第8章

栅格数据

8.1 简介

栅格数据是由一系列的规则格网单元组成，用于表达专题、光谱以及图像等数据。它由多个等间距矩形栅格组成，用于模拟在一定的空间范围内连续变化的地理现象或者图片数据。这种格式常见，如卫星图片、扫描图和航片等，能表达各种地表类型，如高程、植被等。ArcEngine 中的栅格子模块主要包含访问和处理栅格的对象，如 Rasters、Rasterdatasets 和 Raster catalogs 等。这些对象不仅可以访问基于文件的栅格数据，也能访问存储在 GeoDatabase 中的栅格数据。

Raster 数据分为两种类型，一种是专题数据，可用于进行地理分析；另一种是影像数据，一般用于地图的背景显示。ArcGIS 所支持的栅格数据有：GRID、TIFF、ERDAS IMAGE、JPEG 等。在 ArcEngine 中，提供了一种统一的方式来管理任何类型的栅格数据。

栅格数据模型用连续空间的像元来代表要素，像元表现为一个矩形的区域，宽度和高度相等，其大小取决于原始地图的比例尺和绘制地图时最小的单位。栅格是一个二维的动态像元矩阵，每个栅格单元都有一个值，用于表示某个位置的某种属性，如高程、反射率、颜色等。

栅格数据集是存储在文件或数据库中的以某种栅格格式存在的数据对象，通过使用栅格图层将数据加载到地图控件中。

8.2 访问和创建栅格数据

对于栅格数据而言，基于文件的工作空间、基于 ACCESS 的个人数据、基于企业数据库的工作空间等都是其工作空间，在访问栅格数据时，必须打开一个工作空间。工作空间必须通过工作空间工厂来创建，可以通过 RasterWorkspaceFactory 对象来创建。

RasterWorkspaceFactory 对象通常用于生成一个 RasterWorkspace 对象，RasterWorkspace 对象就是一个栅格数据集容器，RasterWorkspaceFactory 类实现了 IWorkspaceFactory 和 IWorkspaceFactory2 接口。RasterWorkspace 对象实现了 IrasterWorkspace 和 IRasterWorkspace2 接口，这两个接口定义了用于打开和设置一个数据集对象，如：

- Copy 方法用于复制一个工作空间中所有的栅格数据集到一个新的或已经存在的工作空间中去；
- OpenRasterDataset 方法用于打开一个栅格数据集；
- CreateRasterDataset 方法用于产生一个新的基于文件的栅格数据集对象，这个数据格式可以为 GRID、TIFF 和 ERDAS Image 之一。

下面的代码片段演示了如何使用这些接口加载栅格数据。

```
IWorkspaceFactory workspaceFactory = new RasterWorkspaceFactoryClass();
IWorkspace workspace;
string filePath = @"E:\vs2005\第 8 章\lesson1\data";
IRasterWorkspace2 rasterWorkspace;
IRasterDataset rasterDataset;
rasterWorkspace = workspaceFactory.OpenFromFile(filePath, 0) as IRasterWorkspace2 ;
rasterDataset = rasterWorkspace.OpenRasterDataset( @"\NewTif");
IRasterLayer rasterLayer = new RasterLayerClass();
rasterLayer.CreateFromDataset(rasterDataset);
this.axMapControl1.Map.AddLayer(rasterLayer as ILayer);
this.axMapControl1.Extent = this.axMapControl1.FullExtent;
```

8.3 栅格数据配准

地理信息系统与遥感技术在应用中多少要涉及栅格数字图像的校正配准处理。例如 TM 影像图与数字高程模型 DEM 图的配准，配准后的 TM 图再用于地面上各类资源的空间分析研究与实际的生产应用。再如，经扫描输入微机的 Tiff 栅格图，由于存在人工操作及仪器误差，其图像总会有变形，因此必要时就得经过几何校正才能投入到应用中去。由此，数字图像的配准与校正技术在 GIS 及 RS 的应用研究中就显得尤为必要。

