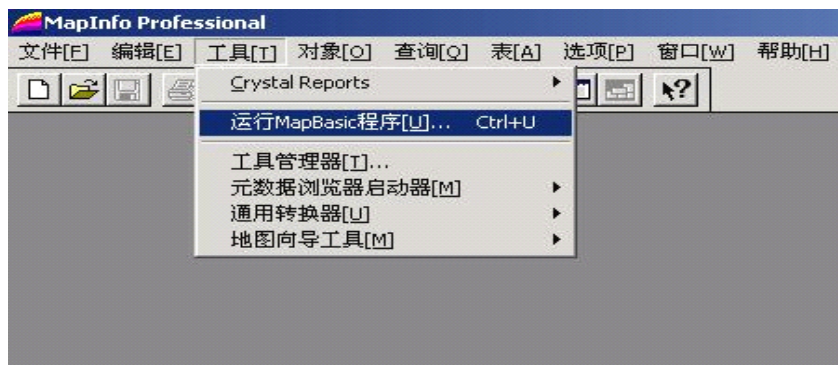
注意：这些代码一定要在 polytoolused 事件中实现，因为每种工具使用都要在多点下才能完成。这里用了 Distance 及 Area 来获得其长度和面积，其它代码请读者自己仔细品味，相信读者会读懂它，不再做解释。

4.6 MapX 地图符号样式的定制

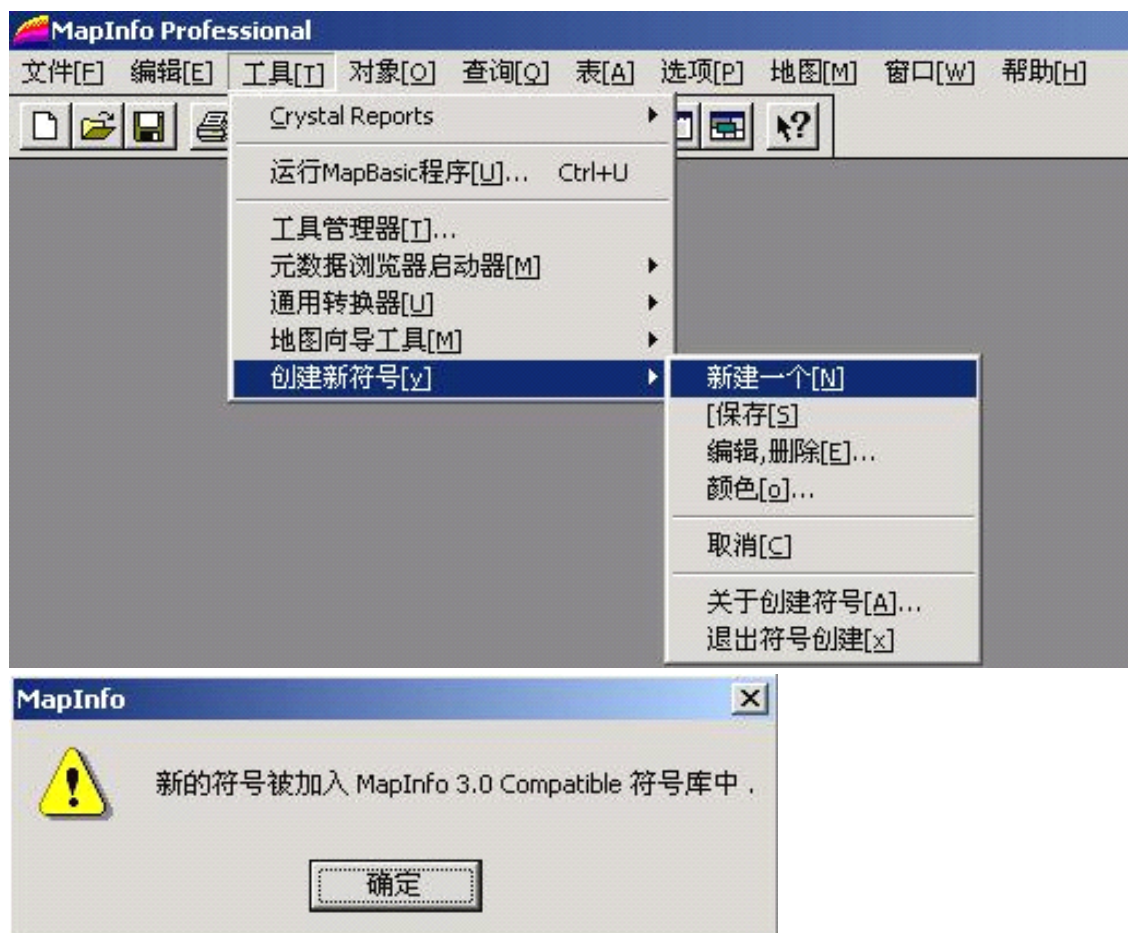
在 mapinfo 中，图元分为点线面三种，每种图元都有自己的样式库。我们用内置工作 miAddLine, miAddPoint, miAddRegion 时，都是使用各自的默认样式画出相应的点线面图元。我们可以在画之前更改其样式，以达到自己所需要的样式。这就是我们前面画图元时首先调用 style.pickSymbol(), style.pickLine(), style.pickRegion() 的原因，也可以在图层对话框中更改其样式，图层对话框的调用方法为 layerDialog();

然而，mapinfo 所带样式库还是非常有限，不可能提供所有的符号和样式，来满足各行各业的应用。如果能够对符号库进行扩充那么就非常完美了。可喜的是 mapinfo professional 自带的 mapbasic 程序 symbol.mbx 可以帮助我们完成这一工作。下面给出点符号样式的扩充方法：

如图在 mapinfo 中运行 mapbasic 程序，选择 symbol.mbx。



工具菜单下将会出现“创建新符号”，单击子菜单“新建一个”，出现符号编辑器，对新符号创建完成后按保存出现保存成功对话框。



注意：本人用的是 mapinfo professional 7.0，若为以前的版本，可能会有所不同，具体请参考相关资料。

Mapinfo 也支持位图形式的点符号，位图作为一个点显示在图层某个位置上。位图存放在 "Program Files\Common Files\MapInfo Shared\MapX Common\CUSTSYMB" 子目录下。MapX 存放在 "Program Files\MapInfo\MapX 5.0\CUSTSYMB" 子目录下。我们只需要得到符号的位图文件然后存在 CUSTSYMB 下就可完成位图号的扩充。这样，我们基本可以达到需要什么就能画什么的程度。

4.7 在图层上添加自定义图元

下面我们实现这样一个功能，在例程 1 新建的图层上画一个位图形式的点符号。分两种情况：（1）给出点的位置坐标，自动出现在图层上；（2）未给出位置坐标，由用户在图层上用鼠标点击出现。Form 上有若干文本输入框，用于输入图元各信息 number (图元编号)，caption (图元名称)，descr (图元描述)，下面一确定按钮点击出现：

4.7.1 实例 8：鼠标点击向图层上添加图元

4.7.1.1 程序功能

4.7.1.2 程序实现

private const int miAddSymbol = 106; //自定义用户工具：添加用户图元到图层，在类的变量声明部分定义

axMap2.CreateCustomTool(miAddSymbol, MapXLib.ToolTypeConstants.miToolTypePoint,
MapXLib.CursorConstants.miSizeAllCursor, MapXLib.CursorConstants.miSizeAllCursor,

```

MapXLib.CursorConstants.miSizeAllCursor, false); //创建该自定义工具, 在 form load 事件中定义
private void axMap1_ToolUsed(object sender, AxMapXLib.CMapXEvents_ToolUsedEvent e)
//在 toolUsed 实现该添加点图元自定义工具
{
if(e.toolNum==miAddSymbol)
{
//取得点击外的位置坐标
x=e.x1;
y=e.y1;
MapXLib.Point pnt=new MapXLib.PointClass();
MapXLib.RowValue rv=new MapXLib.RowValueClass();
MapXLib.RowValues rvs=new MapXLib.RowValuesClass();
MapXLib.Feature ftr;
MapXLib.FeatureFactory feaFac;
MapXLib.Style newStyle=new MapXLib.StyleClass();
feaFac = axMap2.FeatureFactory;
//定义点图元的样式
newStyle.SymbolType =MapXLib.SymbolTypeConstants.miSymbolTypeBitmap; //指定为位图样式
newStyle.SymbolBitmapSize=20; //指定图元大小
newStyle.SymbolBitmapName=bitmapfilename; //指定位图文件名
newStyle.SymbolBitmapTransparent=true; //指定位图透明, 和图层融为一体
axMap2.AutoRedraw = false;
userLayer.Editable =true;
pnt.Set(x, y);
ftr= feaFac.CreateSymbol(pnt, newStyle); //用 featurFactory 生成该位图图元
//*****
//以下代码通过 rowvalue 和 rowvalues 来为新建图元设置所有属性
//*****
dsUserLayer=axMap2.Layers._Item(1).DataSets._Item(1); //例程 1 中自动生成的 dataset.
fldsUserLayer=dsUserLayer.Fields;
rv.Dataset =dsUserLayer;
for(int j=1; j<=fldsUserLayer.Count; j++)
{
if(j==1)
{
rv.Field=fldsUserLayer._Item(j);
rv.Value=number.ToString();
}
if(j==2)
{
rv.Field=fldsUserLayer._Item(j);
rv.Value=caption;
}
}
}

```



```

}
if(j== 3)
{
rv.Field=fldsUserLayer._Item(j);
rv.Value =descr;
}
if(j== 4)
{
rv.Field=fldsUserLayer._Item(j);
rv.Value =x;
}
if(j== 5)
{
rv.Field=fldsUserLayer._Item(j);
rv.Value =y;
}
rvs.Add(rv);
}
userLayer.AddFeature(ftr, rvs); //向图层增加该图元
userLayer.Refresh();
}
//下面是对确认按钮的编程，调用刚才实现的miAddSybol 用户自定义工具
public void addUserSymbol(string infoBitmapname, string infoNumber, string infoCaption, string
infoDescr)//
{
bitmapfilename=infoBitmapname;
number=infoNumber;
caption=infoCaption;
descr=infoDescr;
axMap2.CurrentTool=(MapXLib.ToolConstants)miAddSymbol;
}

```

另外一种不需自定义用内置的 miAddTool 添加。

4.7.2 实例 9：给定坐标向图层上自动添加图元

4.7.3.1 程序功能

4.7.3.2 程序实现

这种情况下不需要自定义 miAddSymbol 工具，直接编程实现，代码大致相同，示例如下：

```

public void AddUserSymbol(ref FeatureInfo featureInfo, string layerName)
//增加到userlayer 后插入 oracle 表中，userName 必须为工作层
{
    try
    {

```

```

MapXLib.Point pnt=new MapXLib.PointClass();
MapXLib.RowValue rv=new MapXLib.RowValueClass();
MapXLib.RowValues rvs=new MapXLib.RowValuesClass();
MapXLib.Feature ftr;
MapXLib.FeatureFactory feaFac;
MapXLib.Layer layer=axMap1.Layers._Item(layerName);
MapXLib.Style newStyle=new MapXLib.StyleClass();
feaFac = axMap1.FeatureFactory;
newStyle.SymbolType =MapXLib.SymbolTypeConstants.miSymbolTypeBitmap;
newStyle.SymbolBitmapSize=20;
if (featureInfo.identity=="1")
{
newStyle.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorBlue;
    newStyle.SymbolBitmapOverrideColor=true;
}
else if(featureInfo.identity=="2")
{
    newStyle.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorGreen;
    newStyle.SymbolBitmapOverrideColor=true;
}
else
{
newStyle.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorRed ;
    newStyle.SymbolBitmapOverrideColor=true;
}
newStyle.SymbolBitmapName=featureInfo.symbolFileName;
newStyle.SymbolBitmapTransparent=true;
axMap1.AutoRedraw = false; //禁止图层自动刷新
layer.Editable =true;
pnt.Set(featureInfo.point_x, featureInfo.point_y);
ftr= feaFac.CreateSymbol(pnt, newStyle);
//*****
//以下代码通过 rowvalue 和 rowvalues 来为新建图元设置所有属性
//*****
MapXLib.Dataset dsUserLayer;
MapXLib.Fields fldsUserLayer;
dsUserLayer=axMap1.DataSets._Item("ds"+layerName);
fldsUserLayer=dsUserLayer.Fields;
rv.Dataset =dsUserLayer;
//MI_PRINX 索引构成规则: number 前 2 位加 yymmddhhmmss
DateTime myDateTime=DateTime.Now;
if (featureInfo.source==null) featureInfo.source="6";

```

```
string rownumber=featureInfo.source.PadLeft (2,
'0')+myDateTime.Year.ToString()+myDateTime.Month.ToString()
```

```
+myDateTime.Day.ToString()+myDateTime.Hour.ToString()+myDateTime.Minute.ToString()+myDa
taTime.Second.ToString();
```

```
for(int j=1; j<=fldsUserLayer.Count; j++)
{
    if(fldsUserLayer._Item(j).Name.ToUpper() == "MAINID")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value=featureInfo.featureID;
    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "SOURCE")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value=featureInfo.source;
    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "NAME")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value=featureInfo.name;
    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "IDENTITY")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.identity;
    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "DESCRIPTION")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.description;
    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "FOUNDTIME")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.foundTime;
    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "MEDIA")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.media;
```

```

    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "X")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.point_x;
    }
    if(fldsUserLayer._Item(j).Name.ToUpper() == "Y")
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.point_y;
    }
    rvs.Add(rv);
}
layer.AddFeature(ftr, rvs);
layer.Refresh();
this.InsertIntoOracle(featureInfo, rownumber, ftr.Style, "symbol", ftr,
layerName);
DeleteAllfeatures(userLayerName);
}
catch{}
axMap1.AutoRedraw = true;
}

```

4.7.2.3 程序说明

1. 位图必须放在 custSymb 子目录下，且位图不支持 24 位以上真彩色，大小最大 48 点。
 2. 本例程不仅画出了位图图元，还给出了设置图元属性的方法：
 - (1)指定 rowvalue 数据集：rv.dataset= dsUserLayer;
 - (2)通过 rowvalue.field 指定要修改的字段：rv.Field=fldsUserLayer._Item(j);
 - (3)通过 rowvalue.value 指定该字段的值：rv.Value=caption;
 - (4)把每个 rowvalue 增加到 rowvalues 集合中去，该 rowvalues 代表该图元的所用属性；
 - (5)利用图层的 addfeature(feature, rowvalues)方法增加新的图元。
- 这样我们可以用记事打开相应图层的.dat 文件查看我们增加的新图元的所有属性。

4.8 获得图元属性

在地理信息系统获取某个图元的属性是最基本的功能之一，实现起来也比较简单。

4.8.1 实例 10：获取选定图元的属性

4.8.1.1 程序功能

用内置选择工具选择某一图元，然后返回该图元的所有属性。

4.8.1.2 程序实现

```

public string[] GetSymbolProperty(string layerName)
{

```

```

MapXLib.Layer layer=null;
MapXLib.Dataset ds=null;
MapXLib.Fields fields=null;
try
{
    layer=axMap1.Layers._Item(layerName);
    ds=axMap1.DataSets._Item("ds"+layerName);
    fields=ds.Fields;
    symbolInfo=new string[fields.Count];
}
catch
{
    return null;
}
foreach(MapXLib.Feature ftr in layer.Selection)
{
    for(int i=1; i<=fields.Count; i++)
    {
        layer.KeyField=fields._Item(i).Name;
        symbolInfo[i-1]=ftr.KeyValue.ToString();
    }
}
return symbolInfo;

```

4.8.1.3 程序说明

1. 属性值的获得：首先设置图层的 KeyField 关键字段属性：layer.KeyField= ds.Fields._Item(j).Name; 然后利用图元的 keyvalue 即可得到图元该字段的值： ftr.KeyValue
2. 循环所有字段就取得图元的所有属性：
for(int i=1; i<=fields.Count; i++)
3. 在本例中将所有属性全转化为 string 类型存于字符串数组 symbolInfo[] 中。

4.9 图元的选取

图元选取利用 MapX 内置的各种选取工具即可，我们对地图的操作往往都是针对地图上所选取的图元进行的，所以许多操作都要在这个选取上作文章了。

图元的选取利用 selection 集合，非常方便，同 selectionchanged 事件结合可以做很多令人兴奋的效果。下面举得例子是利用 selection 集合及 selectionchanged 事件实现类似 infoTip 的功能，即当鼠标指在当前选中图元上时，会自动出现该图元的所有提示。这个功能当然可以用图层的 labelproperty 属性和数据集绑定实现，但我觉得下面的方法会更好。

4.9.1 实例 11：实现 InfoTip 功能

```

//tooltip 属性定义，在 form load 事件中
toolTip1.AutoPopDelay = 5000;

```

```

toolTip1.InitialDelay = 1000;
toolTip1.ReshowDelay = 500;
toolTip1.ShowAlways = true;
// SelectionChanged 事件中实现提示功能
private void axMap1_SelectionChanged(object sender, System.EventArgs e)
{
    MapXLib.Layer layer=axMap2.Layers._Item(1);
    MapXLib.Dataset ds=axMap2.DataSets._Item(1);
    foreach(MapXLib.Feature ftr in layer.Selection)
    {
        selectedFea=ftr;
        string msg="";
        for(int j=1; j<=ds.Fields.Count; j++)
        {
            layer.KeyField =ds.Fields._Item(j).Name;
            symbolProperty[j-1]=ftr.KeyValue.ToString();
            msg+= layer.KeyField+ ": " + symbolProperty[j-1]+ "\n ";
        }
        toolTip1.SetToolTip(this.axMap1, msg);
    }
}

```

注： (1)symbolProperty[] 为一字符串数组，存储图元所有属性。

(2)例子中用了 c#工具箱中的 toolTip 对象，巧妙的实现了信息提示. 关于 tooltip 对象详细说明建议读者查帮助。

4. 10 图元属性的修改

4. 10. 1 实例 12： 修改图元属性

4.10.1.1 程序功能

修改已知图元的属性。

4.10.1.2 程序实现

```

public void modUserSymbol(MapXLib.Feature ftr, string infoNumber, string infoCaption, string
infoDescr, double xIn, double yIn)
{
    if (ftr==null) return;
    number=infoNumber;
    caption=infoCaption;
    descr=infoDescr;
    x=xIn;
    y=yIn;
    MapXLib.RowValues rvs=new MapXLib.RowValuesClass();

```

```

MapXLib.Layer layer=axMap2.Layers._Item(1);
MapXLib.Dataset ds=axMap2.DataSets._Item("dsUserLayer");
for(int j=1; j<=ds.Fields.Count; j++)
{
    layer.KeyField =ds.Fields._Item(j).Name;
    if (j==1)
        ftr.KeyValue=number;
    if (j==2)
        ftr.KeyValue=caption;
    if (j==4)
        ftr.KeyValue=descr;
    if (j==6)
        ftr.KeyValue=x.ToString();
    if (j==7)
        ftr.KeyValue=y.ToString();
    ftr.Update(true, rvs);
}
MessageBox.Show("目标属性修改成功!");
}

```

图元属性修改用图元的 update 方法。

4.11 实例 13：图元的查询

图层的 search(strWhere, , [Variables])方法中以方便的实现图元的查询，具有类似 SQL 查询的强大能力。其中参数 strWhere 为检索图元的条件表达式，相当于 sql 的 where 子句。该语句执行结果返回查找到的图元集合。

