# 一、原理

在DotSpatial中，GDAL组件是采用插件形式调用的，在DotSpatial.Data.Rasters.GdalExtension这个插件中，引用了GDAL相关DLL，并进行了环境声明，从而可以在其中调用GDAL的功能函数。DotSpatial主库中，并不直接引用GDAL，而是在LoadPlugins的过程中，加载DotSpatial.Data.Rasters.GdalExtension.dll这个插件，并通过预定义的接口，调用其中功能或对象（这些功能和对象并不是直接调用的GDAL函数，而是通过预定义的接口，操作DotSpatial.Data.Rasters.GdalExtension.dll插件内部的函数，间接调用GDAL）。

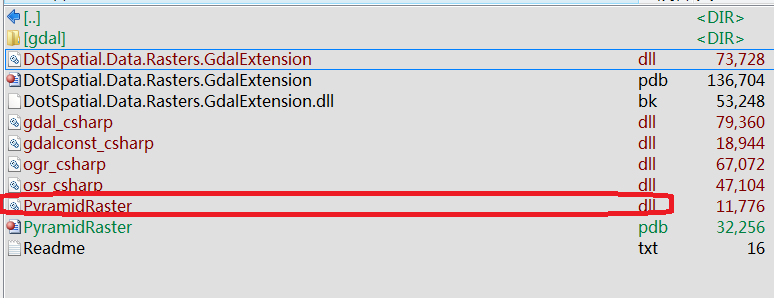
在一个基于DotSpatial开发的解决方案中，要调用GDAL，有两种方式，一种当然是直接引用GDAL相关DLL，进行开发，这种方式会导致有程序中出现两套GDAL库，发生冗余；第二种就是通过模仿DotSpatial.Data.Rasters.GdalExtension插件，添加一个新的插件，在插件里调用GDAL库，并通过预定义接口，将调用GDAL库生成或读取的结果传回解决方案，达到间接调用GDAL的目的。

第一种方法就不详细介绍了，本文主要介绍第二种方式。

# 二、条件或限制

通过添加插件的方式调用GDAL，有几个条件或限制：

1. 插件DLL的生成路径必须与DotSpatial.Data.Rasters.GdalExtension.dll在同个目录下，生成完之后如下图所示，其中红框中的PyramidRaster.dll就是我们自己生成的插件DLL；



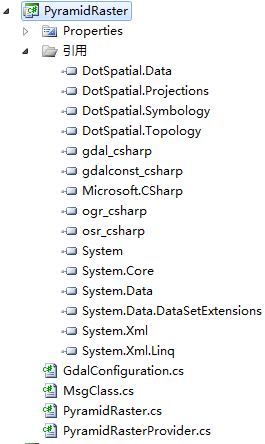
**图一**

1. 生成的DLL与主窗体之间的通信方式限制比较死，由于DotSpatial自身的机制，主库或主窗体是不直接加载插件DLL的，而是通过插件带的接口来与插件进行通信。插件与主窗体之间能够进行通信的接口只有两个地方：1是加载了插件之后，插件中所有基于IDataProvider接口的类都会被加载到DataManager.DefaultDataManager.DataProviders中去，用户可以在其中查找到自己的接口，并通过IDataProvier接口提供的相关函数或接口来间接调用GDAL