栅格数据的配准主要是使用 IGeoReference 接口来实现。通过 ActiveView 进行坐标转换，将 MapControl 中点的位置转换为地图和影像图上的坐标。该接口被 RasterLayer 类实现，该接口提供有栅格数据配准 11 个方法和 1 个属性，如：

- CanGeoRef 属性用于监测该图层是否可以做配准；
- PointTransform 方法用于将鼠标的位置转换为栅格文件上的相对坐标；
- Rectify 方法用于将纠正的结果保存为一个新的栅格文件；
- Register 方法用于将纠正的结果生成 Word 文件，和栅格文件保存在同一个目录下；
- Reset 方法用于取消纠正，但是对 Register 之前的操作不能恢复；
- Shift 方法可提供一点纠正，即平移；
- TwoPointsAdjust 方法可提供两点配准，有一定的缩放；
- Warp 方法可提供 3 点或以上的配准。

8.4 栅格数据处理

栅格数据是 GIS 最基本的数据之一。早期的计算机系统由于存储容量小、运算速度慢，因此栅格数据的实用性受到了极大的限制。在现在的计算机环境下，计算机的运算能力、寻址能力、外部设备的性能大为改善，栅格数据的优势也逐渐显现出来。随着 GIS 技术应用的普及与深入，出现了大量以遥感信息为主的数据源系统，如环境监测、火灾跟踪等，这些遥感数据大多为栅格

数据形式的影像数据和数据流。本节介绍栅格数据处理的一些功能。

8.4.1 栅格数据转换

栅格数据转换是指将用矢量形式表示的空间数据表示为栅格格式，即将 GeoDatabase 中的 FeatureClass 或是基于文件的矢量数据栅格化（ShapeFile、Coverage 等）。在 ArcEngine 的 GeoAnalyst 类库中提供有 IConversionOp 接口，该接口提供有 5 个方法：RasterDataToLineFeatureData、RasterDataToPointFeatureData、RasterDataToPolygonFeatureData、ToFeatureData、ToRasterDataset。其中前 4 个方法用于将栅格数据转换为矢量数据，实现从栅格数据图层中提取点、线、面矢量要素，可用于提取矢量要素的栅格对象有 Raster、RasterDataset、RasterBand 和 RasterDescriptor 等。ToRasterDataset 方法用于实现矢量要素栅格化，可用于栅格化的矢量要素有 FeatureClass、FeatureClassDescriptior 和 Featurelayer 等。该接口为 RasterConversionOpClass 类所实现。

下面的代码片段演示了如何将栅格数据导出为点状矢量数据。

```
    string filePath=@"E:\vs2005\第 8 章\lesson1\data";
    IConversionOp  conversionOP =new RasterConversionOpClass ();
    IRasterWorkspace2 rasterWorkspace;
    RasterDataset rasterDataset ;
    rasterWorkspace = workspaceFactory.OpenFromFile(filePath, 0) as IRasterWorkspace2 ;
    rasterDataset = rasterWorkspace.OpenRasterDataset( @"\NewTif");
    IRasterLayer rasterLayer = new RasterLayerClass();
    rasterLayer.CreateFromDataset(rasterDataset);
    IRaster raster =rasterLayer.Raster ;

    IGeoDataset geoDataset ;
    IWorkspaceFactory workspaceFactory ;
    IWorkspace workspace ;
    workspace =workspaceFactory.OpenFromFile (@"e:\temp",0);
    string outFeatureClassName ="newShapeFile.shp";
    geoDataset  =conversionOP.RasterDataToPointFeatureData  (raster  ,workspace  ,
outFeatureClassName );
```