下面的例程实现一个万能模糊查询，即给定一个查找关键字 keyStr，然后可查出图元属性中包含有有关键字 keyStr 的图元，而不管是哪个属性字段。

```

public void searchSymbols(string keyStr)
{
    MapXLib.Features ftrs;
    MapXLib.Features ftrs1;
    MapXLib.Layer layer=axMap2.Layers._Item(1);
    layer.Selection.ClearSelection(); //清空 selection 集合
    MapXLib.Variables vs=new MapXLib.VariablesClass();
    ftrs=layer.Search("objName LIKE"+" "+ "\""+keyStr+"%", vs);
    //查找名称属性中包含 keyStr 的图元，并存于 ftrs 集合中
    ftrs1=layer.Search("objNumber LIKE"+" "+ "\""+keyStr+"%", vs);
    //查找编号属性中包含 keyStr 的图元，并存于 ftrs1 集合中
    ftrs.Add(ftrs1); //把 ftrs1 集合合并到 ftrs 集合中
    MessageBox.Show("共有满足条件的图元"+ftrs.Count.ToString()+"个!");
    layer.Selection.Add(ftrs);
    //把查找到的所有图元存到 selection 集合中，以便使其高亮显示，处于选中状态。
}

```

```
}
```

4.12 实例 14：鹰眼图的实现

鹰眼图的实现能使电子地图在功能及界面上锦上添花，各种媒体上都有一些介绍，下面介绍用 C# 实现的鹰眼图功能。

4.12.1 程序实现

//初始时，产生鹰眼图矩形框绘画层

```
private void mapsmall()
{
    if (!File.Exists(@appDirectory+"\\mapSmall.tab"))
        m_Layer=mapSmall.Layers.CreateLayer("RectLayer", @appDirectory+
            "\\mapSmall.tab", 1, 32, mapSmall.NumericCoordSys);
    else
        m_Layer = mapSmall.Layers.Add(@appDirectory+"\\mapSmall.tab", 1);
        m_Layer.Editable=true;
        mapSmall.Layers.InsertionLayer=m_Layer;
}
private void axMap1_MapViewChanged(object sender, System.EventArgs e)
{
    if(mapSmall==null) return;
    MapXLib.Feature tempFea;
    MapXLib.Feature m_Fea;
    MapXLib.FeatureFactory feaFact;
    feaFact=mapSmall.FeatureFactory;
    MapXLib.Points tempPnts=new MapXLib.PointsClass();
    MapXLib.Style tempStyle=new MapXLib.StyleClass();
    MapXLib.Features ftrs;
    ftrs=m_Layer.AllFeatures;
    double MapX=0, mapY=0, mapWidth=0, mapHeight=0, MapX1=0, mapY1=0;
    float screenX, screenY, screenWidth, screenHeight, screenX1, screenY1;
    screenX=axMap2.Bounds.X;
    screenY=axMap2.Bounds.Y;
    screenWidth=axMap2.Bounds.Width;
    screenHeight=axMap2.Bounds.Height;
    screenX1=screenX+screenWidth;
    screenY1=screenY+screenHeight;
    axMap2.ConvertCoord(ref screenX, ref screenY, ref MapX, ref mapY,
        MapXLib.ConversionConstants.miScreenToMap);
    axMap2.ConvertCoord(ref screenX1, ref screenY1, ref MapX1, ref mapY1,
        MapXLib.ConversionConstants.miScreenToMap);
    mapWidth=MapX1-MapX;
    mapHeight=mapY1-mapY;
```



```

if (ftrs.Count==0)
{
    tempStyle.RegionPattern=MapXLib.FillPatternConstants.miPatternNoFill;
    tempStyle.RegionColor=(uint)MapXLib.ColorConstants.miColorRed;
    tempStyle.RegionBorderColor=255;
    MapXLib.Points pts=new MapXLib.PointsClass();
    pts.AddXY(MapX, mapY, 1);
    pts.AddXY(MapX+mapWidth, mapY, 2);
    pts.AddXY(MapX+mapWidth, mapY+mapHeight, 3);
    pts.AddXY(MapX, mapY+mapHeight, 4);
    tempFea = feaFact.CreateRegion(pts, tempStyle);
    m_Fea = m_Layer.AddFeature(tempFea, new MapXLib.RowValuesClass());
    m_Layer.Refresh();
    mapSmall.AutoRedraw = true;
}
else
{
    m_Fea=ftrs._Item(1);
    m_Fea.Parts._Item(1).RemoveAll();
    m_Fea.Parts._Item(1).AddXY(MapX, mapY, 1);
    m_Fea.Parts._Item(1).AddXY(MapX+mapWidth, mapY, 2);
    m_Fea.Parts._Item(1).AddXY(MapX+mapWidth, mapY+mapHeight, 3);
    m_Fea.Parts._Item(1).AddXY(MapX, mapY+mapHeight, 4);
    m_Fea.Update(true, new MapXLib.RowValuesClass());
    m_Layer.Refresh();
    mapSmall.AutoRedraw = true;
}
}
void axMap2_MouseDownEvent(object sender, AxMapXLib.CMapXEvents_MouseDownEvent e)
{
    if(mapLarge==null)return;
    double MapX=0;
    double MapY=0;
    axMap2.ConvertCoord(ref e.x, ref e.y, ref MapX, ref MapY,
                        MapXLib.ConversionConstants.miScreenToMap);
    mapLarge.CenterX=MapX;
    mapLarge.CenterY=MapY;
}

```

4. 12. 3 程序说明

- (1) 由主图视图改变影响鹰眼图矩形框的重绘，代码实现在主图的 viewchanged 事件中。
- (2) 在鹰眼中单击鼠标，把单击处的位置点作为主图的中心点重绘主图。代码实现在鹰眼图的 mousedownevent 事件中。
- (3) M_layer 为鹰眼中绘制矩形框的图层。
- (4) mapSmall 代表鹰眼图控件，mapLarge 代表主图控件。

(5) 主图与鹰眼图同步的关键是主图的可视区域地理范围与鹰眼图的矩形框所包含的地理范围一样。

首先得到主图的 Boundsr 的各个顶点的屏幕坐标用函数 axMap1.ConvertCoord() 将 bounds 的各顶点屏幕坐标转化为地理坐标。最后在鹰眼图中根据主图中各顶点的地理坐标画出矩形区域。

(6) 注意 convertCoord() 函数的用法，请参考相关资料，注意其中第五个参数的用法。

请读者仔细领会其中的机理。本例子只给了视图范围同步问题，没有给出编辑同步问题，即在主图中增加一个图元，在鹰眼图中同步增加一个点。留给读者自己练习，相信一定不会难住聪明的读者。

4.13 数据绑定

数据绑定的问题是 MapX 很重要的一个内容，是 MapX 具有生命力的重要体现。有了数据绑定，我们可以给地图赋予意义，可以给基于地理位置的空间对象以各种意义，满足各行各业的需要。然而，就是这个数据绑定，在 C# 下用得确不尽人意，这再一次体现了 MapX 对 .net 的支持不力。但是我们可以避过对我们不利的地方，来解决这个问题。下面就本人成功实现的一些数据绑定的例子，介绍如下：

(1) 用 layerinfo 对象增加层时自动产生与之绑定的数据集

其中相应语句为

```
layerInfo.AddParameter("AutoCreateDataset", 1);  
layerInfo.AddParameter("DatasetName", "dsUserLayer");
```

具体参见例程 1 部分。利用这种方法简单也非常方便，可以满足大部分应用。

(2) 利用 axMap1.DataSets.Add 方法建立数据绑定

程序代码摘要如下：

//定义 Fields 字段集

```
MapXLib.Fields flds=new MapXLib.FieldsClass();      flds.Add("objNumber", "objNumber",  
MapXLib.AggregationFunctionConstants.miAggregationIndividual,  
MapXLib.FieldTypeConstants.miTypeString);  
flds.Add("objName", "objName",  
MapXLib.AggregationFunctionConstants.miAggregationIndividual,  
MapXLib.FieldTypeConstants.miTypeString);  
flds.Add("objDscr", "objDscr",  
MapXLib.AggregationFunctionConstants.miAggregationIndividual,  
MapXLib.FieldTypeConstants.miTypeString);  
flds.Add("objX", "objX", MapXLib.AggregationFunctionConstants.miAggregationIndividual,  
MapXLib.FieldTypeConstants.miTypeFloat);  
flds.Add("objY", "objY", MapXLib.AggregationFunctionConstants.miAggregationIndividual,  
MapXLib.FieldTypeConstants.miTypeFloat);  
//数据绑定与自动标识  
userLayer=axMap1.Layers._Item(1);  
dsUserLayer=axMap1.DataSets.Add(MapXLib.DatasetTypeConstants.miDataSetLayer, userLayer,  
"userdrawlayer", 0, 0, 0, flds, false);  
userLayer.LabelProperties.Dataset =dsUserLayer;  
userLayer.LabelProperties.DataField =dsUserLayer.Fields._Item(2);  
userLayer.AutoLabel =true;  
上面代码的关键部分是 dsUserLayer=axMap1.DataSets.Add(  
MapXLib.DatasetTypeConstants.miDataSetLayer, userLayer, "userdrawlayer", 0,
```

0, 0, flds1, false);

其中 MapXLib.DatasetTypeConstants.miDataSetLayer 为绑定层类型；第二个参数为要绑定的层名，第三个参数为数据集名称，其它参数在 vb, Delphi, powerBuilder 中是可选的，然而在于 c#中参数必须写全，我们如上面处理，flds 为该层的字段集对象。

(3) 与外部数据库绑定

与外部数据库绑定的方法根据数据源的不同有很多种方法，请参看 MapX 帮助文档。下面以与 oracle 数据库绑定为例为介绍，希望能起到抛砖引玉的效果，具体绑定办法见下一章节“MapX 与 oracle 的结合”。

4.14 GPS 在 GIS 系统中的应用

近年来，卫星定位技术的飞速发展，卫星定位系统已普遍运用于物理勘探、电离层测量和航天器导航等诸多民用高新技术领域，并日益在人们的日常生活中得到普及，如移动手机定位，车载导航等。在军事领域，弹道导弹、野战指挥系统、精确弹道测量以及军用地图快速测绘等领域均大量采用了卫星导航定位技术。卫星导航技术对国民经济的发展具有极其重要意义，2000 年 10 月 31 日和 12 月 21 日我国先后成功发射了两颗导航定位试验卫星并建立了我国第一代卫星导航定位系统——“北斗导航系统”。

GPS 与 GIS 结合主要需解决两个问题：

- (1) GPS 定位信息的获取
- (2) GPS 定位信息在 GIS 中的显示

4.14.1 定位信息的接收

GPS 定位信息接收系统主要由 GPS 接收天线、变频器、信号通道、微处理器、存储器以及电源等部分组成。GPS 定位信息一般用 RS-232 串口将定位信息从 GPS 接收机传送到计算机中进行信息提取处理。

GPS 接收机只要处于工作状态就会源源不断地把接收并计算出的 GPS 导航定位信息通过串口传送到计算机中。从串口接收数据并将其放置于缓存，在没有进一步处理之前缓存中是一长串字节流，这些信息在没有经过分类提取之前是无法加以利用的。因此，必须通过程序将各个字段的信息从缓存字节流中提取出来，将其转化成有实际意义的，可供高层决策使用的定位信息数据。同其他通讯协议类似，对 GPS 进行信息提取必须首先明确其帧结构，然后才能根据其结构完成对各定位信息的提取。

4.14.2 定位信息的提取

本文以 GARMIN GPS 天线板为例来介绍 GPS 帧结构。GARMIN GPS 接收到的数据帧主要由帧头、帧尾和帧内数据组成，根据数据帧的不同，帧头也不相同，主要有"\$GPGGA"、"\$GPGSA"、"\$GPGSV"以及"\$GPRMC"等。这些帧头标识了后续帧内数据的组成结构，各帧均以回车符和换行符作为帧尾标识一帧的结束。对于通常的情况，我们所关心的定位数据如经纬度、速度、时间等均可以从"\$GPRMC"帧中获得，该帧的结构及各字段释义如下：

\$GPRMC, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, <10>, <11>*hh

<1> 当前位置的格林尼治时间，格式为 hhmmss

<2> 状态，A 为有效位置，V 为非有效接收警告，即当前天线视野上方的卫星个数少于 3 颗。

<3> 纬度，格式为 ddmm.mmmmm

<4> 标明南北半球，N 为北半球、S 为南半球

<5> 经度，格式为 dddmm.mmmmm

<6> 标明东西半球，E 为东半球、W 为西半球

- <7> 地面上的速度，范围为 0.0 到 999.9
- <8> 方位角，范围为 000.0 到 359.9 度
- <9> 日期，格式为 ddmmyy
- <10> 地磁变化，从 000.0 到 180.0 度
- <11> 地磁变化方向，为 E 或 W

至于其他几种帧格式，除了特殊用途外，平时并不常用，虽然接收机也在源源不断地向主机发送各种数据帧，但在处理时一般先通过对帧头的判断而只对"\$GPRMC"帧进行数据的提取处理。如果情况特殊，需要从其他帧获取数据，处理方法与之也是完全类似的。由于帧内各数据段由逗号分割，因此在处理缓存数据时一般是通过搜寻 ASCII 码"\$"来判断是否是帧头，在对帧头的类别进行识别后再通过对所经历逗号个数的计数来判断出当前正在处理的是哪一种定位导航参数，并作出相应的处理。

4.14.3 定位信息在 MapX 中的显示

在 MapX 应用程序中我们只需把接收到的经纬度值赋给所定位图元的位置属性，即可观察到图元的移动轨迹。

4.14.4 实例 15: GPS 定位系统的应用

4.14.4.1 程序功能

演示 GPS 定位信息的接收提取。

4.14.4.2 程序实现

在 Form 上添加串口通信控件及若干按钮和输入框，如下图：



然后编写各功能代码，代码如下：

```
namespace GPS
{
    public struct GPSInfo
    {
        public double latitude;           //纬度(分)
        public char southornorth;         //北纬(true)或南纬(false)
        public double longitude;          //经度(分)
        public char eastorwest;           //东经(true)或西经(false)
        public string height;             //高度(米)
    }
}
```

```

        public DateTime acceptTime;    //接收时间
        public string  speed;          //车辆的速度
        public bool valid;             //是否可信
    }
public class Form1 : System.Windows.Forms.Form
{
    private System.Windows.Forms.TextBox textBox4;
    private System.Windows.Forms.TextBox textBox3;
    private System.Windows.Forms.TextBox textBox2;
    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.Label label5;
    private AxMSCommLib.AxMSComm axMSComm1;
    private System.Windows.Forms.Button startRecieve;
    private System.Windows.Forms.Button stopRecieve;
    GPSInfo gif=new GPSInfo ();
    private System.Windows.Forms.RichTextBox richTextBox1;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.TextBox textBox5;
    private System.ComponentModel.Container components = null;
    public GPSInfo GetGPSInfo()
    {
        return gif;
    }
    //开始接收
    private void startRecieve_Click(object sender, System.EventArgs e)
    {
        if(!axMSComm1.PortOpen)
            axMSComm1.PortOpen=true;
        axMSComm1.Update();
    }
    ////停止接收
    private void stopRecieve_Click(object sender, System.EventArgs e)
    {
        if(axMSComm1.PortOpen)
            axMSComm1.PortOpen=false;
        axMSComm1.Update();
    }
    //对接收到的GPS信息进行提取，并显示在表单中
    private void axMSComm1_OnComm(object sender, System.EventArgs e)
    {
        string m_comdata ="";

```

```

m_comdata=(string)axMSSComm1.Input;
int length, i;
byte []data=new byte [1024];
length=m_comdata.Length;
string m_zjz;
m_zjz=m_comdata ;
//寻找GPS信号的头标志
int s;
s=m_zjz.IndexOf("$GPRMC, ");
string m_gps;
//NUM为所提取GPS信号的长度
m_gps=m_zjz.Substring(s, m_zjz.Length-s);
int x;
x=m_gps.Length ;
string m_sTime="", m_sPositionY="", m_sPositionX="", m_sDate="";
char ns=' ', ew=' ';
string spd="";
bool valid=false;
int index=0;
for(i=0; (index<11) && (m_gps[i]!='\0'); i++) //帧尾
{
    if(m_gps[i]==', ') //逗号计数
        index++;
    else
    {
        switch(index)
        {
            case 1: //提取出时间
                m_sTime+=m_gps[i];
                break;
            case 2:
                //判断数据是否可信(当GPS天线能接收到有3颗GPS卫星时为A, 可信)
                if(m_gps[i]=='A')
                {
                    valid=true;
                }
                else
                    valid=false;

                break;
            case 3: //提取出纬度
                m_sPositionY+=m_gps[i];
                break;
            case 4: //南北纬

```

```

        ns=m_gps[i];
        break;
    case 5: //提取出经度
        m_sPositionX+=m_gps[i];
        break;
    case 6: //东西经
        ew=m_gps[i];
        break;
    case 7:
        spd+=m_gps[i];
        break;
    case 9: //提取出日期
        m_sDate+=m_gps[i];
        break;
    default:
        break;
    }
}
}
//时间
int day=Int32.Parse (m_sDate.Substring (0, 2));
int month=Int32.Parse (m_sDate.Substring (2, 2));
int year=Int32.Parse (m_sDate.Substring (4, 2))+2000;
int hour1=Int32.Parse (m_sTime.Substring (0, 2));
int minute1=Int32.Parse (m_sTime.Substring (2, 2));
int second1=Int32.Parse (m_sTime.Substring (4, 2));
int hour=23-hour1;
int minute=59-minute1;
int second=60-second1;
gif.acceptTime =new DateTime (year, month, day, hour, minute, second);
gif.acceptTime .AddHours (8);
//有效性
gif.valid =valid;
if(!gif.valid)
{
    MessageBox.Show("接收卫星太少，不能定位！");
}
//南北纬
gif.southornorth =ns;
gif.speed=spd;
//东西经
gif.eastorwest =ew;
//纬度

```

```

        gif.latitude=Int32.Parse (m_sPositionY.Substring (0, 2))+double.Parse
(m_sPositionY.Substring (2, 7))/60;    //前面是度，后面是分
        //经度
        if(m_sPositionX.Length ==10) //超过九十度
        {
            gif.longitude=Int32.Parse (m_sPositionX.Substring (0, 3))+double.Parse
(m_sPositionX.Substring (3, 7))/60;
        }
        if(m_sPositionX.Length ==9) //小于九十度
        {
            gif.longitude=Int32.Parse (m_sPositionX.Substring (0, 2))+double.Parse
(m_sPositionX.Substring (2, 7))/60;
        }
        textBox2.Text=gif.latitude.ToString().Substring(0, 7)+" "+gif.southornorth.ToString();
        textBox1.Text=gif.longitude.ToString().Substring(0, 8)+" "+gif.eastorwest.ToString();
        textBox3.Text=gif.acceptTime.ToString();
        textBox4.Text=gif.speed.ToString();
        int s2;
        s2=m_zjz.IndexOf("$GPGGA, ");
        string m_gpssheigth=m_zjz.Substring (s2, m_zjz.Length -s2);
        int index1=0;
        int l=0;
        string gps_height="";
        for(l=0; (index1<11) && (m_gps[l]!=10); l++) //帧尾
        {
            if(m_gps[l]==' ', ')
                index1++;
            else
            {
                switch(index1)
                {
                    case 1:
                        break;
                    case 2:
                        break;
                    case 3:
                        break;
                    case 4:
                        break;
                    case 5:
                        break;
                    case 6:
                        break;
                    case 7:

```



```

        break;
    case 8:
        break;
    case 11:
        gps_height+=m_gps[l];
        break;
    }
}
}
gif.height=gps_height;
textBox5.Text=gif.height.ToString();
}
//串口配置初始化
private void Form1_Load(object sender, System.EventArgs e)
{
    axMSComm1.CommPort=1;
    axMSComm1.Settings="4800, N, 8, 1";
    axMSComm1.InputLen=0;
    axMSComm1.RThreshold=100;
}
}
}

