# 一、原理

在DotSpatial的标准库中，还没有现成的数据组件或方法，来直接将一个带金字塔的大影像加载成一个图层，并根据地图控件的缩放范围，自动调整适当的金字塔层级和范围，以最高效率显示大数据量影像。

所以，本文的目的在于实现这样一个数据组件，来高效地显示带金字塔的影像。

实现这一数据组件的原理在于：

1. 通过插件方式调用GDAL加载带金字塔的影像（使用插件方式调用GDAL的原理和方法参考另一篇笔记《在DotSpatial中调用GDAL》）；
2. 通过IRasterProvider和IRaster接口，将每次实时读取的内容传回主窗体，动态地生成一个InRamImageData，刷新图层显示内容。

# 二、方法流程

具体的方法流程为

1. 第一步先构建一个插件，使用ＧＤＡＬ来打开影像（参考附件中的PyramidRaster项目）；
2. 在地图控件中添加一个MapImageLayer图层，并针对地图控件的ViewExtentChanged事件，注册一个响应函数，在这个响应函数中，将当前地图控件的显示范围及分辨率参数传到第一步构建的插件中；
3. 在插件中，根据第传来的显示范围和分辨率等参数，计算合适的金字塔层号，并将范围内的影像值读取出来，生成一个Byte类型的数组（目前只支持Byte类型，如果需要后续还可以添加对其它类型影像的支持）；
4. 在主窗体中接收到数组，根据这个数组组合成一张Bitmap图片，并生成一个InRamImageData对象，更新到MapImageLayer图层的Image对象中，实现更新。

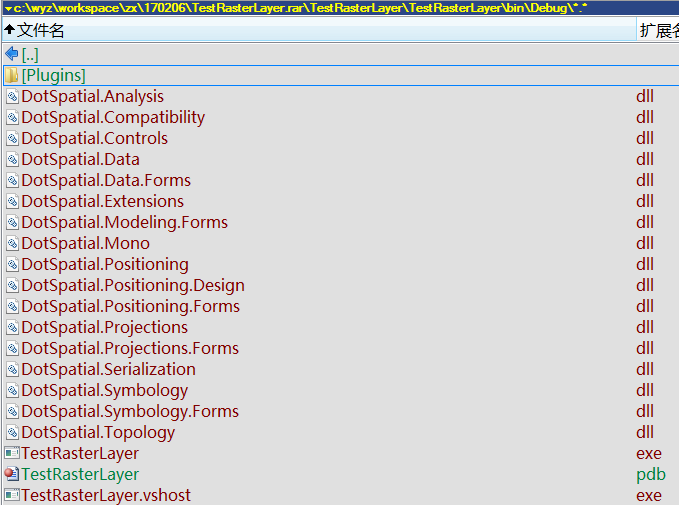
# 三、注意的地方

整个实现过程中，有几个需要注意的地方：

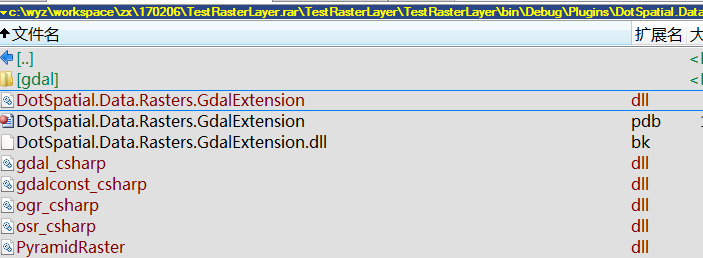
1. DotSpatial主库与插件之间的通信机制是通过DotSpatial.Data里IDataProvider及IRaster等接口实现的，限定得比较死，很多参数不是很方便传输，本文的例子中，采用了一些变通的方式来传递，例如将一些对象序列化为字符串，再使用字符串传递到接口中，接口中使用的时候再反序列化出来（参考附件中的MsgClass类）；
2. 在生成InRamImageData对象时，会出现一个问题：由于金字塔影像图层是根据地图范围，动态地从影像中截取一小部分，并将这一小部分生成一个InRamImageData，加载到地图控件中。但地图会自动更新其包含内容的总空间范围，当地图控件中只有一个金字塔影像图层，没有其它图层时，在地图上执行放大操作时咱们这个动态InRamImageData的范围就变小了，地图也会将其总空间范围变小，带来的问题就是，不能重新执行缩小操作了。所以需要在InRamImageData的基础上进行一个小的修改，本文的实例中，以InRamImageData为基类生成了一个新类MyInRamImageData。这个类的原理是，在InRamImageData的基础上，写一个子类，当地图控件读取图层空间范围时，总是把整个影像的空间范围返回给它，而不是只返回截取的一小部分，避免出现这个问题