ToRasterDataset 方法用于将矢量数据转换为栅格数据，该方法生成的是 IRasterDataset 对象，方法的输入参数要求矢量数据为 FeatureClass 、FeatureClassDescriptor、FeatureLayer 类型的对象，并支持输出 3 种栅格类型：ESRI GRID 、ERDAS Imagine 和 TIFF。在方法的参数中用“GRID”代表 ESRI GRID 类型，生成没有扩展名的栅格文件；“TIFF”、“TIF”代表 TIFF 格式，生成扩展名为“.tiff”、“.tif”的栅格文件；“ERDAS Imagine”生成“.img”格式的文件。下面的代码片段演示了如何把矢量数据转换为栅格数据。

```
    ILayer layer = this.axMapControl1.get_Layer(0);
    IFeatureLayer featureLayer = layer as IFeatureLayer;
    //获取 FeatureClass
    IFeatureClass featureClass = featureLayer.FeatureClass;
    //获取转换的字段
    IFeatureClassDescriptor  featureClassDescriptor = new FeatureClassDescriptorClass ();
    //转换字段
```

```
    featureClassDescriptor.Create(featureClass, null, "code");
    //获取第 1 个参数
    IGeoDataset geoDataset = featureClassDescriptor as IGeoDataset;
    string outPutPath = @"e:\temp";
    IWorkspaceFactory workspaceFactory = new RasterWorkspaceFactoryClass();
    IWorkspace workspace = workspaceFactory.OpenFromFile(outPutPath, 0);
    IConversionOp conversionOp = new RasterConversionOpClass();
    IRasterAnalysisEnvironment rasterAnalysisEnvironment = conversionOp as IrasterAnalysis
Environment;
    //栅格的大小，通过宽度或者高度较小的值与 250 的比值
    double dCellSize = 0.024668;
    object oCellSize = dCellSize as object;
    //先设置栅格大小，再进行转换
    rasterAnalysisEnvironment.SetCellSize(esriRasterEnvSettingEnum.esriRasterEnvValue,    ref
oCellSize);
    //设置存储位置、转换字段以及格式
    IRasterDataset rasterDataset = conversionOp.ToRasterDataset(geoDataset, "GRID",
workspace, "NewRaster");
```

8.4.2　栅格数据变换

栅格数据变换是指将一个栅格数据层，通过某种算法进行变换，得到新的栅格数据图层。例如，将用栅格数据表示的植被、土壤分类数据简单地转换为上一级的分类数据，将用栅格数据表示的高程数据转换为坡度、坡向数据，将用栅格数据表示的农作物分布数据转换为某一类农作物的分布数据等。

8.4.3　栅格数据叠置分析

栅格数据叠置分析，是指将多个栅格数据层，以相同空间位置上的栅格单元为处理单元，通过某种算法进行变换，得到新的栅格数据层。例如，将表示地貌、高程、土壤、水分等一系列地表性质的数据，转换为土地等级或土地评价数据等。

栅格图层叠置分析用到的主要接口有 IMathOp、ILogicalOp、ITrigOp 等。

IMathOp 接口提供有进行栅格数学运算的方法，组件类 RasterMathOps 封装了该接口，接口提供有 21 个方法。如：

- Abs 方法用于计算栅格像元的绝对值，并写入新数据集；
- Divide 方法用于把两个输入栅格图层像元值相除，并写入新数据集；
- Exp 方法用于计算栅格像元以 e 为底的次方值，并写入新数据集；
- Exp10 方法用于计算栅格像元以 10 为底的次方值，并写入新数据集；
- Exp2 方法用于计算栅格像元以 2 为底的次方值，并写入新数据集；
- Times 方法用于将两个输入栅格图层像元值相乘，并写入数据集。

ILogicalOp 接口提供有进行栅格逻辑运算的功能，提供有 15 个方法。如：

- BooleanAnd 方法用于两个 Rasters 对象之间进行布尔“and”操作；
- BooleanOr 方法用于两个 Rasters 对象之间进行布尔“Or”操作；
- CombinatorialAnd 方法用于两个 Rasters 对象之间进行组合“and”操作。