```

4.15 多媒体信息在 GIS 系统中的应用

将多媒体数据嵌入 GIS 系统，必将大大提升 GIS 系统的功能，同时提高了用户的音视觉效果。

4.15.1 GIS 中嵌入多媒体的方法

在电子地图系统体中嵌入多媒体信息的方法有两种：

1. 在数据库中存储多媒体数据所对应文件的路径和文件名，在需要时检索出该多媒体文件，然后用内置功能或外挂程序来播放。
2. 直接存储多媒体数据的二进制格式到数据库，在需要时将该数据读出进行处理。

下面我们利用比较简便的第一种方法，将地图上的图元同多媒体数据文件关联起来，多媒体信息限定为文字、图片、视频和声音四种。

4.15.2 实例 16：在 MapX 系统中嵌入多媒体数据

4.15.2.1 实现思路

在新建图层时增加一字段，专门存储多媒体数据的路径，在访问图元时，读取该多媒体字段内容，根据文件类型，启动相应的打开程序来打开多媒体文件。

4.15.2.2 程序实现

```

using System.Diagnostics;
//建立多媒体功能的图层

```

```

MapXLib.Fields flds=new MapXLib.FieldsClass();
flds.AddStringField("name", 50, false);
.....
flds.AddStringField("media", 50, false); //存储多媒体文件路径
.....
layerInfo.AddParameter("Fields", flds);

layer = axMap1.Layers.Add(layerInfo, 1);
//读取多媒体文件并用相应程序打开
.....
layer.KeyField=flds._Item("media").Name;
string mediaFileName=ftr.KeyValue;
int startindex=symbol.IndexOf(".")+1;
if (substring(mediaFileName, startindex, 3)=="txt")//读取多媒体文件扩展名
Process.Start("notepad.exe", mediaFileName);
elseif(substring(mediaFileName, startindex, 3)==" .jpg" )
Process.Start("mspaint.exe", mediaFileName);
else
Process.Start("msplayer2.exe", mediaFileName);

```

4.15.2.3 程序说明

1. substring(mediaFileName, startindex, 3): 用来判断多媒体文件的类型, 根据不同类型, 用不同的应用程序打开。
2. Process Start (String fileName , String arguments): 通过指定应用程序的名称和一组命令行参数来启动进程资源, 并将该资源与新的 System.Diagnostics.Process 组件关联。注意在使用该组件时一定要添加对 System.Diagnostics的引用。

第五章 MapX 与 Oracle 结合

把 MapX 与 oracle 的结合作为单独一章节，因为笔者觉得它非常重要。Oracle 作为数据库界的泰斗，其强大面向对象的数据库功能如果能和 MapX 结合起来，将起到珠联璧合的作用。

我们知道，地理信息系统处理的数据包括两部分，一是与该图元所占据的空间位置相关的空间数据，另一个是该图元的属性数据。一般作法是将两部分分开处理：空间数据以文件形式存储，如 mapinfo 的 .map 文件；属性数据以关系数据库形式存储或以文本文件形式存储，如 mapinfo 的 .dat 文件。这种方式在单用户的情况下，基本上可以满足要求，然而对于多用户下，在分布式网络平台上使用的 GIS 系统，就难以满足地理信息共享，并发控制及安全性的要求。如果能找到一种数据库，把空间数据和属性数据统一存储管理，将会对 GIS 系统的应用提高到一个更加高的水平上来。我们称具有这种功能的数据库为空间数据库。而 Oracle 数据库就有这样一种能力来实现空间数据的存储。

5.1 Oracle 数据库对 GIS 的支持

为什么要在 MapX 用 Oracle，避开其卓越性能及缜密的安全性不谈，我认为它只少有两点非常适合 mapinfo 产品。

5.1.1 面向对象的数据库支持

Oracle 数据库支持面向对象的功能，即其中表的一个单元格可存储一个对象，打破了数据库表中字段不可再分的传统理论。恰好地图上的每个图元它是一个空间对象。oracle 具有 GEO 数据类型支持，为存储图元空间点线面属性提供了支持。

5.1.2. Oracle spatial 组件的引入

Oracle Spatial 是部署企业范围的空间信息系统和基于 web 以及基于位置的无线应用程序的基础。它为位置信息提供数据管理，允许用户直接在他们的应用程序和服务中轻松插入位置信息，操作地理和位置数据的语法与应用于 CHAR、DATE 或 INTEGER 类型的相同。

Oracle9i 和 Oracle Spatial 的具体特性包括(摘自 oracle 官方网站)：

- 针对所有函数和操作的开放、标准的 SQL 访问
- 空间对象类型存储，可容纳几何类型和线性引用
- 空间操作和函数，包括层限制和集合（例如，并集和用户定义的集合）
- 快速参考树和四叉树索引
- 综合存储、管理和使用测量数据
- 空间索引分区支持
- 强大的线性引用系统
- 支持异种数据无缝集成（融合）的工具，包括投影管理和坐标转换
- 与 Oracle9i Application Server 无线版本集成

本章将分几个专辑完整介绍 MapX 与 oracle 如何连接、MapX 数据如何存入 oracle 以及 oracle 数据如何显示到 MapX 中等一系列应用，并通过由浅入深的方法，一步一步引领大家进入 MapX 与 oracle 联合后产生的神奇境界，希望能给读者有所帮助。

5.2 循序渐进学习 Oracle Spatial 在 MapX 中的应用

Oracle8 是面向对象的关系型数据库管理系统，作为一个对象可以放在表的一格中，Oracle Spatial 提供了对空间几何对象的存储机制。面向对象的数据应用以及空间几何对象的存储对于未接触过的读者都感觉非常陌生，不知从何入手，本章节将通过一个实例带领大家一步一步走进这个世界，由表入里，由浅入深，从实践到理论，再从理论到实践，使大家把握问题的本质，掌握解决问题的方法。

5.2.1 oracle 服务器的安装

安装 oracle 服务器，注意选择 oracle spatial 组件，默认为选择。本人安装的是 oracle 8i enterprise edition 8.2.7.0.0，建立一用户全局数据库 caiqin，system 帐户口令 manager。

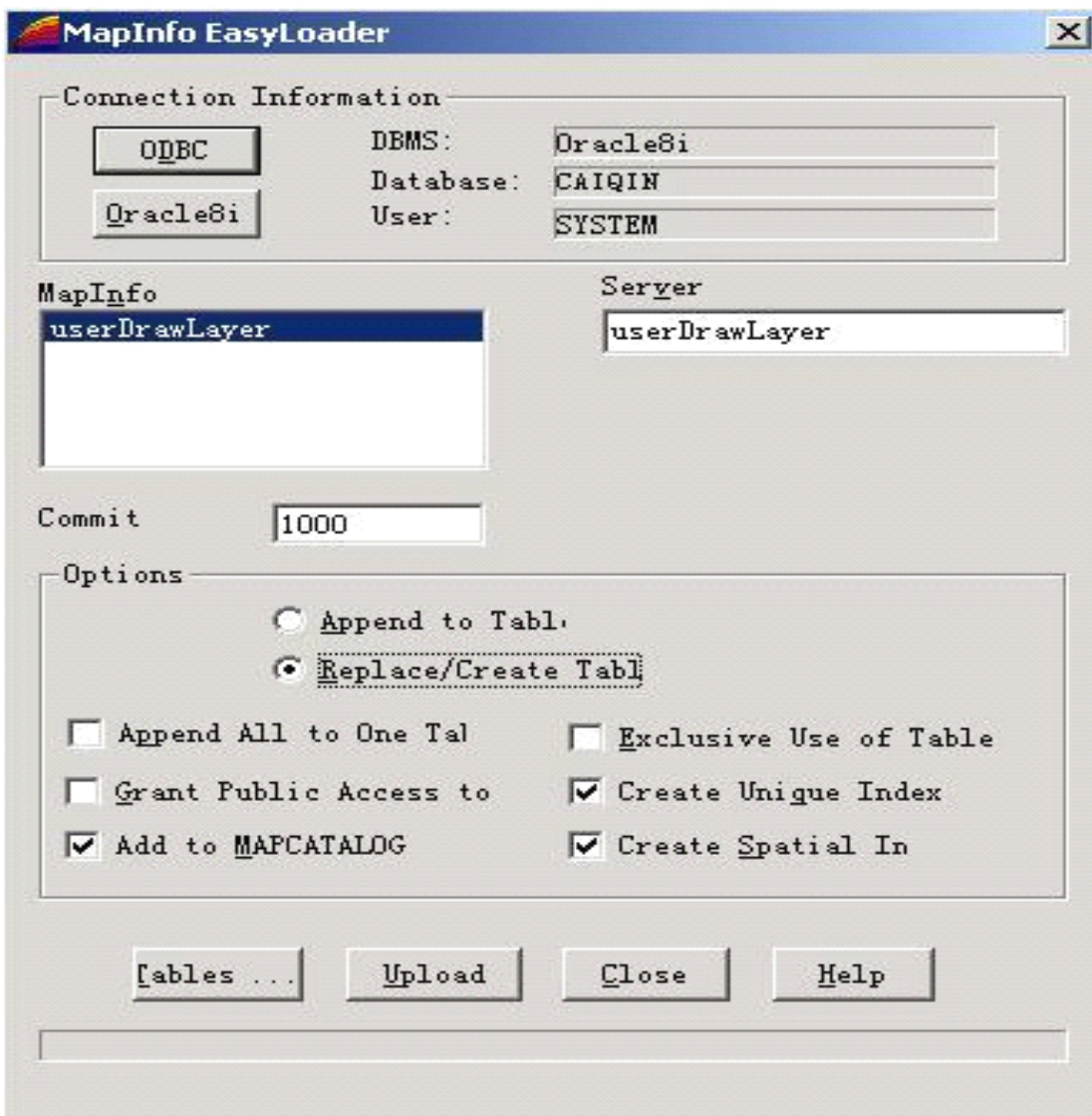
5.2.2 准备由 Oracle Spatial 存储的图层文件

以第四章例程 1 的方法建立一新层，并在新层上加入一些图元，并输入其属性数据。这样会得到 .tab, .dat, .map 等文件。打开 .tab 文件可看到自定义的图层属性字段结构定义，打开 .dat 文件可看到图层上各图元的属性信息按先后顺序存放。

5.2.3 Easyloader 上载工具

在网上下载一个与 oracle 版本相对应的图层数据上载软件 Easyloader，下载地址 www.mapinfo.com。Mapinfo Professional 中也会自带。

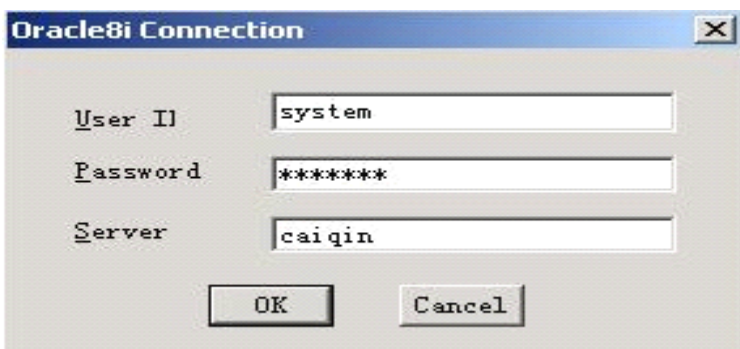
Easyloader 是用来实现把 Mapinfo 地图数据一次性向 oracle 数据库上载的软件，建议读者在命令行带参数运行该程序，如下所示 C:\easyload\easyloader /y。为什么带参数 y 运行后面再做解释。运行后界面如下：



要上载地图数据需要采取以下步骤：

5.2.3.1 连接服务器

连接服务器有几种方式：ODBC 和 Oracle8i。在这里我们选择用 Oracle8i 来连接 Oracle Spatial 服务器，单击 Oracle8i button，将会提示输入 Oracle8i 连接信息如下图（若为客户机，机器上一定要有 Oracle8i 客户端程序）。



User ID: system

Password: manager

Server: caiqin

确认输入正确后，单击 ok button，连接信息将会填充到 DBMS，Database 及 user 域。

5.2.3.2 选择要上载的 mapinfo 表文件

单击 Tables button，选择例程 1 创建的新图层的 userdrawlayer. tab 文件，该文件将会出现在 MapInfo 列表框中，server 域为该文件上载到 Oracle 数据库后的表名，默认与 .tab 文件名一样。

5.2.3.3 选择上载参数

(1) Append to Table/Replace(Create) Table: 追加到已有表还是新建表或替代已有表。选择第二项新建或替代表。

(2) Append All to One: 把所有 mapinfo 表追加到数据库中的一个表中，前提是所有的表结构要一致，默认。

(3) Grant Public Access: 对服务器数据表授予公开访问权限，默认。

(4) Exclusive Use of Tables: 加速表的上载时间，默认。

(5) Create Unique Index: 为上载后的表创建唯一索引，该索引列的字段名为 mi_prinx；选择该选项。

(6) Create Spatial Index: 创建空间索引，加速空间对象的查找，默认。

(7) Add to Mapcatlog: Mapcatlog 是位于服务器中的一个表，它为上载后的各个表创建了一个目录，上载后的各个表均在此表中注册。如果该目录表不存在， Easyloader 将会自动产生一个。

5.2.3.4 上载表

单击 Upload button 开始上载过程，并显示上载进度，若完成，显示完成信息。

5.2.4 图层信息在 Oracle 中的存储结构

上载成功后，Oracle 数据库将会新增一个 MAPCATALOG 表和一个图层数据表。

5.2.4.1 MAPINFO_MAPCATALOG 表

Oracle 数据库新增了 MAPINFO 表空间，该空间中多了一个 MAPINFO_MAPCATALOG 表。该表的结构如下：

SPATIALTYPE	FLOAT(126)	空间对象类型
TABlename	CHAR(32)	oracle 表名
OWNERNAME	CHAR(32)	表的属主
SPATIALCOLUMN	CHAR(32)	表中存储空间对象的那一列的列名
DB_X_LL	FLOAT(126)	图层边界左下 x 坐标
DB_Y_LL	FLOAT(126)	图层边界左下 y 坐标
DB_X_UR	FLOAT(126)	图层边界右上 x 坐标
DB_Y_UR	FLOAT(126)	图层边界右上 y 坐标
COORDINATESYSTEM	CHAR(254)	指定图层使用的坐标系统
SYMBOL	CHAR(254)	描述图元的样式子句
XCOLUMNNAME	CHAR(32)	包含 X 坐标的列名
YCOLUMNNAME	CHAR(32)	包含 Y 坐标的列名

该表部分部分关键内容为：

TYPE	TABlename	OWNER	SPATIALCOLUMN	...	COORD	SYMBOL	...
13.3	ASIACAPS	SYSTEM	GEOLOC	Pen (1, 2, 0)	...
13.3	USERDRAWLAYER	SYSTEM	GEOLOC	Symbol ("zhs. bmp", 0, 20, 0)	...

其中第一行为笔者先前用 easyloader 作实验时上载的一个表，在此作了注册。第二行即上一章节例程 1 所建的 userdrawlayer 层上载后的表在此自动注册。symbol 列存储了空间对象的样式，关于 symbol 列的语法以后再详细介绍。我们以后会知道，这个表是必须的，因为这个表包含了每一个要地图化的 oracle 数据表中空间对象如何显示的信息。

我们再看一下第二个变化即我们上载了 userdrawlayer 层后的所得到的 userdrawlayer 表。

5.2.4.2 图层数据表

在 SYSTEM 表空间新增了 userdrawlayer 图层数据表。打开 SYSTEM 表空间，找到 userdrawlayer 数据表，该表的结构为：

OBJNUMBER	VARCHAR2(6)
OBJNAME	VARCHAR2(12)
OBJDSCR	VARCHAR2(50)
OBJX	NUMBER
OBJY	NUMBER
MI_RENDITION	VARCHAR2(254)
MI_PRINX	VARCHAR2(20)
GEOLOC	MDSYS.SDO_GEOMETRY

该表的部分内容如下：

OBJNUMBER	OBJNAME	OBJDSCR	...	MI_RENDITION	MI_PRI NX	GEOLOC
01001	市中心医院	大型中西医结合医院...	...	symbol ("hospial.bmp", 0, 20, 1)	1	
01002	解放路	全长 4 公里，西起...	...	pen (1, 1, 0)	2	
01003	西湖	著名旅游胜地，总面积...	...	Pen (1, 2, 0) Brush (2, 16777215, 16777215)	3	
...			

由该表结构我们一眼就可以看出，前五个字段就是我们在新建层时自定义的图元编号、图元名称、图元描述及图元位置，上载时把它们全部搬移过来。另外又增加了三个字段 MI_RENDITION, MI_PRINX, GEOLOC。下面较详细解释这三个字段：

1. MI_PRINX：索引字段，为每个图元产生一唯一索引，这是我们用 easyloader 上载时指定参数 create unique index 的结果。

2. MI_RENDITION：从表数据可以看出，它是用来描述图元的样式的。表中是三个代表性图元的样式，分别是医院（点图元），街道（线样式）和水域（面样式）。下面就点线面样式定义语法介绍如下：

(1) 点符号样式

点符号样式用 symbol 子句来指定，symbol 子句有三种形式：一种是指为 MapInfo 3.0-style 符号；一种是指定为 TrueType 字体符号；一种是指定为采用位图的符号。其语法列于下表：

符号类型	语法	例子
------	----	----

MapInfo 3.0-style	Symbol(shape, color, size)	Symbol(35, 0, 12)
TrueType font	Symbol(shape, color, size, font, fontstyle, rotation)	Symbol(64, 255, 12, "MapInfo Weather" , 17, 0)
bitmap	Symbol(bitmapname, color, size, customstyle)	Symbol("sign.bmp" , 255, 18, 0)

(2) 线符号样式

线符号样式采用 pen 子句，其语法为：

Pen(thickness, pattern, color)，例如：Pen(1, 2, 0)。

(3) 区域样式

区域样式用 brush 子句指定闭合区域的填充方案，其语法为：

Brush(pattern, color, backgroundcolor)，例如：Brush(2, 255, 65535)。

注意：如果要绘制一个区域图元时，要把 pen 子句和 brush 子句结合起来使用，用 pen 子句指定区域边界样式，用 brush 子句指定区域内的填充样式。

至于各子句中用到的样式常量，颜色常量请参见 MapX5.0 帮助文档中“MapX reference information”-“appendix D: constants”中的 Color Constants, FillPatternConstants, PenStyleConstants 等。

3. GEOLOC： 存储图元的空间信息，关于 GEOLOC 字段请看下一节“空间对象类型 SDO_GEOMETRY”专题。

从上面 SYSTEM.USERDRAWLAYER 数据表的内容可以看出，每一行的前面几个字段代表图元的属性数据，后面的 MI_RENDITION 字段及 GEOLOC 字段代表图元的空间对象数据，Oracle 数据库完美地把两者统一了起来，用一致的方式来管理，这就是 Oracle 提供的对象关系模型。

5.2.4.3 空间对象类型 SDO_GEOMETRY

在上一节我们用表数据编辑器打开图层数据表后，大家并没有看到上面表格中画出的 GEOLOC 列，这是因为该列是 SDO_GEOMETRY 空间对象类型，是真正存储图元空间数据的部分，在表中隐藏。我们可以通过 sql 语句观察每个图元的空间数据的值到底是什么样子。

在 sql plus 中用 sql 语句查看 GEOLOC 字段的值：

```
SQL>select GEOLOC from USERDRAWLAYER;
```

回车后将会显示图层中每一个图元的空间数据，下面选出的是点、线、面三种典型图元的 GEOLOC 值。

点图元：

```
GEOLOC(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
```

```
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-137.5192, 32.969427, NULL), NULL, NULL)
```