# 三、流程

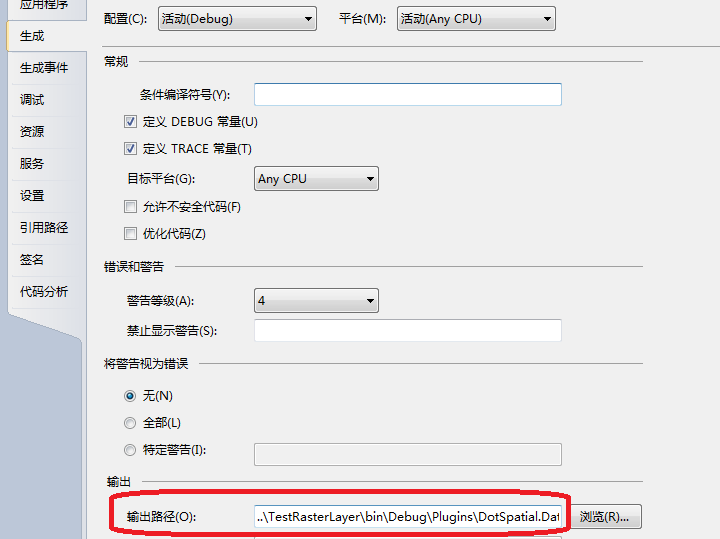
### 1. 新建解决方案

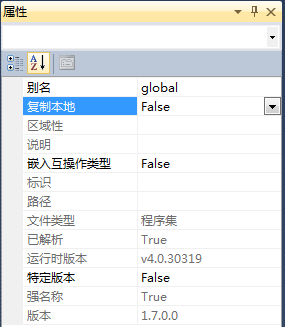
新建一个类库项目，本文例子中，为PyramidRaster.dll，如下图所示：



**图二**

在项目属性页中设置项目的生成路径，将其设置成与DotSpatial.Data.Rasters.GdalExtension.dll同一目录，如下图：



在这个项目中，添加到DotSpatial一些主要组件的引用，例如上图中的DotSpatial.Data等。注意这些组件的“复制本地”属性，都需要设置为False，否则会把这些DLL拷贝到图一所在目录下，如下图。

**图四**

再添加对GDAL相关组件的引用，如图二中的gdal\_csharp.dll等，注意这些组件的“复制本地”属性，需要设置成True。否则可能会出错

### 2. 声明GDAL环境

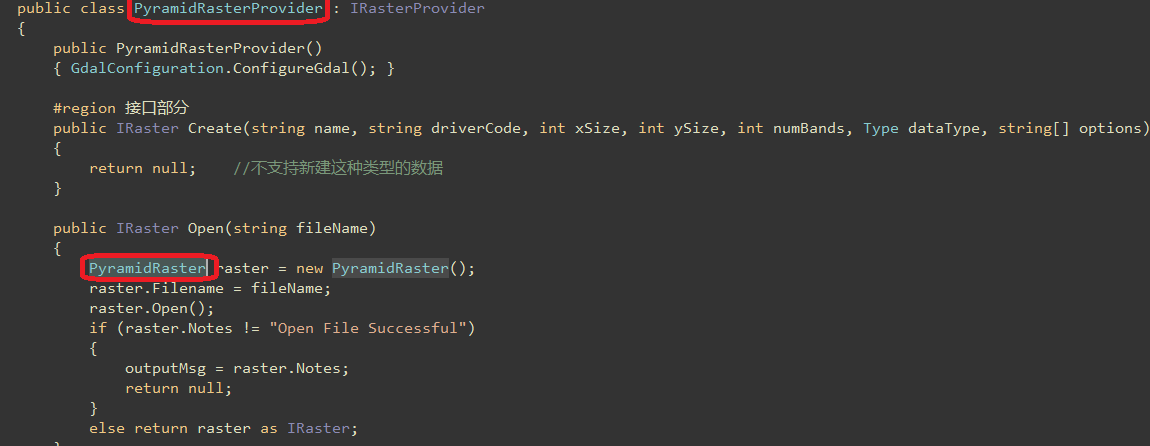
在使用GDAL组件之前，还需要声明一下GDAL的环境，包括声明GDAL的实际DLL所在路径，当前运行是X86还是X64，一些插件注册等。

这里的环境声明使用了与DotSpatial.Data.Rasters.GdalExtension.dll相同的方式，采用了GdalConfiguration.cs类，在IDataProvider类的构造函数中直接调用GdalConfiguration.ConfigureGdal()即可。详情参考附件

### 3. 添加实现IDataProvider接口的类

在项目中添加一个类，并实现IDataProvider接口。在本文的实例中，添加的类名称为PyramidRasterProvider，实现的是IRasterProvider接口（IRasterProvider接口实现了IDataProvider接口，所以相当于间接实现了IDataProvider接口）。

由于IRasterProvider接口的Open函数返回的是标准的IRaster接口对象，因此在本文的实例中，还添加一个实现IRaster接口的类，名称为PyramidRaster，主要的GDAL操作都是在PyramidRaster这个类里进行的，如下图所示：



相关代码参考附件。