ITrigOp 接口提供有进行三角函数运算的功能，提供有 13 个方法。如：

- ACos 方法用于计算反余弦；
- ASin 方法用于计算反正弦。

下面的代码片段演示了两个栅格图层的相乘运算。

```
    IMathOp  mathOp =new RasterMathOpsClass ();
    IGeoDataset resultDataset= new RasterDataset ();
    ILayer layer ;
    IRasterLayer rasterLayer1,rasterLayer2;
    layer =this.axMapControl1.get_Layer (0);
    rasterLayer1 =layer as IRasterLayer ;
    layer =this.axMapControl1.get_Layer (1);
    rasterLayer2 =layer as IRasterLayer ;
    resultDataset =mathOp.Times (rasterLayer1.Raster as IGeoDataset ,rasterLayer2.
Raster as IGeoDataset );
```

8.4.4 栅格数据与矢量数据叠加分析

栅格数据与矢量数据叠加分析，是指将一个栅格数据层通过与用矢量数据表示的面要素叠加，得到矢量数据层中地理对象的某一属性信息。例如，将一个表示农作物种植分布的栅格数据层与用矢量表示的行政数据层叠置，即可获得该区域范围内各行政单位的各类农作物种植面积。

这种叠置分析，往往是针对矢量数据的点数据和线数据进行，一般是通过对缓冲区分析等空间分析方法，生成点对象或线对象一定范围内的缓冲区多边形，然后用此多边形与栅格数据层进行叠置。

8.5 栅格图层渲染

专题图是依据要素的一个或多个属性而设置不同的符号，从而达到区分不同类型要素的目的。Arc Engine 提供有 RasterRender 抽象类，用于实现栅格数据的渲染。该类有 5 个子类，负责进行不同类型的着色运算，这些类都实现了 IRasterRender 接口，该接口定义了栅格图层符号化的公共属性和方法。通过 IRasterLayer.Renderer 属性获取的一个栅格图层的符号化对象模型如图 8-1 所示。

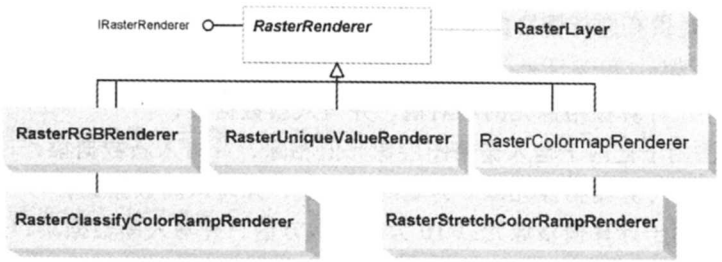


图 8-1

8.5.1 RasterRGBRenderer（栅格 RGB 符号化）

RasterRGBRenderer 通过绘制红、绿、蓝等 3 个波段的栅格数据集来显示，该组件类主要实

现 IRasterRenderer、IRasterRGBRenderer2。通过 IRasterRenderer 提供的属性 Raster 设置渲染的栅格数据集对象，通过 IRasterRGBRenderer2 提供的 3 个波段索引属性：RedBandIndex、GreenBandIndex、BlueBandIndex 进行渲染。下面的代码片段演示了栅格 RGB 如何符号化。

```
private void ChangeRGBRenderer (IRasterLayer rasterLayer)
{
   IRaster raster =rasterLayer.Raster ;
   IRasterBandCollection  rasterBandCollection ;
   rasterBandCollection=raster as IRasterBandCollection ;
    //如果波段小于 3 个，则不进行渲染
   if (rasterBandCollection.Count <3)
      return ;
   IRasterRGBRenderer2 rasterRGBRender=new RasterRGBRendererClass ();
   IRasterRenderer rasterRender =rasterRGBRender  as IRasterRenderer ;
   rasterRender.Raster =raster ;
   rasterRender.Update ();
   rasterRGBRender.RedBandIndex =2;
   rasterRGBRender.GreenBandIndex =1;
   rasterRGBRender.BlueBandIndex =0;
   rasterRender.Update ();
   rasterLayer.Renderer =rasterRGBRender ;
}
```

8.5.2 RasterUniqueValueRenderer（唯一值符号化）

唯一值符号化采用多种着色颜色，依据图层中的某个字段值给每个值一个单独的颜色，因此可以区分存在的每一个要素。该组件类实现 IRasterUniqueValueRenderer 接口，提供有 4 个方法和 11 个属性。如：