线图元：

```
GEOLOC(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
```

```
SDO_GEOMETRY(2002, NULL, SDO_POINT_TYPE(0, 0, NULL), SDO_ELEM_INFO_ARRAY(1, 2, 1),  
SDO_ORDINATE_ARRAY(-114.9666, .61627203, -114.9666, .61627203))
```

面图元：

```
GEOLOC(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
```

```
SDO_GEOMETRY(2003, NULL, SDO_POINT_TYPE(56.7574949, 3.18633104, NULL), SDO_ELEM_
```



```
INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(35.8301621, 16.007866, 67.6552619, -1
4.205263, 77.6848291, 16.577925, 35.8301621, 16.007866))
```

可以看出,空间实体的空间信息是存储在一个字段名为 GEOLOC 的列中,该列的类型为 SDO_GEOMETRY。下面对这种存储空间对象的列的类型定义作出语法解释。

GEOLOC 空间对象的结构定义为:

GEOLOC(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)即该对象类型字段由 SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES 五个子属性构成,下面分别作一解释:

SDO_GTYPE:指定该空间实体的类型,数值型,长度 4 位。第一位表示维数(2-2 维空间,3-3 维空间),第二位指定线性参考度量维(指定权值的维数),缺省为 0,第三四位表示图元类型(00-未定义,01-点,02-直线或曲线,03-多边形,04-多种形状集合,05-多点,06-多线,07-多个多边形),例如:2001 表示二维空间中的点。

SDO_SRID:指定坐标系统代码,数值型。如 8307,代表 Longitude / Latitude (WGS 84)坐标系统。null 值表示没有指定坐标系统。

SDO_POINT(X, Y, Z):以三维组元指定点图元的空间位置。对于线图元和面图元,该属性被忽略。

SDO_ELEM_INFO:可变长数组,用来表明如何解释存储在 SDO_ORDINATES 中的数据。若为点图元,赋值为 NULL;直线图元赋值为 SDO_ELEM_INFO_ARRAY(1, 2, 1);矩形图元赋值为 SDO_ELEM_INFO_ARRAY(1, 1003, 1)。

SDO_ORDINATES:可变长数组,用来存储空间实体边界顶点的坐标。如直线 SDO_ORDINATE_ARRAY(-114.9666, .61627203, -114.9666, .61627203);三角区域 SDO_ORDINATE_ARRAY(35.8301621, 16.007866, 67.6552619, -1 4.205263, 77.6848291, 16.577925, 35.8301621, 16.007866);若为点图元赋值为 NULL。

通过上面的解释,大家就不难看懂每个图元的 GEOLOC 字段的值的含义了,同时我们也可以通过编程实现空间对象向 Oracle 数据库的存储了。

还有,上面提到建议读者带参数/y 运行 Easyloader 工具,原因是带了参数/y 后,该工具在上载时,能够自动提取出每个图元的样式,即会提取出图元 symbol 子句,pen 子句,brush 子句,表现为上载完毕后得到的 Oracle 数据表中增加了 MI_RENDITION 列,该列存储了每个图元的样式,相信大家一看就明白。

5.2.4.4 手工创建 MapCataLog 表及在 OracleSpatial 下的图层表

上面我们用上载工具 Easyloader,很方便得到了必须的 MAPINFO_MAPCATALOG 目录表以及图层 MapInfo 表在 Oracle 中存储的表结构。而这两个表,在我们以后的编程以及应用程序的运行中都离不开,所以 Easyloader 给我们提供了很大的方便,但是我们最终要放弃它,我们用它的目的,正如大家所看到的,是由它引出一系列关于空间对象数据在 Oracle 中存储必须掌握的一些概念。因为这些概念比较抽象,我们没有先讲这些抽象概念,而是先让读者看到结果,得到感性认识,然后再上升至理性认识,我想这样的方式应该更符合我们的认知习惯,更容易理解。

如果没有 Easyloader 这个工具,我们就要手工方式来创建这两个表了,表结构同上面所讲,然后用 sql 语句创建,并做必要的权限控制。以下为创建脚本:

```
//mapinfo.sql
//创建用户 mapinfo
CREATE USER "MAPINFO" PROFILE "DEFAULT" IDENTIFIED BY
"*****" DEFAULT
TABLESPACE "SYSTEM" TEMPORARY
TABLESPACE "SYSTEM" ACCOUNT UNLOCK;
```

```

//创建mapinfo.mapinfo_mapcatalog表
CREATE TABLE "MAPINFO"."MAPINFO_MAPCATALOG"("SPATIALTYPE" FLOAT(
    126), "TABLENAME" CHAR(32), "OWNERNAME" CHAR(32),
    "SPATIALCOLUMN" CHAR(32), "DB_X_LL" FLOAT(126), "DB_Y_LL" FLOAT(
    126), "DB_X_UR" FLOAT(126), "DB_Y_UR" FLOAT(126),
    "COORDINATESYSTEM" CHAR(254), "SYMBOL" CHAR(254), "XCOLUMNNAME"
    CHAR(32), "YCOLUMNNAME" CHAR(32))
    TABLESPACE "SYSTEM" PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS
    255
    STORAGE ( INITIAL 64K NEXT 64K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1)
    LOGGING;
//创建索引
CREATE UNIQUE INDEX "MAPINFO"."MAPCATALOG_IDX"
    ON "MAPINFO"."MAPINFO_MAPCATALOG"("TABLENAME", "OWNERNAME")
    TABLESPACE "SYSTEM" PCTFREE 10 INITRANS 2 MAXTRANS 255
    STORAGE ( INITIAL 64K NEXT 64K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 50 FREELISTS 1)
    LOGGING;
//授权访问mapinfo.mapinfo_mapcatalog表
grant select, insert, update on mapinfo.mapinfo_mapcatalog to public;
CREATE TABLE "SYSTEM"."USERLAYER"("SOURCE" VARCHAR2(50), "NAME"
    VARCHAR2(50), "IDENTITY" VARCHAR2(50), "DESCRIPTION" VARCHAR2(
    50), "FOUNDTIME" VARCHAR2(50), "OBJX" NUMBER, "OBJY" NUMBER,
    "MI_RENDITION" VARCHAR2(254), "MI_PRINX" NUMBER, "GEOLOC"
    "MDSYS"."SDO_GEOMETRY", "MEDIA" VARCHAR2(100))
    TABLESPACE "TOOLS" PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
    STORAGE ( INITIAL 32K NEXT 32K MINEXTENTS 1 MAXEXTENTS 4096
    PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1)
    LOGGING;

```

5.2.5 用程序实现 MapX 图元到 oracle 数据库的上载

上面大家已经看出，用 Easyloader 工具可以一次性把 MapX 图元上载至数据库保存，非常方便。然而，我们在应用程序的运行中，不可能调用 Easyloader 工具上载所有数据，往往是在运行过程中只上载那些刚增加的新的图元。因为图层在 Oracle 中的数据表的字段定义我们已经知道，我们只需要按照字段定义把图元的相应值插入到 Oracle 数据表中即可，下面这个函数就以这种思路实现。

5.2.5.1 程序功能

程序实现将图元的所有信息上载至 Oracle 图层数据表。

5.2.5.2 程序实现：

```

public void insertIntoOracle(string number, string name, string dscr, double x, double y, string
    prinx, MapXLib.Style style, string featureType, MapXLib.Feature ftr)

```

```

{
    string rendition="";
    string geoloc="";
    if (featureType=="symbol")
    {
        rendition="symbol ("+"\""+style.SymbolBitmapName+"\""+", "+
            style.SymbolBitmapColor.GetHashCode()+", "+
            style.SymbolBitmapSize+", "+style.SymbolType.GetHashCode()+")";
        //          geoloc="MDSYS.SDO_GEOMETRY(2001, NULL, MDSYS.SDO_POINT_TYPE("+
            x.ToString()+", "+y.ToString()+", "+NULL), "+NULL, NULL)";
    }
    if (featureType=="pen")
    {
        rendition="pen ("+style.LineWidth+", "+style.LineStyle.GetHashCode()+
            ", "+style.LineColor+")";
        geoloc="MDSYS.SDO_GEOMETRY(2002, NULL, MDSYS.SDO_POINT_TYPE(0, 0, NULL),
            MDSYS.SDO_ELEM_INFO_ARRAY(1, 2, 1), MDSYS.SDO_ORDINATE_ARRAY(";
        for(int i=1; i<=ftr.Parts._Item(1).Count; i++)
            geoloc+=ftr.Parts._Item(1)._Item(i).X.ToString()+", "+
                ftr.Parts._Item(1)._Item(i).Y.ToString()+", ";
        geoloc=geoloc.Substring(0, geoloc.Length-1);
        geoloc+="))";
    }
    if (featureType=="region")
    {
        rendition="Pen (1,2,0) Brush (2,16777215,16777215)"; geoloc="MDSYS.SDO_GEOMETRY(2002,
            NULL, MDSYS.SDO_POINT_TYPE("+
            x.ToString()+", "+y.ToString()+", "+NULL), "+
            "MDSYS.SDO_ELEM_INFO_ARRAY(1, 2, 1), "+MDSYS.SDO_ORDINATE_ARRAY(";
        for(int j=1; j<=ftr.Parts._Item(1).Count; j++)
            geoloc+=ftr.Parts._Item(1)._Item(j).X.ToString()+",
"+ftr.Parts._Item(1)._Item(j).Y.ToString()+", ";
        geoloc=geoloc.Substring(1, geoloc.Length-1);
        geoloc+="))";
    }
}

OracleConnection myConn = new OracleConnection(myConnString);
string myInsertQuery = "INSERT INTO USERDRAWLAYER (OBJNUMBER, OBJNAME, OBJRELATION, OBJDSCR,
OBJTIME, OBJX, OBJY, MI_RENDITION, MI_PRINX, GEOLOC) Values('"+number+"', '"+name+"',
'"+relation+"', '"+dscr+"', '"+time+"', "+x.ToString()+", "+y.ToString()+", '"+rendition+"',
'"+prinx+"', "+geoloc+")";

OracleCommand myOracleCommand = new OracleCommand(myInsertQuery);
myOracleCommand.Connection = myConn;

```

```
myConn.Open();
myOracleCommand.ExecuteNonQuery();
myConn.Close();
}
```

5.2.5.3 程序说明

(1)函数传入参数: string number, string name, string dscr, double x, double y 为图元的属性数据传入; string prinx 为图元的唯一索引, 这里由用户按照某种方式编程得到, 将会赋给表的 MI_PRINX 字段; MapXLib.Style style 为图元的类型, 是点(用 symbol 表示), 线(用 pen 表示), 还是区域(用 region 表示); MapXLib.Feature ftr 为欲插入的图元。

(2)局部变量: string rendition 存储图元的样式, 将会赋给表的 MI_RENDITION 字段; string geoloc 存储图元的空间对象数据, 将会赋给表的 GEOLOC 字段。

(3)图元的 rendition 及 geoloc 变量的计算: 根据我们前面对字段 MI_RENDITION 及 GEOLOC 的语法定义得到。ftr.Parts._Item(1)是为计算折线及区域图元的顶点个数。请读者对照语法仔细研读以上代码, 相信不难看懂。

(4)Oracle 数据库的连接: 用 OracleConnection 来连接 Oracle 服务器, 注意使用该方式连接时一定要引用命名空间 System.Data.OracleClient, 如果没有请在“项目”菜单中点击“添加引用”, 然后在 NET 选项卡中单击 System.Data.OracleClient.dll, 单击选择按钮后确定即可;

5.2.6 用程序实现 oracle 数据表数据下载至 MapX 中显示

5.2.6.1 程序功能

在 MapX 中生成一新的图层, 该图层的所有图元信息均来自 Oracle 图层数据表。

5.2.6.2 程序实现

```
//*****
//从 Oracle Spatial 加载空间数据以生成新的图层
//*****
private void oracleToLayer()
{
    MapXLib.Layer layer;
    string conn="UID=system; PWD=manager; srvr=caiqin";
    string querystr="select * from SYSTEM.USERDRAWLAYER";
    MapXLib.LayerInfo LayerInfoObject=new MapXLib.LayerInfoClass();
    LayerInfoObject.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeServer;
    LayerInfoObject.AddParameter("name", "DrawLayerFromDB");
    LayerInfoObject.AddParameter("ConnectionString", conn);
    LayerInfoObject.AddParameter("Query", querystr);
    LayerInfoObject.AddParameter("toolkit", "ORAINET");
    LayerInfoObject.AddParameter("AutoCreateDataset", 1);
    LayerInfoObject.AddParameter("DatasetName", "DrawLayerFromDB");
    layer=axMap2.Layers.Add(LayerInfoObject, 1);
    userLayer=layer;
    axMap2.Refresh();
    axMap2.AutoRedraw=true;
}
```

```

userLayer.Editable =true;
//this.getSymbolStyle();
layer.LabelProperties.Dataset =layer.DataSets._Item(1);
layer.LabelProperties.DataField =layer.DataSets._Item(1).Fields._Item(2);
    layer.LabelProperties.Position=MapXLib.PositionConstants.miPositionBC;
layer.LabelProperties.Style.TextFont.Size=10;
layer.LabelProperties.Offset=4;
layer.AutoLabel =true;
}

```

5.2.6.3 程序说明

(1) Layerinfo 对象:

这里我们再次利用 Layerinfo 对象来实现从 MapX 访问 oracle 数据, 并进行显示。下面针对 Layerinfo 对象作一简单介绍。

Layerinfo 对象描述了将要被添加层的一些信息, Layerinfo 将被作为 layers.add() 的第一个参数应用, 以实现把新层增加到 layers 集合中。它有一个 type 属性, 定义新层的类型, 取值为 LayerTypeConstants 常量, 其含义如下:

Layerinfo. type	Value	description
miLayerInfoTypeTab	0	已存在的 mapinfo tab 表
miLayerInfoTypeUserDraw	1	用户绘制图层
miLayerInfoTypeRaster	2	栅格图层
miLayerInfoTypeShape	3	Shape 文件 (. shp)
miLayerInfoTypeServer	4	数据库服务器
miLayerInfoTypeGeodictUserName	5	由 geodictionary 管理的图层
miLayerInfoTypeTemp	6	创建临时图层
miLayerInfoTypeNewTable	7	创建新的 mapinfo 图层

另外, Layerinfo 对象有许多参数来具体描述欲添加的图层, 这些参数随 type 取值的不同而不同, 具体请参见 MapX5.0 帮助文档。

本例是访问 Oracle 数据库中的地理信息, 所以 type 取值应为 miLayerInfoTypeServer, 在这种方式下, layerinfo 应包含如下参数:

参数名称	必填	参数说明
Name	Yes	新增图层的名称

ConnectionString	Yes	DBMS 连接字符串
Query	Yes	Sql 查询语句
ToolKit	Yes	工具包名称, 取值 ODBC 或 ORAINET
Cache	No	是否启用缓存管理
MBRSearch	No	是否利用最小外接矩形查找
AutoCreateDataset	No	是否自动产生数据集实现数据绑定
DatasetName	No	数据集的名称
Visible	No	新层是否可见, 默认可见

这些参数的添加方法为:

LayerInfoObject.AddParameter(参数名称, 参数取值);

针对 oracle 数据库, 参数 ConnectionString="UID=system; PWD=manager; SRVR=caiqin", 其中 UID, PWD, SRVR 为必须的连接关键字, 针对其它 ODBC 数据源, 连接关键字请使用 DSN, DRIVER, UID, PWD。格式均为 "key=value", 它们之间以 ";" 号隔开。其它参数请参见示例代码。

(2) 本例程中使用了 AutoCreateDataset 参数取值为 1, 自动产生一数据集 DrawLayerFromDB, 实现新层与 Oracle 数据库的数据绑定。

(3) 例程最后一段代码实现自动标注功能。

5.2.7 图元样式的还原

大家可能会发现, 利用上面的方法, 当 MapX 把上载表读入显示时, 所有图元都变成了同一样式, 原本各式各样的图元其本来面目都被丢失。这是因为, 我们上面提到的目录表 MAPINFO_MAPCATALOGG, 它只为每个上载表(图层)保存了一种样式作为其缺省样式, 保存在其 symbol 字段。所以当将上载表再次读入显示时, 该表中每一个图元便恢复成这种缺省样式了。明白了机理, 我们就可以用其它方法把丢失的样式找回来。

前面在上载图元时, 我们已经把每个图元的样式保存在了表的 MI_RENDITION 字段中, 所以读出时我们就可利用这个字段把每个图元的样式还原出来。下面的函数就是还原图元样式的代码。

5.2.7.1 程序功能

实现把丢失的图元样式找回来。

5.2.7.2 程序实现

```
private void getSymbolStyle()
{
    MapXLib.Dataset dsUserLayer=axMap2.DataSets._Item("DrawLayerFromDB");
    MapXLib.Fields fldsUserLayer=dsUserLayer.Fields;
    MapXLib.RowValues rvs;
    MapXLib.RowValue rv;
    foreach (MapXLib.Feature fea in axMap2.Layers._Item("DrawLayerFromDB").AllFeatures)
    {
        rvs=dsUserLayer.get_RowValues(fea);
        rv=rvs._Item("MI_RENDITION");
        string symbol=rv.Value.ToString();
    }
}
```

```

        if (symbol.Substring(0, 3)=="sym")
        {
            int startindex=symbol.IndexOf("\")+1;
            int lastindex=symbol.LastIndexOf("\");
            int len=lastindex - startindex;
            string bitmapfile=symbol.Substring(startindex, len);
            fea.Style.SymbolBitmapName=bitmapfile;
            fea.Update(true, rvs);
        }
        if (symbol.Substring(0, 3)=="pen")
        {
            int startindex1=symbol.IndexOf(", ");
            int lastindex1=symbol.LastIndexOf(", ");
            int len1=lastindex1 - startindex1-1;
            int startindexkh=symbol.IndexOf("(");
            int lastindexkh=symbol.LastIndexOf(")");
            int len2=lastindexkh - lastindex1-1;

            fea.Style.LineStyle=(MapXLib.PenStyleConstants)(Convert.ToInt16(symbol.Substring(startindex1+1, len1)));
            fea.Style.LineWidth=Convert.ToInt16(symbol.Substring(startindexkh+1, 1));
            fea.Style.LineColor=(uint)Convert.ToInt16(symbol.Substring(lastindex1+1, len2));
            fea.Update(true, rvs);
        }
    }
}

```

5.2.7.3 程序说明

- (1) DrawLayerFromDB 为从 Oracle Spatial 加载空间数据以生成新的图层的数据集，见上一例；
- (2) foreach(MapXLib.Feature fea in axMap2.Layers._Item("DrawLayerFromDB").AllFeatures) 为穷举图层中的每一个图元；
- (3) rvs=dsUserLayer.get_RowValues(fea)：得到每个图元的所有字段信息 rowvalues()；
rv=rvs._Item("MI_RENDITION")：得到每个图元的样式；
- (4) 用字符串运算函数提取出样式 SymbolBitmapName, LineStyle, linewidth, lineColor 等。
- (5) 用提取出的值修改图元的样式：fea.style；
- (6) 样式修改后，用 feature.update() 更新图元；

5.3 在网络环境下实现图层信息共享

在网络环境下，能够实现地理信息的共享将会在某些特殊场合发挥重大作用，比如，多人在不同的地域只标绘本区域内的地理信息，而大家可互相看到他人标绘的地理信息，即实现地理信息的共享，这种方法可称作协同标绘。如前所述，我们可以利用 MapX 同 OracleSpatial 结合来解决这一问题。然而我们在实践中发现，这种方法有一些不太令人满意的地方，主要有两点：一是样式丢失，需要手工还原，二是当数据集刷新时，大大增加数据库的负担，运行效率低。为此我们想绕过 OracleSpatial 的机制，通过另一种方法来解决。

下面我们以位图符号图元为例来介绍这种协同标绘的实现方法。

5.3.1 实现思路

1. 建一数据表，存储图元的属性，其结构如下：

ID	source	...	filename	x	y	C1	C2	C3	C4	...
1	C1	...	Home.bmp	old	new	new	new	...
2	C1	...	Hospital.bmp	old	new	old	old	...
3	C1	...	Home.bmp	old	modified	modified	modified	...
4	C1	...	Hospital.bmp	old	deleted	deleted	deleted	...