# 四、附件说明

为减小附件体积，附件中的DLL文件都去掉了，Bin目录下的DLL文件如下图



Plugins下的文件如下图



# 五、纠错

附件代码中，PyramidRasterLayerClass.cs类，在图层移除时，未释放GDAL的Dataset，导致离线底图对应的栅格文件一直被占用，需要修改两个地方：

1. PyramidRasterLayerClass.cs类中，\_baseMapLayer\_RemoveItem方法中，加一行raster.Dispose();
2. PyramidRaster.cs类中，重载一下Dispose函数，在Dispose中将dataset释放掉

protected override void Dispose(bool disposeManagedResources)

{

base.Dispose(disposeManagedResources);

dataset.Dispose();

dataset = null;

}

1. PyramidRasterLayerClass的MapFrame\_ViewExtentsChanged需要添加一个对图层是否可见的判断

void MapFrame\_ViewExtentsChanged(object sender, ExtentArgs e)

{

if (\_bw.IsBusy != true && \_baseMapLayer.IsVisible==true)

{

\_bw.RunWorkerAsync();

}

}

4. PyramidRasterLayerClass的\_bw\_DoWork函数中，在将三波段合成彩色图像时，之前的方法在RBG波段顺序上有点问题，Bitmap里的波段顺序应该是BGR，正确的代码如下：

rgbValues[j\*4] = blueBytes[i][j];

rgbValues[j\*4 +1 ] = greenBytes[i][j];

rgbValues[j\*4+2] = redBytes[i][j];

rgbValues[j\*4+3] = alpha;

5. 为保证图层在移除时，对应的影像能够正确地被释放（在Legend中的右键菜单中移除图层，会正确地触发RemoveItem事件，但在代码里直接使用Map.Layers.Remove()函数，则不会触发），在PyramidRasterLayerClass中除了需要绑定ItemRemove事件之外

\_baseMapLayer.RemoveItem += new EventHandler(\_baseMapLayer\_RemoveItem);

最好还绑定一下Dispose事件

\_baseMapLayer.Disposed += new EventHandler(\_baseMapLayer\_Disposed);

在两个事件中，都加入一下影像释放代码：

void \_baseMapLayer\_RemoveItem(object sender, EventArgs e)

{

if (!raster.IsDisposed)

raster.Dispose();

\_baseMapLayer = null;

//\_serviceDropDown.SelectedItem = \_emptyProvider;

map.MapFrame.ViewExtentsChanged -= MapFrame\_ViewExtentsChanged;

}

void \_baseMapLayer\_Disposed(object sender, EventArgs e)

{

if (!raster.IsDisposed)

raster.Dispose();

\_baseMapLayer = null;

//\_serviceDropDown.SelectedItem = \_emptyProvider;

map.MapFrame.ViewExtentsChanged -= MapFrame\_ViewExtentsChanged;

}

然后，在使用Map.Layers.Remove（）函数时，在后面加一个Layer.Dispose（），像这样

IMapLayer tmpLyr = map1.Layers[i];

map1.Layers.Remove(tmpLyr);

tmpLyr.Dispose();