- AddValue 方法，用于将单个要素的某个字段值和与之相匹配的着色符号加入 RasterUniqueValueRenderer 对象；
- Symbol 属性用于设置每个值显示的符号；
- Field 属性用于设置取值字段；
- HeadingCount 属性用于设置渲染的标题数目；
- set_ClassCount 属性用于设置渲染的类别数目，即唯一值的个数。

下面的代码演示了如何使用该组件类实现唯一值符号化。

```
private void ChangeRender2UniqueValueRender(IRasterLayer rasterLayer)
{
    IRaster raster =rasterLayer.Raster ;
    ITable table;
    IRasterBand rasterBand;
    IRasterBandCollection rasterBandCollection;
    rasterBandCollection = raster as IRasterBandCollection;
    rasterBand = rasterBandCollection.Item(0);
    bool tablExist;
    rasterBand.HasTable(tablExist);
    if (!tablExist)
```

```
            return;
        table = rasterBand.AttributeTable;
        int numOfValues = table.RowCount(null);
        int fieldIndex;
        string fieldName="value";
        fieldIndex = table.FindField(fieldName);
        IRandomColorRamp randomColorRamp=new RandomColorRampClass ();
        randomColorRamp.Size = numOfValues;
        randomColorRamp.Seed = 100;
        randomColorRamp.CreateRamp(true);
        ISimpleFillSymbol simpleFillSymbol;
        IRasterUniqueValueRenderer rasterUniqueValueRender = new RasterUniqueValueRenderer
        Class();
        IRasterRenderer rasterRender = rasterUniqueValueRender as IRasterRenderer;
        rasterRender.Raster = raster;
        rasterRender.Update();
        rasterUniqueValueRender.HeadingCount = 1;
        rasterUniqueValueRender.set_Heading(0, "all data values");
        rasterUniqueValueRender.set_ClassCount(0, numOfValues);
        rasterUniqueValueRender.Field = fieldName;
        int i;
        IRow row;
        object labelValue;
        for (i = 0; i < numOfValues - 1;i++ )
        {
            row = table.GetRow(i);
            labelValue = row.get_Value(fieldIndex);
            rasterUniqueValueRender.AddValue(0, i, labelValue);
            rasterUniqueValueRender.set_Label(0, i, labelValue.ToString());
            simpleFillSymbol = new SimpleFillSymbolClass();
            simpleFillSymbol.Color = randomColorRamp.Colors.Next();
            rasterUniqueValueRender.set_Symbol(0, i, simpleFillSymbol);
        }
        rasterRender.Update();
        rasterLayer.Renderer = rasterUniqueValueRender;
    }
```

8.5.3 RasterClassfyColorRampRenderer（分类符号化）

分类符号化是将栅格图层划分为指定个数的类，每个类用一种颜色渲染。该类实现IRasterClassfyColorRampRenderer接口，提供有8个属性。如：

- ClassCount属性用于设置或获取分类的数目；
- ClassField属性用于获取或设置用于渲染的字段；
- Symbol属性用于设置或获取用于渲染的填充符号；
- Description属性用于获取或设置指定索引类的锚；
- Label属性用于获取或设置指定类的标签。