各字段意义如下：

ID: 图元的 featureID。

Source: 图元的来源即标绘方，表示该图元是由哪个标绘方标绘的。表中第一行表示该图元是由 C1 标绘的。字段 ID 就是由标绘方生产出该图元后提供的 FeatureID。

Filename: 位图符号图元的文件名。该文件必须存于 MapX 应用程序的 CustSymbol 目录下。

x, y: 符号图元的经纬度。

Ci: 表示各标绘方（客户端）对于该图元的访问状态。访问状态有以下四种——

old 表示标绘方 Ci 已处理该图元；

new 表示该图元为新加图元，标绘方 Ci 还未在本地添加该新图元；

modified 表示该图元被修改，标绘方 Ci 还未在本地作相应修改；

deleted 表示该图元已被删除，该标绘方还未在本地作相应删除。

表中第一行表示该图元是由 C1 标绘的，其 featureID 为 1，C2，C3，C4 等各标绘方还未在其本地添加该图元。

2. 本地编辑图元：

在本地标绘出图元并输入图元各属性信息，在数据表中按上表结构插入该图元各属性信息，并置本方访问标志为 old，其它各方访问标志为 new。

在本地修改图元属性信息后，同时在数据表中修改该图元相应信息，并置本方访问标志为 old，其它各方访问标志为 modified。

在本地删除图元后，同时在数据表中查找到该图元，并置本方访问标志为 old，其它各方访问标志为 deleted。

3. 图元信息异地同步：

各标绘方定时在数据表中查找该标绘方访问标志字段为 new 的图元，读取该图元的各种信息，在本地添加该图元；查找该标绘方访问标志字段为 modified 的图元，读取该图元的各种信息，并在本地作相应修改；查找该标绘方访问标志字段为 deleted 的图元，并在本地删除该图元；

这样定时刷新时只读取新添加的或已修改的图元，不仅实现了网络环境下异地信息的共享，而且大大提高了系统运行速度。

5.3.3 程序实现

```
public bool NewUserLayer(string layerName)
//新建自定义图层，若存在则添加到图层集中
{
    MapXLib.Layer layer;
    MapXLib.Fields flds=new MapXLib.FieldsClass();
```



```

flds.AddIntegerField("mainID", false);
//该图元在生产方图层中的 ID, 如由本机产生则为 featureID。
//否则由数据库的 ID 字段得到
flds.AddStringField("source", 50, false);
flds.AddStringField("name", 50, false);
flds.AddStringField("identity", 50, false);
flds.AddStringField("description", 50, false);
flds.AddStringField("foundTime", 50, false);
flds.AddStringField("media", 100, false);
flds.AddFloatField("X", false);
flds.AddFloatField("Y", false);
MapXLib.LayerInfo layerInfo;
layerInfo=new MapXLib.LayerInfoClass();
layerInfo.AddParameter("FileSpec", @appDirectory+"\\ "+layerName+".tab");
layerInfo.AddParameter("Name", layerName);
layerInfo.AddParameter("Fields", flds);
layerInfo.AddParameter("AutoCreateDataset", 1);
layerInfo.AddParameter("DatasetName", "ds"+layerName);
if (!File.Exists(@appDirectory+"\\ "+layerName+".tab"))
{
    layerInfo.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeNewTable;
}
else
{
    layerInfo.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeTab;
}
try
{
    layer = axMap1.Layers.Add(layerInfo, 1);
    axMap1.Refresh();
    axMap1.AutoRedraw=true;
    return true;
}
catch
{
    return false;
}
}
public void InsertIntoOracle(FeatureInfo featureInfo)
{
    string c1="new", c2="new", c3="new", c4="new", c5="new", c6="new";
    if (locate==1){c1="old"; }
    if (locate==2){c2="old"; }
    if (locate==3){c3="old"; }

```

```

        if (locate==4){c4="old"; }
        if (locate==5){c5="old"; }
        if (locate==6){c6="old"; }
        string str="INSERT INTO SYSTEM.INFO(ID, SOURCE, NAME, IDENTITY,
DESCRIPTION, FOUNDTIME, MEDIA, X, Y, BMP, C1, C2, C3, C4, C5, C6)";
        str+=" Values("; str+=featureInfo.featureID.ToString()+", "+featureInfo.source+", "
+featureInfo.name+", "+featureInfo.identity+", "+featureInfo.description+", "
+featureInfo.foundTime+", "+featureInfo.media+", "+featureInfo.point_x.ToString()+", "
+featureInfo.point_y.ToString()+", "+featureInfo.symbolFileName+", "+c1+", "+c2+", "
+c3+", "+c4+", "+c5+", "+c6+"")";
        //OracleConnection myConn = new OracleConnection(connectString);
        OracleCommand myOracleCommand = new OracleCommand(str);
        myOracleCommand.Connection = myConn;
        //myConn.Open();
        myOracleCommand.ExecuteNonQuery();
        //myConn.Close();
    }
    public void ModifyUserSymbol(FeatureInfo featureInfo , string selectedFeatureKey , string
layerName)//layerName 为 viewlayer 层的 name
    {
        string c1="modified" , c2="modified" , c3="modified" , c4="modified" , c5="modified" ,
c6="modified";
        if (locate==1){c1="old"; }
        if (locate==2){c2="old"; }
        if (locate==3){c3="old"; }
        if (locate==4){c4="old"; }
        if (locate==5){c5="old"; }
        if (locate==6){c6="old"; }

        MapXLib.Layer layer=null;
        try
        {
            layer=axMap1.Layers._Item(layerName);
        }
        catch
        {
            return;
        }
        MapXLib.Feature selectedFeature=layer.GetFeatureByKey(selectedFeatureKey);
        if (selectedFeature==null) return;
        MapXLib.RowValues rvs=new MapXLib.RowValuesClass();
        MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName);
        MapXLib.Fields fldsUserLayer=ds.Fields;
        MapXLib.Style style=selectedFeature.Style;

```

```

if (featureInfo.identity=="1")
{
    style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorBlue;
    style.SymbolBitmapOverrideColor=true;
}
else if(featureInfo.identity=="2")
{
    style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorGreen;
    style.SymbolBitmapOverrideColor=true;
}
else
{
    style.SymbolBitmapColor=0;
    style.SymbolBitmapOverrideColor=false;
}
selectedFeature.Style=style;
for(int j=1; j<=ds.Fields.Count; j++)
{
    layer.KeyField =fldsUserLayer._Item(j).Name;
    if (fldsUserLayer._Item(j).Name.ToUpper() == "SOURCE")
        selectedFeature.KeyValue=featureInfo.source;
    if (fldsUserLayer._Item(j).Name.ToUpper() == "NAME")
        selectedFeature.KeyValue=featureInfo.name;
    if (fldsUserLayer._Item(j).Name.ToUpper() == "IDENTITY")
        selectedFeature.KeyValue=featureInfo.identity;
    if (fldsUserLayer._Item(j).Name.ToUpper() == "DESCRIPTION")
        selectedFeature.KeyValue=featureInfo.description;
    if (fldsUserLayer._Item(j).Name.ToUpper() == "FOUNDTIME")
        selectedFeature.KeyValue=featureInfo.foundTime;
    if (fldsUserLayer._Item(j).Name.ToUpper() == "MEDIA")
        selectedFeature.KeyValue=featureInfo.media;
    if (fldsUserLayer._Item(j).Name.ToUpper() == "X")
        selectedFeature.KeyValue=featureInfo.point_x.ToString();
    if (fldsUserLayer._Item(j).Name.ToUpper() == "Y")
        selectedFeature.KeyValue=featureInfo.point_y.ToString();
    selectedFeature.Update(true, rvs);
}
//selectedFeature.Update(true, rvs);
axMap1.AutoRedraw=true;
//
string str="UPDATE SYSTEM.INFO SET ";
str+="NAME='"+featureInfo.name+"'";
str+=", IDENTITY='"+featureInfo.identity+"'";
str+=", DESCRIPTION='"+featureInfo.description+"'";
str+=", FOUNDTIME='"+featureInfo.foundTime+"'";

```

```

        str+=", MEDIA='"+featureInfo.media+"'";
        str+=", X='"+featureInfo.point_x.ToString();
        str+=", Y='"+featureInfo.point_y.ToString();
        str+=", C1='"+c1+"'";
        str+=", C2='"+c2+"'";
        str+=", C3='"+c3+"'";
        str+=", C4='"+c4+"'";
        str+=", C5='"+c5+"'";
        str+=", C6='"+c6+"'";
        str+="          WHERE          ID='"+selectedFeature.FeatureID.ToString()+"          AND
SOURCE='"+locate.ToString()+"'";
        //OracleConnection myConn = new OracleConnection(connectString);
        OracleCommand myOracleCommand = new OracleCommand(str);
        myOracleCommand.Connection = myConn;
        //myConn.Open();
        myOracleCommand.ExecuteNonQuery();
        //myConn.Close();
        MessageBox.Show("modified successfully");
    }
public void DeleteUserSymbol(string layerName, string selectedFeatureKey)
//删除 viewlayer 层图元, layerName 为 viewLayer 图层的 name
{
    string c1="deleted", c2="deleted", c3="deleted", c4="deleted", c5="deleted", c6="deleted";
    if (locate==1){c1="old"; }
    if (locate==2){c2="old"; }
    if (locate==3){c3="old"; }
    if (locate==4){c4="old"; }
    if (locate==5){c5="old"; }
    if (locate==6){c6="old"; }
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        return;
    }
    MapXLib.Feature selectedFeature=layer.GetFeatureByKey(selectedFeatureKey);
    if (selectedFeature==null) return;
    //
    string str="UPDATE SYSTEM.INFO SET ";
    str+="C1='"+c1+"', C2='"+c2+"', C3='"+c3+"', C4='"+c4+"', C5='"+c5+"', C6='"+c6+"' ";
    str+="WHERE ID='"+selectedFeature.FeatureID.ToString()+" AND SOURCE='"+locate.ToString();

```

```
//OracleConnection myConn = new OracleConnection(connectString);
OracleCommand myOracleCommand = new OracleCommand(str);
myOracleCommand.Connection = myConn;
//myConn.Open();
myOracleCommand.ExecuteNonQuery();
//myConn.Close();
layer.DeleteFeature(selectedFeature);
}
private void timer1_Tick(object sender, System.EventArgs e)
{
    FeatureInfo featureInfo=new FeatureInfo();
    MapXLib.Variables vs=new MapXLib.VariablesClass();
    MapXLib.Features features;
    MapXLib.Feature feature=null;
    string whereCondition, colStr="", type;
    //查询数据库中新增加的图元加到本地图层中
    locate=4;
    switch(locate)
    {
        case 1:
        {
            whereCondition="C1='new' or C1='modified' or C1='deleted'";
            colStr="C1";
            break;
        }
        case 2:
        {
            whereCondition="C2='new' or C2='modified' or C2='deleted'";
            colStr="C2";
            break;
        }
        case 3:
        {
            whereCondition="C3='new' or C3='modified' or C3='deleted'";
            colStr="C3";
            break;
        }
        case 4:
        {
            whereCondition="C4='new' or C4='modified' or C4='deleted'";
            colStr="C4";
            break;
        }
        case 5:
```

```

        {
            whereCondition="C5='new' or C5='modified' or C5='deleted'";
            colStr="C5";
            break;
        }
        case 6:
        {
            whereCondition="C6='new' or C6='modified' or C6='deleted'";
            colStr="C6";
            break;
        }
        default: whereCondition="C6='';
            break;
    }
    string str="SELECT ID, SOURCE, NAME, IDENTITY, DESCRIPTION,
    FOUNDTIME, MEDIA, X, Y, BMP, "+colStr+" FROM SYSTEM.INFO ";
    str+="WHERE "+whereCondition;
    OracleCommand myOracleCommand = new OracleCommand(str);
    myOracleCommand.Connection = myConn;
    OracleDataReader myReader=myOracleCommand.ExecuteReader();
    while(myReader.Read())
    {
        featureInfo.featureID=myReader.GetInt32(0);
        featureInfo.source=myReader.GetValue(1).ToString();
        try{featureInfo.name=myReader.GetString(2); }
        catch{featureInfo.name=""; }
        featureInfo.identity=myReader.GetString(3);
        try{featureInfo.description=myReader.GetString(4); }
        catch{featureInfo.description=""; }
        try{featureInfo.foundTime=myReader.GetString(5); }
        catch{featureInfo.foundTime=""; }
        try{featureInfo.media=myReader.GetString(6); }
        catch{featureInfo.media=""; }
        featureInfo.point_x=myReader.GetDouble(7);
        featureInfo.point_y=myReader.GetDouble(8);
        featureInfo.symbolFileName=myReader.GetString(9);
        type=myReader.GetString(10);
        string str1;
        if (type=="new")
        {
            this.AddUserSymbolOnLayer(featureInfo, "userLayer");
        }
        if (type=="modified")
        {

```

```

string searchStr="mainID="+featureInfo.featureID.ToString()+
" and source like "+ "\"%" +featureInfo.source+"%\\";
features=axMap1.Layers._Item("userLayer").Search(searchStr, vs);
if (features.Count>0)
{
feature=features._Item(1);
this.ModifyUserSymbol(featureInfo, feature.FeatureKey, "userLayer");
}
}
if(type=="deleted")
{
string searchStr="mainID="+featureInfo.featureID.ToString()+
" and source like "+ "\"%" +featureInfo.source+"%\\";
features=axMap1.Layers._Item("userLayer").Search(searchStr, vs);
if (features.Count>0)
{
feature=features._Item(1);
this.DeleteUserSymbol("userLayer", feature.FeatureKey);
}
}
}

```

//修改对应列的标志为 old，表示新增的或新修改的或新删除的图元在本机已做同步更改

```

str1="UPDATE SYSTEM.INFO SET "+colStr+"='old'";
OracleCommand myOracleCommand1 = new OracleCommand(str1);
myOracleCommand1.Connection = myConn;
myOracleCommand1.ExecuteNonQuery();
}
myReader.Close();
//MessageBox.Show(axMap1.Layers._Item(1).AllFeatures.Count.ToString());
}

```

5.3.3 程序说明：略

第六章 MapCtrl 控件的开发方法

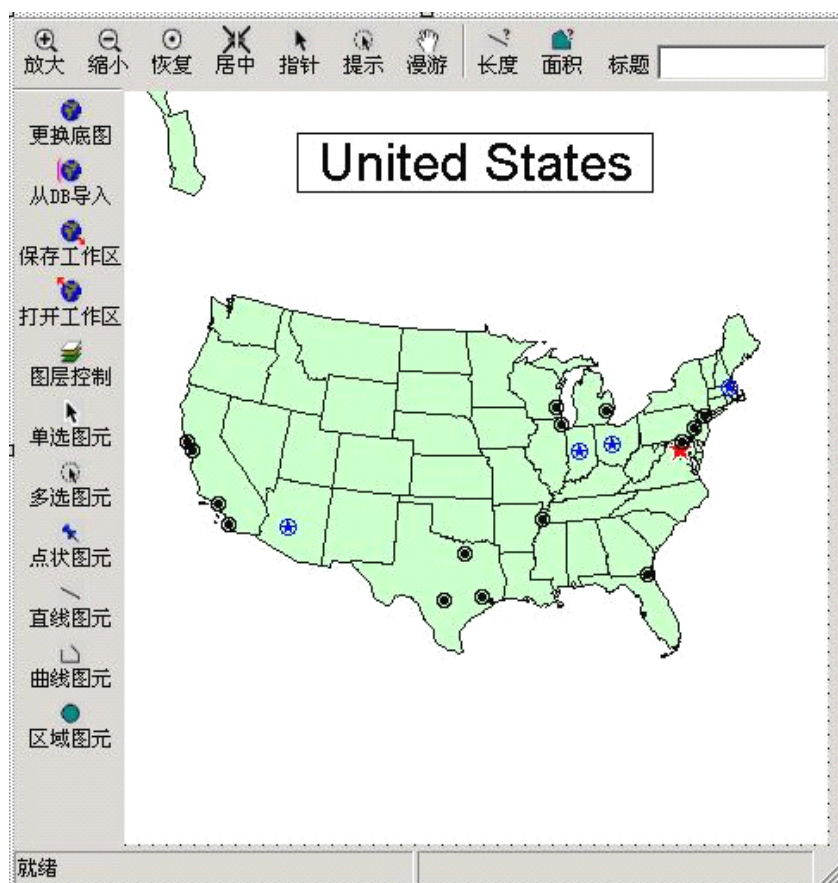
在这一章节我们对 MapX 进行二次封装，实现地理信息系统的大部分功能，从而从整体上把握地理信息系统的开发方法。

6.1 主要功能

1. 地图基本功能：放大、缩小、居中、漫游、测量
2. 新建图层、删除图层、图层控制
3. 增加图元、修改图元、删除图元、查找图元
4. 利用 OracleSpatial 把图元存于空间数据库中以便网络共享
5. 鹰眼图功能

6.2 开发步骤

1. 选择“菜单文件”——“新建项目”，打开新建项目对话框。
2. 在项目类型中选择“visual C#”类型，在模板中选择“windows 控件库”，输入控件库名称“mapCtrl”，单击“确定”，出现用户控件设计窗体。
3. 在设计窗体中添加 MapX5.0 控件，StatusBar 控件，ToolTip 控件，ToolBar 控件等，设计效果如下图：



4. 打开代码编辑器, 定义各种控件接口, 添加各种变量定义, 功能函数, 自定义事件及事件处理程序等, 完成以上所需的各种功能。

5. 设计完成后, 另存为 mapCtrl.cs。

6. 选择菜单“生成” — “生成 mapCtrl”, 生成 mapCtrl 控件库。

6.3 程序实现

整个控件的原代码摘要如下:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Data.OracleClient;
using System.Windows.Forms;
using MapXLib;
using System.IO;
namespace mapCtrl
{
    public delegate void AddSymbolDelegate ();
    public delegate void ChangeMapDelegate ();
    public delegate void MeasureDelegate(int mode, string[] value1);
    public delegate void MapRefreshDelegate();

    public struct FeatureInfo
    {
        //数据属性
        public int featureID;           //图元的唯一标示符（只读）
        public string source;           //图元来源(提供者)
        public string name;             //图元名称
        public string description;       //图元描述
        public string media;            //与图元关联的多媒体信息
        public string identity;          //图元身份(类型)
        public string foundTime;         //时间
        //符号属性
        public string symbolFileName;    //图元符号bmp文件名
        //几何属性
        public int featureType;          //图元类型：0点图元，1线图元，2面图元
        public double point_x;          //点图元的经度
        public double point_y;          //点图元的纬度
        public double[] line_xy;        //线图元的顶点坐标序列：x1, y1, x2, y2....
        public double[] polygon_xy;     //面图元的顶点坐标序列
        //扩展属性
        public string property1;
```

```

    public string property2;
    public string property3;
    public string property4;
    public string property5;
}
public struct featureRefreshProp
{
    public string featureKey;
    public string symbol;
}
public class mapCtrl : System.Windows.Forms.UserControl
{
    public AxMapXLib.AxMap axMap1;
    private AxMapXLib.AxMap mapSmall; //对鹰眼图控件的引用
    private MapXLib.Layer m_Layer;
    private double mapZoom, mapCenterX, mapCenterY;
    //保存初始缩放比例、中心点坐标
    private const int miGetLength=100; //自定义用户工具：测量长度
    private const int miGetArea=101; //自定义用户工具：测量面积
    private const int miAddSymbol=102; //
    private string geoset;
    private string appDirectory; //应用程序目录
    private string connectionString; //数据库连接字符串
    private string connStringForLayerInfo; //layerinfo 对象对数据库的连接参数
    private string[] symbolProperty; //
    private string selectedFeatureKey; //当前具有焦点的图元
    private string userLayerName;
    private System.Windows.Forms.ToolTip toolTip1;
    private System.Windows.Forms.OpenFileDialog openFileDialog1;
    private System.Windows.Forms.ContextMenu mapMenu;
    private System.Windows.Forms.MenuItem menuItemChangeGST;
    private System.Windows.Forms.MenuItem menuItemSaveGST;
    private System.Windows.Forms.MenuItem menuItemLayerCTL;
    private System.ComponentModel.IContainer components;
    public RunInfo rif=new RunInfo ();
    public event AddSymbolDelegate addSymbolDelegate;
    public event ChangeMapDelegate changeMapDelegate;
    public event MeasureDelegate measureDelegate;
    public event MapRefreshDelegate mapRefreshDelegate;
    public featureRefreshProp []frp=new featureRefreshProp [1];
    public FeatureInfo currentFeatureInfo=new FeatureInfo ();
    public mapCtrl()
    {

```

```

        // 该调用是 Windows.Forms 窗体设计器所必需的。
        InitializeComponent();
        // TODO: 在 InitializeComponent 调用后添加任何初始化
    }
    /// <summary>
    /// 清理所有正在使用的资源。
    /// </summary>
    protected override void Dispose( bool disposing )
    {
        if( disposing )
        {
            if( components != null )
                components.Dispose();
        }
        base.Dispose( disposing );
    }

    #region 组件设计器生成的代码
    /// <summary>
    /// 设计器支持所需的方法 - 不要使用代码编辑器
    /// 修改此方法的内容。
    /// </summary>
    private void InitializeComponent()
    {
        this.components = new System.ComponentModel.Container();
        System.Resources.ResourceManager resources=
        new System.Resources.ResourceManager(typeof(mapCtrl));
        this.axMap1 = new AxMapXLib.AxMap();
        this.toolTip1= new System.Windows.Forms.ToolTip(this.components);
        this.openFileDialog1= new System.Windows.Forms.OpenFileDialog();
        this.mapMenu = new System.Windows.Forms.ContextMenu();
        this.menuItemChangeGST= new System.Windows.Forms.MenuItem();
        this.menuItemSaveGST = new System.Windows.Forms.MenuItem();
        this.menuItemLayerCTL = new System.Windows.Forms.MenuItem();
        ((System.ComponentModel.ISupportInitialize)(this.axMap1)).BeginInit();
        this.SuspendLayout();
        this.axMap1.Dock = System.Windows.Forms.DockStyle.Fill;
        this.axMap1.Enabled = true;
        this.axMap1.Location = new System.Drawing.Point(0, 0);
        this.axMap1.Name = "axMap1";
        this.axMap1.OcxState=
        ((System.Windows.Forms.AxHost.State)(resources.GetObject("axMap1.OcxState")));
        this.axMap1.Size = new System.Drawing.Size(360, 296);
        this.axMap1.TabIndex = 0;
        this.axMap1.PolyToolUsed+=new

```

```

AxMapXLib.CMapXEvents_PolyToolUsedEventHandler(this.axMap1_PolyToolUsed);
    this.axMap1.SelectionChanged+=new System.EventHandler(this.axMap1_SelectionChanged);
    this.axMap1.ToolUsed+=new
        AxMapXLib.CMapXEvents_ToolUsedEventHandler(this.axMap1_ToolUsed);
    this.axMap1.MouseUpEvent+=new
AxMapXLib.CMapXEvents_MouseUpEventHandler(this.axMap1_MouseUpEvent);
    this.axMap1.MouseDownEvent+=new
AxMapXLib.CMapXEvents_MouseDownEventHandler(this.axMap1_MouseDownEvent);
    this.axMap1.MouseMoveEvent+=new
AxMapXLib.CMapXEvents_MouseMoveEventHandler(this.axMap1_MouseMoveEvent);
    this.axMap1.MapViewChanged+=new System.EventHandler(this.axMap1_MapViewChanged);
    //
    // mapMenu
    //
    this.mapMenu.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
        this.menuItemChangeGST,
this.menuItemSaveGST,
this.menuItemLayerCTL});
    //
    // menuItemChangeGST
    //
    this.menuItemChangeGST.Index = 0;
    this.menuItemChangeGST.ShowShortcut = false;
    this.menuItemChangeGST.Text = "更换地图";
    this.menuItemChangeGST.Click+=new System.EventHandler(this.menuItemChangeGST_Click);
    //
    // menuItemSaveGST
    //
    this.menuItemSaveGST.Index = 1;
    this.menuItemSaveGST.ShowShortcut = false;
    this.menuItemSaveGST.Text = "保存地图设置";
    this.menuItemSaveGST.Click+=new System.EventHandler(this.menuItemSaveGST_Click);
    //
    // menuItemLayerCTL
    //
    this.menuItemLayerCTL.Index = 2;
    this.menuItemLayerCTL.ShowShortcut = false;
    this.menuItemLayerCTL.Text = "地图图层控制";
    this.menuItemLayerCTL.Click+=new System.EventHandler(this.menuItemLayerCTL_Click);
    //
    // mapCtrl
    //
    this.Controls.Add(this.axMap1);
    this.Name = "mapCtrl";
    this.Size = new System.Drawing.Size(360, 296);

```

```

        this.Load += new System.EventHandler(this.mapCtrl_Load);
        ((System.ComponentModel.ISupportInitialize)(this.axMap1)).EndInit();
        this.ResumeLayout(false);
    }
    #endregion
    //属性获取
    public string SelectedFeatureKey
    {
        get
        {
            return selectedFeatureKey;
        }
    }
    public string Geoset
    {
        get
        {
            return geoset;
        }
        set
        {
            geoset = value;
        }
    }
    public string ConnectString
    {
        get
        {
            return connectString;
        }
        set
        {
            connectString=value;
        }
    }
    public string UserLayerName
    {
        get
        {
            return userLayerName;
        }
        set
        {
            userLayerName=value;
        }
    }
    public string ConnStringForLayerInfo

```

```
{
    get
    {
        return connStringForLayerInfo;
    }
    set
    {
        connStringForLayerInfo=value;
    }
}

public string AppDirectory
{
    get
    {
        return appDirectory;
    }
    set
    {
        appDirectory=value;
    }
}
//电子地图基本操作：放大、缩小、漫游、恢复
public void MapZoomIn()
{
    axMap1.CurrentTool=ToolConstants.miZoomInTool;
}
public void MapZoomOut()
{
    axMap1.CurrentTool=ToolConstants.miZoomOutTool;
}
public void MapPan()
{
    axMap1.CurrentTool=ToolConstants.miPanTool;
}
public void MapCenter()
{
    axMap1.CurrentTool=ToolConstants.miCenterTool;
}
public void MapArrow()
{
    axMap1.CurrentTool=ToolConstants.miArrowTool;
}
public void MapReset()
{
    axMap1.ZoomTo(this.mapZoom, this.mapCenterX, this.mapCenterY);
}
```

```

//图元选择
public void FeaturePointSelect()
{
    axMap1.CurrentTool=ToolConstants.miSelectTool;
    axMap1.AutoRedraw=true;
}
public void FeaturePolygonSelect()
{
    axMap1.CurrentTool=ToolConstants.miPolygonSelectTool;
    axMap1.AutoRedraw=true;
}
public void FeatureRadiusSelect()
{
    axMap1.CurrentTool=ToolConstants.miRadiusSelectTool;
    axMap1.AutoRedraw=true;
}
public void FeatureRectSelect()
{
    axMap1.AutoRedraw=false;
    axMap1.CurrentTool=ToolConstants.miRectSelectTool;
    //axMap1.CurrentTool=ToolConstants.miRadiusSelectTool;
}
//地图测量
public void GetLength()
{
    axMap1.CurrentTool=(MapXLib.ToolConstants)miGetLength;
}
public void GetArea()
{
    axMap1.CurrentTool=(MapXLib.ToolConstants)miGetArea;
}

private void mapCtrl_Load(object sender, System.EventArgs e)
{
    //每个图元有 7 个属性
    symbolProperty=new string[7]; //仅用于 selectionchanged 事件中显示 tooltip
    //保存应用程序目录
    appDirectory = Directory.GetCurrentDirectory();
    //保存地图初始值
    mapZoom=axMap1.Zoom;
    mapCenterX=axMap1.CenterX;
    mapCenterY=axMap1.CenterY;
    //产生自定义地图测量工具
    axMap1.CreateCustomTool(miGetArea, MapXLib.ToolTypeConstants.miToolTypePolygon,
        MapXLib.CursorConstants.miCrossCursor, MapXLib.CursorConstants.miCrossCursor,

```



```

        MapXLib.CursorConstants.miCrossCursor, false);
axMap1.CreateCustomTool(miGetLength, MapXLib.ToolTypeConstants.miToolTypePoly,
    MapXLib.CursorConstants.miCrossCursor, MapXLib.CursorConstants.miCrossCursor,
    MapXLib.CursorConstants.miCrossCursor, false);
    }
private void axMap1_PolyToolUsed(object sender, AxMapXLib.CMapXEvents_PolyToolUsedEvent e)
{
    double length=0, area=0;
    string []value1=new string [1];
    if(e.toolNum==miGetLength)
    //长度及面积为只读属性从控件的 length 和 area 属性中读取
    {
        MapXLib.Points pts=(MapXLib.Points)e.points;
        MapXLib.Point pt1, pt2;
        for(int i=1; i<pts.Count; i++)
        {
            pt1=pts._Item(i);
            pt2=pts._Item(i+1);
            length+=axMap1.Distance(pt1.X, pt1.Y, pt2.X, pt2.Y);
        }
        value1[0]=length.ToString() ;
        measureDelegate(0, value1);
    }
    else if(e.toolNum==miGetArea)
    {
        MapXLib.Points pts=(MapXLib.Points)e.points;
        MapXLib.FeatureFactory dd=axMap1.FeatureFactory;
        MapXLib.Style style=axMap1.DefaultStyle;
        area=dd.CreateRegion(pts, style).Area;
        value1[0]=area.ToString() ;
        measureDelegate(1, value1);
    }
}
//地图控制
public bool OpenGeoset(ref string geosetFileName)//参数要带扩展名.gst
{
    openFileDialog1.DefaultExt="*.gst";
    openFileDialog1.Filter="geoset file (*.gst)|*.gst";
    openFileDialog1.ShowDialog();
    geosetFileName=openFileDialog1.FileName;
    if (!File.Exists(geosetFileName))
        return false;
    else
    {

```

```

        geoset=geosetFileName;
        axMap1.GeoSet=openFileDialog1.FileName;
        axMap1 .TitleText ="";
        mapZoom=axMap1.Zoom;
        mapCenterX=axMap1.CenterX;
        mapCenterY=axMap1.CenterY;
        return true;
    }
}
public bool OpenGeoset(string geosetFileName)//参数要带扩展名.gst
{
    if (!File.Exists(geosetFileName))
        return false;
    else
    {
        try
        {
            geoset=geosetFileName;
            axMap1.GeoSet=geosetFileName;
            axMap1 .TitleText ="";
            mapZoom=axMap1.Zoom;
            mapCenterX=axMap1.CenterX;
            mapCenterY=axMap1.CenterY;
            return true;
        }
        catch
        {
            return false;
        }
    }
}
public void SaveGeoset(string geosetFileName)
{
    try
    {
        axMap1.SaveMapAsGeoset(axMap1.Title.ToString(), @geosetFileName);
    }
    catch {}
}
public void LayerCtrl()//图层控制
{
    axMap1.Layers.LayersDlg(1, 1);
}
public bool NewUserLayer(string layerName)
//新建自定义图层，若存在则添加到图层集中

```

```

{
    MapXLib.Layer layer;
    MapXLib.Fields flds=new MapXLib.FieldsClass();
    flds.AddStringField("source", 50, false);
    flds.AddStringField("name", 50, false);
    flds.AddStringField("identity", 50, false);
    flds.AddStringField("description", 50, false);
    flds.AddStringField("foundTime", 50, false);
    flds.AddFloatField("objX", false);
    flds.AddFloatField("objY", false);
    MapXLib.LayerInfo layerInfo;
    layerInfo=new MapXLib.LayerInfoClass();
    layerInfo.AddParameter("FileSpec", @appDirectory+"\\ "+layerName+".tab");
    layerInfo.AddParameter("Name", layerName);
    layerInfo.AddParameter("Fields", flds);
    layerInfo.AddParameter("AutoCreateDataset", 1);
    layerInfo.AddParameter("DatasetName", "ds"+layerName);
    if (!File.Exists(@appDirectory+"\\ "+layerName+".tab"))
    {
        layerInfo.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeNewTable;
    }
    else
    {
        layerInfo.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeTab;
    }
    try
    {
        layer = axMap1.Layers.Add(layerInfo, 1);
        axMap1.Refresh();
        return true;
    }
    catch
    {
        return false;
    }
}

public bool LoadRasterLayer(string layerName)//装载栅格图层
{
    int layerNumber=axMap1.Layers.Count;
    try
    {
        axMap1.Layers.Add(layerName+".tab", layerNumber+1);
        return true;
    }
    catch
    {

```

```

        return false;
    }
}
public void DeleteLayer(string layerName)
{
    int index;
    for(int i=1; i<=axMap1.Layers.Count; i++)
    {
        if (axMap1.Layers._Item(i).Name==layerName)
        {
            index=i;
            axMap1.Layers.Remove(index);
            return;
        }
    }
}
private void OpenDatabase(string connectionString)//打开到数据库的连接
{
    OracleConnection myConn = new OracleConnection(connectionString);
    myConn.Open();
}
public bool NewLayerTable(string layerName)
//在 oracle 中建立图层对应的表，表名等于层名，可由 easyloader 工具实现
{
    MapXLib.Fields fields=new MapXLib.FieldsClass();
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        if (layer==null) return false;
    }
    fields=layer.DataSets._Item("ds"+layerName).Fields;
    int fieldCount=layer.DataSets._Item("ds"+layerName).Fields.Count;
    string [] fieldSet=new string[fieldCount+3]; //后面为图元空间属性字段
    for(int i=1; i<=fieldCount; i++)
    {
        fieldSet[i]=fields._Item(i).Name;
    }
    string str="";
    for(int j=1; j<=fieldCount-2; j++)
    {
        str+=fieldSet[j]+" VARCHAR2(50) NULL"+" , ";
    }
}

```

```

    }
    str+="OBJX NUMBER, OBJY NUMBER, ";
    str+="MI_RENDITION VARCHAR2(50) NULL,
    MI_PRINX NUMBER, GEOLOC MDSYS.SDO_GEOMETRY NULL";
    string creatLayerTable="CREATE TABLE "+SYSTEM+"."+layerName+"("+str+)";
    string id=rif.dataBaseMap.id ;
    string password=rif.dataBaseMap.password ;
    string dataSource=rif.dataBaseMap.server ;
    connectString="user id="+id+" ; data source="+dataSource+"; password="+password;
    OracleConnection myConn = new OracleConnection(connectString);
    myConn.Open();
    OracleCommand myOracleCommand =
        new OracleCommand(creatLayerTable);
    myOracleCommand.Connection = myConn;
    try
    {
        myOracleCommand.ExecuteNonQuery();
        myConn.Close();
        return true;
    }
    catch
    {
        myConn.Close();
        return false;
    }
    return true;
}

public void AddUserSymbol(ref FeatureInfo featureInfo, string layerName)
//增加到 userlayer 后插入 oracle 表中
{
    try
    {
        MapXLib.Point pnt=new MapXLib.PointClass();
        MapXLib.RowValue rv=new MapXLib.RowValueClass();
        MapXLib.RowValues rvs=new MapXLib.RowValuesClass();
        MapXLib.Feature ftr;
        MapXLib.FeatureFactory feaFac;
        MapXLib.Layer layer=axMap1.Layers._Item(layerName);
        MapXLib.Style newStyle=new MapXLib.StyleClass();
        feaFac = axMap1.FeatureFactory;
        newStyle.SymbolType =MapXLib.SymbolTypeConstants.miSymbolTypeBitmap;
        newStyle.SymbolBitmapSize=20;
        //根据图元身份不同图元用不同的颜色区分
        if (featureInfo.identity=="0")

```

```

        {
            newStyle.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorBlue;
            newStyle.SymbolBitmapOverrideColor=true;
        }
else if(featureInfo.identity=="1")
    {
        newStyle.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorGreen;
        newStyle.SymbolBitmapOverrideColor=true;
    }
else
    {
        newStyle.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorRed ;
        newStyle.SymbolBitmapOverrideColor=true;
    }
newStyle.SymbolBitmapName=featureInfo.symbolFileName;
newStyle.SymbolBitmapTransparent=true;
axMap1.AutoRedraw = false; //禁止图层自动刷新
layer.Editable =true;
pnt.Set(featureInfo.point_x, featureInfo.point_y);
ftr= feaFac.CreateSymbol(pnt, newStyle);
/*****
//以下代码通过 rowvalue 和 rowvalues 来为新建图元设置所有属性
*****/
MapXLib.Dataset dsUserLayer;
MapXLib.Fields fldsUserLayer;
dsUserLayer=axMap1.DataSets._Item("ds"+layerName);
fldsUserLayer=dsUserLayer.Fields;
rv.Dataset =dsUserLayer;
//MI_PRINX 索引构成规则: number 前 2 位加 yymmddhhmmss
DateTime myDateTime=DateTime.Now;
if (featureInfo.source==null) featureInfo.source="6";
string rownumber=featureInfo.source.PadLeft (
2, '0')+myDateTime.Year.ToString()+myDateTime.Month.ToString()
+myDateTime.Day.ToString()+myDateTime.Hour.ToString()+myDateTime.Minute.ToString()+myDateTime.Second.ToString();
for(int j=1; j<=fldsUserLayer.Count; j++)
{
    if(j==1)
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value=featureInfo.source;
    }
    if(j==2)
    {
        rv.Field=fldsUserLayer._Item(j);

```

```

        rv.Value=featureInfo.name;
    }
    if(j==3)
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value=featureInfo.identity;
    }
    if(j==4)
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.description;
    }
    if(j==5)
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.foundTime;
    }
    if(j==6)
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.point_x;
    }
    if(j==7)
    {
        rv.Field=fldsUserLayer._Item(j);
        rv.Value =featureInfo.point_y;
    }
    rvs.Add(rv);
}
layer.AddFeature(ftr, rvs);
layer.Refresh();
this.InsertIntoOracle(featureInfo, rownumber, ftr.Style, "symbol", ftr, layerName);
DeleteAllfeatures(userLayerName);
}
catch{}
axMap1.AutoRedraw = true;
}

```