下面的代码片段演示了如何使用该组件类对象。

```
private void ChangeRender2ClassfyColorRampRenderer(IRasterLayer rasterLayer)
{
    IRaster raster = rasterLayer.Raster;
    //创建分类
    IRasterClassifyColorRampRenderer rasterClassifyColorRampRender = new
RasterClassify ColorRampRendererClass();
    IRasterRenderer rasterRender = rasterClassifyColorRampRender as IRasterRenderer;
    rasterClassifyColorRampRender.ClassCount = 20;
    IRgbColor fromColor = new RgbColorClass();
    IRgbColor toColor = new RgbColorClass();
    fromColor = getRGB(255, 0, 0);
    toColor = getRGB(0, 255, 0);
    IAlgorithmicColorRamp algorithmicColorRamp = new AlgorithmicColorRampClass();
    algorithmicColorRamp.Size = 20;
    algorithmicColorRamp.FromColor = fromColor;
    algorithmicColorRamp.ToColor = toColor;
    algorithmicColorRamp.CreateRamp(true);
    IFillSymbol fillSymbol = new SimpleFillSymbolClass();
    for (int i =0;i<rasterClassifyColorRampRender.ClassCount -1;i++)
    {
        fillSymbol.Color =algorithmicColorRamp.Colors.Next ();
        rasterClassifyColorRampRender.set_Symbol (i,fillSymbol as ISymbol );
        rasterClassifyColorRampRender.set_Label (i,"Class"+i.ToString (0));
    }
    rasterRender.Update ();
    rasterLayer.Renderer =rasterClassifyColorRampRender ;
}
```

8.5.4 RasterStretchColorRampRenderer

RasterStretchColorRampRenderer 组件类使用一个 AlgorithmicColorRamp 颜色带，来渲染栅格图层。该颜色带是通过指定起始颜色、终止颜色来确定一个颜色带，起始、终止颜色使用 HSV 模型。该组件类主要实现 IRasterStretchColorRampRenderer 接口，提供有 1 个方法和 6 个属性。如：

- BandIndex 属性用于设置栅格数据集的波段索引；
- ColorRamp 属性用于设置渲染的颜色带。

下面的代码片段演示了如何使用该组件类对象。

```
//获取颜色对象
private IRgbColor getRGB(int r, int g, int b)
{
    IRgbColor pColor;
    pColor = new RgbColorClass();
    pColor.Red = r;
    pColor.Green = g;
    pColor.Blue = b;
```

```
        return pColor;
    }
    private void ChangeRender2RaseterStretchColorRampRender(IRasterLayer rasterLayer)
    {
        IRaster raster =rasterLayer.Raster ;
        IRasterStretchColorRampRenderer rasterStretchColorRampRender = new RasterStretchColor
RampRendererClass();
        IRasterRenderer rasterRender = rasterStretchColorRampRender as IRasterRenderer;
        rasterRender.Raster = raster;
        rasterRender.Update();
        IColor fromColor = new RgbColorClass();
        fromColor = getRGB(255, 0, 0);
        IColor toColor = new RgbColorClass();
        toColor = getRGB(0, 255, 0);
        //创建起止颜色带
        IAlgorithmicColorRamp algorithmicColorRamp = new AlgorithmicColorRampClass();
        algorithmicColorRamp.Size = 255;
        algorithmicColorRamp.FromColor = fromColor;
        algorithmicColorRamp.ToColor = toColor;
        algorithmicColorRamp.CreateRamp(true);
        //选择拉伸颜色带符号化的波段
        rasterStretchColorRampRender.BandIndex = 0;
        //设置拉伸颜色带符号化所采用的颜色带
        rasterStretchColorRampRender.ColorRamp = algorithmicColorRamp;
        rasterRender.Update();
        rasterLayer.Renderer = rasterStretchColorRampRender ;
    }
```

8.5.5 RasterDiscreteColorRenderer（点密度符号化）

RasterDiscreteColorRenderer 组件类非常简单，主要实现 IRasterDiscreteColorRenderer 接口，只有两个属性，其中 Colormap 属性用于设置颜色带对象，NumColors 属性用于设置颜色的数目。

8.6 本章小结

栅格数据是在 GIS 中经常需要处理的另一大类地理数据，用于模拟连续分布特征的对象。ArcEngine 在数据处理中使用了一个统一的数据模型，因此栅格数据的操作方法与矢量数据有很大的相似性。本章简单介绍了栅格数据的创建与访问、栅格数据处理、栅格图层渲染等，并给出一些代码片段演示了各个组件类的使用。其中的 RasterColormapRenderer 渲染方法在“帮助”中介绍的不够详细，需要使用该渲染方法的读者，可以参考其他的书籍。