```

public void ModifyUserSymbol(FeatureInfo featureInfo, string layerName)

```

```

//修改图元属性, layerName 为 viewlayer 层的 name

```

```

{
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
}

```

```

    }
    catch
    {
        return;
    }
    MapXLib.Feature selectedFeature=layer.GetFeatureByKey(featureInfo.featureKey );
    if (selectedFeature==null) return;
    MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName);
    MapXLib.RowValues rowValues=ds.get_RowValues(selectedFeature);
    MapXLib.RowValue rowValue=rowValues._Item("MI_PRINX");
    string prinx=rowValue.Value.ToString(); //取得该图元在数据库中行的索引值
    string rendition="";
    string geoloc="";
    MapXLib.Style style=selectedFeature.Style;
    if (featureInfo.identity=="0")
    {
        style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorBlue;
        style.SymbolBitmapOverrideColor=true;
    }
    else if(featureInfo.identity=="1")
    {
        style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorGreen;
        style.SymbolBitmapOverrideColor=true;
    }
    else
    {
        style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorRed ;
        style.SymbolBitmapOverrideColor=true;
    }
    rendition="symbol"+"\""+style.SymbolBitmapName+"\"
+" ,   "+style.SymbolBitmapColor+"   ,   "+style.SymbolBitmapSize+"   ,   "+style.SymbolType+")" ;
    geoloc="MDSYS.SDO_GEOMETRY(2001, NULL,
        MDSYS.SDO_POINT_TYPE("+featureInfo.point_x.ToString()+",
        "+featureInfo.point_y.ToString()+", "+NULL), "+NULL, NULL)";
    MapXLib.Fields fields=new MapXLib.FieldsClass();
    fields=axMap1.DataSets._Item("ds"+layerName).Fields;
    int fieldCount=fields.Count;
    string [] fieldSet=new string[fieldCount+1];
    for(int i=1; i<=fieldCount; i++)
    {
        fieldSet[i]=fields._Item(i).Name;
    }
    fieldSet[fieldCount-1]="GEOLOC";
    string tableName=layerName.Substring(0, layerName.Length-6);

```



```

string str="UPDATE SYSTEM."+tableName+" SET ";
str+=fieldSet[0]+"='"+featureInfo.source+"'";
str+=", "+fieldSet[1]+"='"+featureInfo.name+"'";
str+=", "+fieldSet[2]+"='"+featureInfo.identity+"'";
str+=", "+fieldSet[3]+"='"+featureInfo.description+"'";
str+=", "+fieldSet[4]+"='"+featureInfo.foundTime+"'";
str+=", "+fieldSet[5]+"='"+featureInfo.point_x.ToString();
str+=", "+fieldSet[6]+"='"+featureInfo.point_y.ToString();
str+=", "+fieldSet[7]+"='"+rendition+"'";
str+=", "+fieldSet[8]+"='"+geoloc+"'";
str+=" WHERE MI_PRINX="+prinx;
OracleConnection myConn = new OracleConnection(connectString);
OracleCommand myOracleCommand = new OracleCommand(str);
myOracleCommand.Connection = myConn;
myConn.Open();
myOracleCommand.ExecuteNonQuery();
myConn.Close();
}

public void ModifyUserSymbol(string featureKey, double X, double Y, string layerName)
//修改图元属性, layerName 为 viewlayer 层的 name
{
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        return;
    }
    FeatureInfo featureInfo=new FeatureInfo ();
    GetFeaturePropertyByFeatureKey(layerName, ref featureInfo, featureKey);
    featureInfo.point_x =X;
    featureInfo .point_y =Y;
    MapXLib.Feature selectedFeature=layer.GetFeatureByKey(featureInfo.featureKey );
    if (selectedFeature==null) return;
    MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName);
    MapXLib.RowValues rowValues=ds.get_RowValues(selectedFeature);
    MapXLib.RowValue rowValue=rowValues._Item("MI_PRINX");
    string prinx=rowValue.Value.ToString(); //取得该图元在数据库中的索引值
    string rendition="";
    string geoloc="";
    MapXLib.Style style=selectedFeature.Style;

```

```

if (featureInfo.identity=="0")
{
    style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorBlue;
    style.SymbolBitmapOverrideColor=true;
}
else if(featureInfo.identity=="1")
{
    style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorGreen;
    style.SymbolBitmapOverrideColor=true;
}
else
{
    style.SymbolBitmapColor=(uint)MapXLib.ColorConstants.miColorRed ;
    style.SymbolBitmapOverrideColor=true;
}
rendition="symbol ("+"\""+featureInfo.symbolFileName +"\""+", "+
style.SymbolBitmapColor+", "+style.SymbolBitmapSize+", "+style.SymbolType+");
geoloc="MDSYS.SDO_GEOMETRY(2001, NULL, MDSYS.SDO_POINT_TYPE("+
featureInfo.point_x.ToString()+", "+featureInfo.point_y.ToString()+", "+NULL), "+
NULL, NULL)";
MapXLib.Fields fields=new MapXLib.FieldsClass();
fields=axMap1.DataSets._Item("ds"+layerName).Fields;
int fieldCount=fields.Count;
string [] fieldSet=new string[fieldCount+1];
for(int i=1; i<=fieldCount; i++)
{
    fieldSet[i-1]=fields._Item(i).Name;
}
fieldSet[fieldCount-1]="GEOLOC";
string tableName=layerName.Substring(0, layerName.Length-6);
string str="UPDATE SYSTEM."+tableName+" SET ";
str+=fieldSet[0]+"='"+featureInfo.source+"'";
str+=", "+fieldSet[1]+"='"+featureInfo.name+"'";
str+=", "+fieldSet[2]+"='"+featureInfo.identity+"'";
str+=", "+fieldSet[3]+"='"+featureInfo.description+"'";
str+=", "+fieldSet[4]+"='"+featureInfo.foundTime+"'";
str+=", "+fieldSet[5]+"='"+featureInfo.point_x.ToString()+";
str+=", "+fieldSet[6]+"='"+featureInfo.point_y.ToString()+";
str+=", "+fieldSet[7]+"='"+rendition+"'";
str+=", "+fieldSet[8]+"='"+geoloc+"'";
str+=" WHERE MI_PRINX="+prinx;
OracleConnection myConn = new OracleConnection(connectString);
OracleCommand myOracleCommand = new OracleCommand(str);
myOracleCommand.Connection = myConn;

```

```

myConn.Open();
myOracleCommand.ExecuteNonQuery();
myConn.Close();
}
public void DataSetRefresh(string layerName)
//与图层关联的数据集刷新
{
    MapXLib.Layer layer=axMap1.Layers._Item(layerName+"FromDB");
    MapXLib.Dataset ds;
    ds=layer.DataSets._Item("ds"+layerName+"FromDB");
    ds.Refresh();
}
//将图元的各种属性插入 Oracle 数据表
public void InsertIntoOracle(FeatureInfo featureInfo, string prinx, MapXLib.Style style, string featureType,
MapXLib.Feature ftr, string layerName)
{
    string rendition="";
    string geoloc="";
    if (featureType=="symbol")
    {
        rendition="symbol ("+"\""+featureInfo.symbolFileName +"\""+", "+
style.SymbolBitmapColor+"      ,      "+style.SymbolBitmapSize+"      ,      "+style.SymbolType+)"      ;
        geoloc="MDSYS.SDO_GEOMETRY(2001      ,      NULL      ,
MDSYS.SDO_POINT_TYPE("+featureInfo.point_x.ToString()+")      ,      "+featureInfo.point_y.ToString()+")      ,
"+"NULL), "+"NULL, NULL)";
    }
    MapXLib.Fields fields=new MapXLib.FieldsClass();
    fields=axMap1.DataSets._Item("ds"+layerName).Fields;
    int fieldCount=fields.Count;
    string [] fieldSet=new string[fieldCount+4];
    for(int i=1; i<=fieldCount; i++)
    {
        fieldSet[i-1]=fields._Item(i).Name;
    }
    fieldSet[fieldCount+1]="MI_RENDITION";
    fieldSet[fieldCount+2]="MI_PRINX";
    fieldSet[fieldCount+3]="GEOLOC";
    string str="INSERT INTO "+layerName+"(";
    str+=fieldSet[0];
    for(int j=1; j<fieldCount; j++)
    {
        str+=", "+fieldSet[j];
    }
    str+=", MI_RENDITION, MI_PRINX, GEOLOC) Values(";

```

```

        str+="\""+featureInfo.source+"\", \""+featureInfo.name+"\", \""+featureInfo.identity+"\", \""+
featureInfo.description+"\", \""+featureInfo.foundTime+"\", \""+featureInfo.point_x.ToString()+
\", \""+featureInfo.point_y.ToString()+\", \""+rendition+"\", \""+prinx+"\", \""+geoloc+"\"";
        OracleConnection myConn = new OracleConnection(connectString);
        OracleCommand myOracleCommand = new OracleCommand(str);
        myOracleCommand.Connection = myConn;
        myConn.Open();
        myOracleCommand.ExecuteNonQuery();
        myConn.Close();
    }
    //*****
    //从 Oracle Spatial 加载空间数据以生成新的图层
    //*****
    public void OracleToLayer(string tableName)
    //tableName 为 oracle 数据库表名与其对应的图层名一样
    {
        axMap1 .AutoRedraw =false;
        try
        {
            bool flag=false;
            for(int i=1; i<=axMap1.DataSets.Count ; i++)
            {
                if(axMap1.DataSets._Item(i).Name == "ds"+tableName+"FromDB")
                {
                    flag=true;
                    break;
                }
            }
            MapXLib.Layer layer;
            if(flag)
            {
                DataSetRefresh(tableName);
                axMap1.Refresh();
                getSymbolStyle(tableName+"FromDB");
            }
            else
            {
                string querystr="select * from SYSTEM."+tableName.ToUpper();
                MapXLib.LayerInfo LayerInfoObject=new MapXLib.LayerInfoClass();
                LayerInfoObject.Type=MapXLib.LayerInfoTypeConstants.miLayerInfoTypeServer;
                LayerInfoObject.AddParameter("name", tableName+"FromDB");
                LayerInfoObject.AddParameter("ConnectionString", ConnStringForLayerInfo);
                LayerInfoObject.AddParameter("Query", querystr);
                LayerInfoObject.AddParameter("toolkit", "ORAINET");
            }
        }
        catch { }
    }

```

```

        LayerInfoObject.AddParameter("AutoCreateDataset", 1);
        LayerInfoObject.AddParameter("DatasetName", "ds"+tableName+"FromDB");
        layer=axMap1.Layers.Add(LayerInfoObject, 1);
        axMap1.DataSets._Item("ds"+tableName+"FromDB").Refresh();
        axMap1.Refresh();
        layer.Editable =true;
        this.getSymbolStyle(tableName+"FromDB");
    }
}
catch
{
}
}

```

```
public void getSymbolStyle(string layerName)
```

//恢复图元样式，层名与表名一致

```

{
    MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName);
    MapXLib.Fields fldsUserLayer=ds.Fields;
    MapXLib.RowValues rvs;
    MapXLib.RowValue rv;
    featureRefreshProp []frptemp=
new featureRefreshProp [axMap1.Layers._Item(layerName).AllFeatures.Count ];
    foreach(MapXLib.Feature fea in axMap1.Layers._Item(layerName).AllFeatures)
    {
        try
        {
            rvs=ds.get_RowValues(fea);
            rv=rvs._Item("MI_RENDITION");
            string symbol=rv.Value.ToString();
            //判断相同 featureKey 是否有变化
            int i=0;
            if (symbol.Substring(0, 3)=="sym")
            {
                int startindex=symbol.IndexOf("\")+1;
                int lastindex=symbol.LastIndexOf("\");
                int len=lastindex - startindex;
                string bitmapfile=symbol.Substring(startindex, len);
                startindex=symbol.IndexOf(",")+1;
                string str="";
                for(i=startindex; symbol[i]!=';', i++)
                {
                    str+=symbol[i];
                }
            }
        }
        catch
        {
        }
    }
}

```

```

        uint SymbolBitmapColor=(uint)Convert.ToInt32(str);
        bool SymbolBitmapOverrideColor=true;
        if (str!="0")
        {
            SymbolBitmapColor=(uint)Convert.ToInt32(str);
            SymbolBitmapOverrideColor=true;
        }
        else
        {
            SymbolBitmapOverrideColor=false;
        }
        string SymbolBitmapName=bitmapfile;
        bool SymbolBitmapTransparent=true;
        fea.Style.SymbolBitmapName =SymbolBitmapName;
        fea.Style.SymbolBitmapTransparent =SymbolBitmapTransparent;
        fea.Style.SymbolBitmapOverrideColor =SymbolBitmapOverrideColor;
        fea.Style.SymbolBitmapColor =SymbolBitmapColor;
        fea.Update(true, rvs);
    }
}
catch
{}
}
frp=frptemp;
mapRefreshDelegate();    //已经刷新
}
private void getSymbolStyle(string layerName, string featureKey)//层名与表名一致
{
    MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName);
    MapXLib.Fields fldsUserLayer=ds.Fields;
    MapXLib.RowValues rvs;
    MapXLib.RowValue rv;
    try
    {
        MapXLib.Feature fea=axMap1.Layers._Item(layerName).GetFeatureByKey (featureKey);
        rvs=ds.get_RowValues(fea);
        rv=rvs._Item("MI_RENDITION");
        string symbol=rv.Value.ToString();
        if (symbol.Substring(0, 3)=="sym")
        {
            int startindex=symbol.IndexOf("\"")+1;
            int lastindex=symbol.LastIndexOf("\"");
            int len=lastindex - startindex;
            string bitmapfile=symbol.Substring(startindex, len);

```

```

//
startindex=symbol.IndexOf(",")+1;
string str="";
for(int i=startindex; symbol[i]!=';', i++)
{
    str+=symbol[i];
}
uint SymbolBitmapColor=(uint)Convert.ToInt32(str);
bool SymbolBitmapOverrideColor=true;
if (str!="0")
{
    SymbolBitmapColor=(uint)Convert.ToInt32(str);
    SymbolBitmapOverrideColor=true;
}
else
{
    SymbolBitmapOverrideColor=false;
}
string SymbolBitmapName=bitmapfile;
bool SymbolBitmapTransparent=true;
fea.Style.SymbolBitmapName =SymbolBitmapName;
fea.Style.SymbolBitmapTransparent =SymbolBitmapTransparent;
fea.Style.SymbolBitmapOverrideColor =SymbolBitmapOverrideColor;
fea.Style.SymbolBitmapColor =SymbolBitmapColor;
fea.Update(true, rvs);
}
}
catch
{}
}

```

```

public void DeleteAllfeatures(string layerName)

```

```

//删除用户工作层 userLayer 中所有图元

```

```

{
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        if (layer==null) return;
    }
    foreach(MapXLib.Feature feature in axMap1.Layers._Item(layerName).AllFeatures)
    {

```

```

        axMap1.Layers._Item(layerName).DeleteFeature(feature);
    }
    axMap1.Layers._Item(layerName).Pack(MapXLib.LayerPackConstant.miPackAll);
}

private void axMap1_SelectionChanged(object sender, System.EventArgs e)
//用于生成 viewlayer 图层图元 InfoTip
{
    string layerName="userLayer";
    MapXLib.Layer layer=axMap1.Layers._Item(layerName+"FromDB");
    MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName+"FromDB");
    foreach(MapXLib.Feature ftr in layer.Selection)
    {
        selectedFeatureKey=ftr.FeatureKey;
        string msg="";
        for(int j=1; j<=ds.Fields.Count; j++)
        {
            layer.KeyField =ds.Fields._Item(j).Name;
            msg+=ftr.KeyValue.ToString();
        }
        toolTip1.SetToolTip(this.axMap1, msg);
    }
}

public void GetSymbolPropertyOnUserLayer(string layerName, ref FeatureInfo featureInfo)
//获取所有选中的图元的属性
{
    MapXLib.Layer layer=null;
    MapXLib.Dataset ds=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
        ds=axMap1.DataSets._Item("ds"+layerName);
    }
    catch
    {
        if (layer==null) return;
    }
    foreach(MapXLib.Feature ftr in layer.AllFeatures )
    {
        featureInfo.featureKey =ftr.FeatureKey ;
        featureInfo.point_x =ftr.CenterX ;
        featureInfo.point_y =ftr.CenterY ;
        featureInfo.symbolFileName =currentFeatureInfo.symbolFileName;
        featureInfo.source =rif.netStatu.localNetIndex.ToString () ;
    }
}

```



```

        featureInfo.name = "点击操作点";
        featureInfo.identity = currentFeatureInfo.identity ;
        featureInfo.foundTime = DateTime.Now.ToString();
    }
}

```

```
public void GetSelectedFeatureProperty(string layerName, ref FeatureInfo []featureInfo)
```

```
//获取和数据库关联的图层的所有选中图元的属性(数据库相关)
```

```

{
    MapXLib.Layer layer=null;
    MapXLib.Dataset ds=null;
    MapXLib.Fields fields=null;
    MapXLib.RowValues rvs;
    MapXLib.RowValue rv;
    try
    {
        layer=axMap1.Layers._Item(layerName);
        ds=axMap1.DataSets._Item("ds"+layerName);
        fields=ds.Fields;
    }
    catch
    {
        if (layer==null) return;
    }
    try
    {
        int count=layer.Selection.Count ;
        if(count==0)
            featureInfo =null;
        else
            featureInfo =new FeatureInfo [count];
        int i=0;
        foreach(MapXLib.Feature ftr in layer.Selection)
        {
            featureInfo [i].featureKey =ftr.FeatureKey ;
            featureInfo [i].point_x =ftr.CenterX ;
            featureInfo [i].point_y =ftr.CenterY ;
            featureInfo [i].symbolFileName =ftr.Style.SymbolBitmapName;
            layer.KeyField=fields._Item(1).Name;
            featureInfo [i].source =ftr.KeyValue.ToString();
            layer.KeyField=fields._Item(2).Name;
            featureInfo [i].name =ftr.KeyValue.ToString();
            layer.KeyField=fields._Item(3).Name;
            featureInfo [i].identity =ftr.KeyValue.ToString();
        }
    }
}

```

```

        layer.KeyField=fields._Item(4).Name;
        featureInfo [i].description  =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(5).Name;
        featureInfo [i].foundTime  =ftr.KeyValue.ToString();
        rvs=ds.get_RowValues(ftr);
        rv=rvs._Item("MI_RENDITION");
        string symbol=rv.Value.ToString();
        int startindex=symbol.IndexOf("\"")+1;
        int lastindex=symbol.LastIndexOf("\"");
        int len=lastindex - startindex;
        string bitmapfile=symbol.Substring(startindex, len);
        featureInfo[i].symbolFileName  =bitmapfile;
        i++;
    }
}
catch {}
}

```

```

public void GetAllFeatureProperty(string layerName, ref FeatureInfo []featureInfo)

```

//获取和数据库关联的图层的所有图元的属性（数据库相关）

```

{
    MapXLib.Layer layer=null;
    MapXLib.Dataset ds=null;
    MapXLib.Fields fields=null;
    MapXLib.RowValues rvs;
    MapXLib.RowValue rv;
    try
    {
        layer=axMap1.Layers._Item(layerName);
        ds=axMap1.DataSets._Item("ds"+layerName);
        fields=ds.Fields;
    }
    catch
    {
        if (layer==null) return;
    }
    try
    {
        int count=layer.AllFeatures.Count ;
        featureInfo =new FeatureInfo [count];
        int i=0;
        foreach(MapXLib.Feature ftr in layer.AllFeatures )
        {
            featureInfo [i].featureKey =ftr.FeatureKey ;

```

```

        featureInfo [i].point_x =ftr.CenterX ;
        featureInfo [i].point_y =ftr.CenterY ;
        featureInfo [i].symbolFileName  =ftr.Style.SymbolBitmapName;
        layer.KeyField=fields._Item(1).Name;
        featureInfo [i].source =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(2).Name;
        featureInfo [i].name =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(3).Name;
        featureInfo [i].identity =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(4).Name;
        featureInfo [i].description  =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(5).Name;
        featureInfo [i].foundTime  =ftr.KeyValue.ToString();
        rvs=ds.get_RowValues(ftr);
        rv=rvs._Item("MI_RENDITION");
        string symbol=rv.Value.ToString();
        int startindex=symbol.IndexOf("\"")+1;
        int lastindex=symbol.LastIndexOf("\"");
        int len=lastindex - startindex;
        string bitmapfile=symbol.Substring(startindex , len);
        featureInfo[i].symbolFileName  =bitmapfile;
        i++;
    }
}
catch {}
}

```

```

public void GetFeaturePropertyByFeatureKey(string layerName, ref FeatureInfo featureInfo, string featureKey)

```

```

//获取确定 featurekey 的图元属性（数据库相关）

```

```

{
    MapXLib.Layer layer=null;
    MapXLib.Dataset ds=null;
    MapXLib.Fields fields=null;
    MapXLib.RowValues rvs;
    MapXLib.RowValue rv;
    try
    {
        layer=axMap1.Layers._Item(layerName);
        ds=axMap1.DataSets._Item("ds"+layerName);
        fields=ds.Fields;
    }
    catch
    {
        if (layer==null) return;
    }
}

```

```

    }
    try
    {
        MapXLib.Feature ftr =layer.GetFeatureByKey (featureKey);
        featureInfo.featureKey =ftr.FeatureKey ;
        featureInfo.point_x =ftr.CenterX ;
        featureInfo.point_y =ftr.CenterY ;
        layer.KeyField=fields._Item(1).Name;
        featureInfo.source =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(2).Name;
        featureInfo.name =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(3).Name;
        featureInfo.identity =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(4).Name;
        featureInfo.description =ftr.KeyValue.ToString();
        layer.KeyField=fields._Item(5).Name;
        featureInfo.foundTime =ftr.KeyValue.ToString();
        rvs=ds.get_RowValues(ftr);
        rv=rvs._Item("MI_RENDITION");
        string symbol=rv.Value.ToString();
        int startindex=symbol.IndexOf("\")+1;
        int lastindex=symbol.LastIndexOf("\");
        int len=lastindex - startindex;
        string bitmapfile=symbol.Substring(startindex, len);
        featureInfo.symbolFileName =bitmapfile;
    }
    catch {}
}

public void DeleteUserSymbol(string layerName, string selectedFeatureKey)
//删除 viewlayer 层图元, layerName 为 viewLayer 图层的 name
{
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        return;
    }
    MapXLib.Feature selectedFeature=layer.GetFeatureByKey(selectedFeatureKey);
    if (selectedFeature==null) return;
    MapXLib.Dataset ds=axMap1.DataSets._Item("ds"+layerName);

```

```

MapXLib.RowValues rowValues=ds.get_RowValues(selectedFeature);
MapXLib.RowValue rowValue=rowValues._Item("MI_PRINX");
string prinx=rowValue.Value.ToString();
string str="delete from system."+layerName.Substring(0, layerName.Length-6);
str+=" where MI_PRINX="+prinx;
OracleConnection myConn = new OracleConnection(connectString);
OracleCommand myOracleCommand = new OracleCommand(str);
myOracleCommand.Connection = myConn;
myConn.Open();
myOracleCommand.ExecuteNonQuery();
myConn.Close();
}
public void DeleteDBLayerSelectedFeature()
//删除数据库关联的所有选中的图元
{
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(userLayerName+"FromDB");
    }
    catch
    {
        return;
    }
    try
    {
        foreach(MapXLib.Feature selectedFeature in layer.Selection )
        {
            DeleteUserSymbol(userLayerName+"FromDB", selectedFeature .FeatureKey );
        }
    }
    catch {}
}
public void DeleteDBLayerAllFeature()
//删除数据库关联的所有图元
{
    MapXLib.Layer layer=null;
    try
    {
        layer=axMap1.Layers._Item(userLayerName+"FromDB");
    }
    catch
    {
        return;
    }
}

```

```

foreach(MapXLib.Feature selectedFeature in layer.AllFeatures)
{
    DeleteUserSymbol(userLayerName+"FromDB", selectedFeature.FeatureKey );
}
OracleToLayer (userLayerName);
}
public void AddSymbolTool(string layerName, string bitmapFileName, string Identity)
//用 mouse 增加图元至工作图层上
{
    axMap1.AutoRedraw =false;
    MapXLib.Layer layer=null;
    currentFeatureInfo.identity =Identity;
    currentFeatureInfo.symbolFileName =bitmapFileName;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        if (layer==null) return;
    }
    MapXLib.Style style=new MapXLib.StyleClass();
    style.SymbolType =MapXLib.SymbolTypeConstants.miSymbolTypeBitmap;
    style.SymbolBitmapSize=20;
    style.SymbolBitmapName=bitmapFileName;
    style.SymbolBitmapTransparent=true;
    layer.OverrideStyle =true;
    layer.Style =style ;
    layer.Editable=true;
    axMap1.Layers.InsertionLayer=layer;
    axMap1.CurrentTool=MapXLib.ToolConstants.miAddPointTool;
}
private void menuItemChangeGST_Click(object sender, System.EventArgs e)
{
    string geoset1="";
    bool flag=OpenGeoset(ref geoset1);
    if(flag)
    {
        geoset=geoset1;
        rif.mapInfoSet.GeoSet =geoset1;
        changeMapDelegate();
        mapSmall.GeoSet =geoset1.Substring (0, geoset1.Length -4)+"eye.gst";
        mapSmall.TitleText ="";
        SetEyeMap(mapSmall);
    }
}

```

```

    }
}
private void menuItemSaveGST_Click(object sender, System.EventArgs e)
{
    SaveGeoset (Geoset );
}
private void menuItemLayerCTL_Click(object sender, System.EventArgs e)
{
    LayerCtrl ();
}
private void axMap1_MouseDownEvent(object sender,
    AxMapXLib.CMapXEvents_MouseDownEvent e)
{
    if(e.button ==2)
    {
        System.Drawing .Point pt=new System.Drawing .Point ((int)e.x , (int)e.y);
        mapMenu.Show (this, pt);
    }
}
private void axMap1_MouseUpEvent(object sender,AxMapXLib.CMapXEvents_MouseUpEvent e)
{
    if(e.button ==1 && axMap1 .CurrentTool ==MapXLib.ToolConstants.miAddPointTool)
    {
        FeatureInfo featureInfo=new FeatureInfo ();
        GetSymbolPropertyOnUserLayer(userLayerName , ref featureInfo);
        AddUserSymbol(ref featureInfo , userLayerName);
        OracleToLayer (userLayerName );
        addSymbolDelegate();
    }
}
//设置数据库关联层的可选状态，其他都不可选
public void SetDBLayerSelectable()
{
    foreach(MapXLib.Layer layer1 in axMap1 .Layers )
    {
        layer1.Selectable =false;
    }
    MapXLib.Layer layer=axMap1 .Layers ._Item (userLayerName+"FromDB");
    layer.Selectable =true;
}
//获取所有选中的图元
private void GetSelectedFeatureKey(string layerName, ref string []featureKey)
{
    MapXLib.Layer layer=null;
    try
    {

```

```

        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        return;
    }
    int count=layer.Selection.Count;
    featureKey=new string[layer.Selection.Count];
    int i=0;
    foreach(MapXLib.Feature selectedFeature in layer.Selection )
    {
        featureKey[i]=selectedFeature.FeatureKey;
        i++;
    }
}
//设置图元的选中状态
private void SetSelectedFeatureKey(string layerName, string[] featureKey)
{
    MapXLib.Layer layer=null;
    MapXLib.Feature feature=null;
    try
    {
        layer=axMap1.Layers._Item(layerName);
    }
    catch
    {
        return;
    }
    for(int i=0; i<featureKey.Length; i++)
    {
        feature=layer.GetFeatureByKey(featureKey[i]);
        int id=feature.FeatureID;
        layer.Selection.AddByID(id);
    }
}
public void SetEyeMap(AxMapXLib.AxMap mapEye)
{
    try
    {
        m_Layer=mapSmall.Layers.CreateLayer("RectLayer", @"d: \mapSmall", 1,
            32, mapSmall.NumericCoordSys);
        mapSmall = mapEye;
        //初始时, 产生鹰眼图矩形框绘画层
        if (!File.Exists(Application.StartupPath + "\\mapSmall.tab"))
            m_Layer=mapSmall.Layers.CreateLayer("RectLayer", Application.StartupPath+

```



```

        "\\mapSmall.tab", 1, 32, mapSmall.NumericCoordSys);
    else
        m_Layer = mapSmall.Layers.Add(Application.StartupPath+"\\mapSmall.tab", 1);
        m_Layer.Editable=true;
    }
    catch {}
}
private void axMap1_MapViewChanged(object sender, System.EventArgs e)
{
    if(mapSmall==null) return;
    mapSmall.AutoRedraw = false;
    MapXLib.Feature tempfea=new MapXLib.FeatureClass();
    MapXLib.Style tempsty=new MapXLib.StyleClass();
    MapXLib.FeatureFactory feaFact;
    MapXLib.Feature m_fea=new MapXLib.FeatureClass();
    feaFact=mapSmall.FeatureFactory;
    tempsty.RegionPattern=MapXLib.FillPatternConstants.miPatternNoFill;
    tempsty.RegionBorderColor=255;
    tempsty.RegionBorderWidth=2;
    MapXLib.Features ftrs;
    ftrs=m_Layer.AllFeatures;
    MapXLib.PointsClass ps=new MapXLib.PointsClass ();
    MapXLib.PointClass p=new MapXLib.PointClass ();
    if(ftrs.Count==0)//矩形边框还没有
    {
        float a=0 ;
        float b=(float)axMap1.Width ;
        float c=0 ;
        float d=(float)axMap1.Height ;
        double x=0, y=0;
        axMap1.ConvertCoord(refa, refc , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
        p.Set (x, y) ;
        ps.Add (p, 1);
        axMap1.ConvertCoord(refb, refc , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
        p.Set (x, y) ;
        ps.Add (p, 2);
        axMap1.ConvertCoord(refb, refd , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
        p.Set (x, y) ;
        ps.Add (p, 3);
        axMap1.ConvertCoord(refa, refd , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
        p.Set (x, y) ;
        ps.Add (p, 4);
        tempfea = feaFact.CreateRegion(ps, tempsty);
    }
}

```

```

m_fea=m_Layer.AddFeature(tempfea, new MapXLib.RowValuesClass());
}
else
{
m_fea=ftrs._Item(1);
m_fea.Parts._Item(1).RemoveAll();
float a=0 ;
float b=(float)axMap1.Width ;
float c=0 ;
float d=(float)axMap1.Height ;
double x=0, y=0;
axMap1.ConvertCoord(refa, refc , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
m_fea.Parts._Item(1).AddXY (x, y, 1);
axMap1.ConvertCoord(refb, refc , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
m_fea.Parts._Item(1).AddXY (x, y, 2);
axMap1.ConvertCoord(refb, refd , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
m_fea.Parts._Item(1).AddXY (x, y, 3);
axMap1.ConvertCoord(refa, refd , refx, refy, MapXLib.ConversionConstants .miScreenToMap);
m_fea.Parts._Item(1).AddXY (x, y, 4);
m_fea.Update (true, new MapXLib.RowValuesClass());
}
}
}
}

```

第七章 分发基于.net 平台的 MapX 应用程序

7.1 .NET Framework 概述

.NET Framework 是一种新的计算平台，它简化了在高度分布式 Internet 环境中的应用程序开发。.NET Framework 旨在实现下列目标：

提供一个一致的面向对象的编程环境，而无论对象代码是在本地存储和执行，还是在本地执行但在 Internet 上分布，或者是在远程执行的。

提供一个将软件部署和版本控制冲突最小化的代码执行环境。

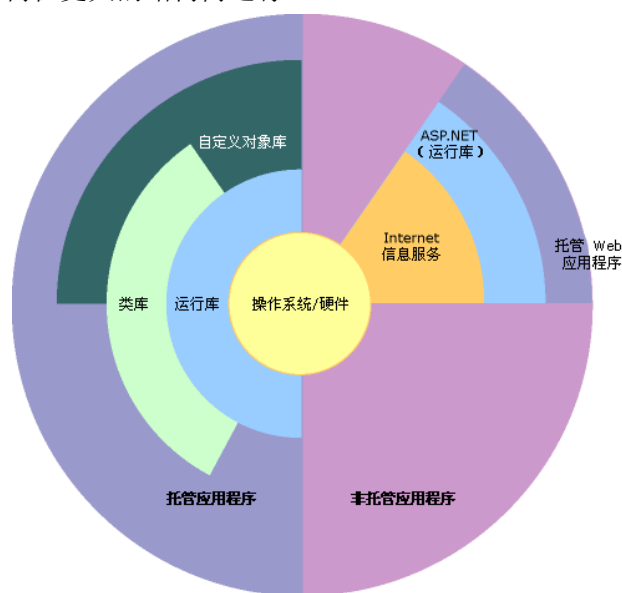
提供一个保证代码（包括由未知的或不完全受信任的第三方创建的代码）安全执行的代码执行环境。

提供一个可消除脚本环境或解释环境的性能问题的代码执行环境。使开发人员的经验在面对类型大不相同的应用程序（如基于 Windows 的应用程序和基于 Web 的应用程序）时保持一致。按照工业标准生成所有通信，以确保基于 .NET Framework 的代码可与任何其他代码集成。

.NET Framework 具有两个主要组件：公共语言运行库和 .NET Framework 类库。公共语言运行库是 .NET Framework 的基础。您可以将运行库看作一个在执行时管理代码的代理，它提供核心服务（如内存管理、线程管理和远程处理），而且还强制实施严格的类型安全以及可确保安全性和可靠性的其他形式的代码准确性。事实上，代码管理的概念是运行库的基本原则。以运行库为目标的代码称为托管代码，而不以运行库为目标的代码称为非托管代码。.NET Framework 的另一个主要组件是类库，它是一个综合性的面向对象的可重用类型集合，您可以使用它开发多种应用程序，这些应用程序包括传统的命令行或图形用户界面（GUI）应用程序，也包括基于 ASP.NET 所提供的最新创新的应用程序（如 Web 窗体和 XML Web services）。

.NET Framework 可由非托管组件承载，这些组件将公共语言运行库加载到它们的进程中并启动托管代码的执行，从而创建一个可以同时利用托管和非托管功能的软件环境。.NET Framework 不但提供若干个运行库宿主，而且还支持第三方运行库宿主的开发。

下面的插图显示公共语言运行库和类库与应用程序之间以及与整个系统之间的关系。该插图还显示托管代码如何在更大的结构内运行。



7.2 .NET Framework 的主要组件和功能

7.2.1 公共语言运行库

公共语言运行库管理内存、线程执行、代码执行、代码安全验证、编译以及其他系统服务。这些功能是在公共语言运行库上运行的托管代码所固有的。至于安全性，取决于包括托管组件的来源（如 Internet、企业网络或本地计算机）在内的一些因素，托管组件被赋予不同程度的信任。这意味着即使用在同一活动应用程序中，托管组件既可能能够执行文件访问操作、注册表访问操作或其他须小心使用的功能，也可能不能够执行这些功能。

运行库强制实施代码访问安全。例如，用户可以相信嵌入在 Web 页中的可执行文件能够在屏幕上播放动画或唱歌，但不能访问他们的个人数据、文件系统或网络。这样，运行库的安全性功能就使通过 Internet 部署的合法软件能够具有特别丰富的功能。

运行库还通过实现称为通用类型系统（CTS）的严格类型验证和代码验证基础结构来加强代码可靠性。CTS 确保所有托管代码都是可以自我描述的。各种 Microsoft 和第三方语言编译器生成符合 CTS 的托管代码。这意味着托管代码可在严格实施类型保真和类型安全的同时使用其他托管类型和实例。

此外，运行库的托管环境还消除了许多常见的软件问题。例如，运行库自动处理对象布局并管理对对象的引用，在不再使用它们时将它们释放。这种自动内存管理解决了两个最常见的应用程序错误：内存泄漏和无效内存引用。

运行库还提高了开发人员的工作效率。例如，程序员可以用他们选择的开发语言编写应用程序，却仍能充分利用其他开发人员用其他语言编写的运行库、类库和组件。任何选择以运行库为目标的编译器供应商都可以这样做。以 .NET Framework 为目标的语言编译器使得用该语言编写的现有代码可以使用 .NET Framework 的功能，这大大减轻了现有应用程序的迁移过程的工作负担。

尽管运行库是为未来的软件设计的，但是它也支持现在和以前的软件。托管和非托管代码之间的互操作性使开发人员能够继续使用所需的 COM 组件和 DLL。

运行库旨在增强性能。尽管公共语言运行库提供许多标准运行库服务，但是它从不解释托管代码。一种称为实时（JIT）编译的功能使所有托管代码能够以它在其上执行的系统的本机语言运行。同时，内存管理器排除了出现零碎内存的可能性，并增大了内存引用区域以进一步提高性能。

最后，运行库可由高性能的服务器端应用程序（如 Microsoft® SQL Server™ 和 Internet 信息服务（IIS））承载。此基础结构使您在享受支持运行库宿主的行业最佳企业服务器的优越性能的同时，能够使用托管代码编写业务逻辑。

7.2.2 .NET Framework 类库

.NET Framework 类库是一个与公共语言运行库紧密集成的可重用的类型集合。该类库是面向对象的，并提供您自己的托管代码可从中导出功能的类型。这不但使 .NET Framework 类型易于使用，而且还减少了学习 .NET Framework 的新功能所需要的时间。此外，第三方组件可与 .NET Framework 中的类无缝集成。例如，.NET Framework 集合类实现一组可用于开发您自己的集合类的接口。您的集合类将与 .NET Framework 中的类无缝地混合。

正如您对面向对象的类库所希望的那样，.NET Framework 类型使您能够完成一系列常见编程任务（包括诸如字符串管理、数据收集、数据库连接以及文件访问等任务）。除这些常见任务之外，类库还包括支持多种专用开发方案的类型。例如，可使用 .NET Framework 开发下列类型的应用程序和服务：

控制台应用程序

- . Windows GUI 应用程序 (Windows 窗体)
- . ASP.NET 应用程序
- . XML Web services
- . Windows 服务

例如, Windows 窗体类是一组综合性的可重用的类型, 它们大大简化了 Windows GUI 的开发。如果要编写 ASP.NET Web 窗体应用程序, 可使用 Web 窗体类。

7.3 安装 .NET Framework

为 .NET Framework 编写的应用程序和控件要求 .NET Framework 安装在运行它们的计算机上。Microsoft 提供了可再发行的安装程序 (Dotnetfx.exe), 其中包含运行 .NET Framework 应用程序所必需的公共语言运行库和 .NET Framework 组件。可以从 Microsoft MSDN 下载中心或 Microsoft windows update web 站点下载 Dotnetfx.exe 的最新版本。如果您以前安装了 .NET Framework SDK 或 Microsoft Visual Studio .NET, 则不需要安装 Dotnetfx.exe。

尤其要注意的是, 您无法在运行 Microsoft Windows 95 操作系统的计算机上安装 .NET Framework。如果达不到最低配置要求, Dotnetfx.exe 安装程序将停止安装可再发行组件包。

7.4 Map 客户安装

要使 MapX 应用程序能够在客户机器上运行, 必须安装 MapX 支撑文件, 通常有以下两种方法在客户端安装:

(1) 安装程序自动安装的文件

运行MapX控件安装程序将在默认的 MapX 安装目录中安装以下内容:

C: \Program Files\MapInfo\MapX 5.0

- MapX 控件 mapx50.dll
- 它的支持的 dll
- 光栅和网格 dll 以及处理程序
- 默认的数据集驱动程序

这是最简单的分发方法, MapX 开发人员只需进行非常少的工作, 因为某些过程和任务已自动化。

(2) 在用户端注册必须的文件

将MapX DLL 及其相关 DLL数据集驱动程序(用于数据绑定)及其它所有支撑文件拷贝到用户机器上然后运行 regsvr32.exe 实用程序进行注册。

7.5 制作安装程序

创建windows 安装项目, 建立安装程序, 请参照MSDN中windows应用程序安装部署一节。