[一、简介 4](#_Toc485579531)

[1.GDAL 4](#_Toc485579532)

[2.Ogr 4](#_Toc485579533)

[二、大体结构 5](#_Toc485579534)

[GDAL基础类 6](#_Toc485579535)

[抽象类 6](#_Toc485579536)

[具体描述 6](#_Toc485579537)

[OGR体系结构 9](#_Toc485579538)

[Gdal数据模型 10](#_Toc485579539)

[数据集（Dataset） 10](#_Toc485579540)

[坐标系统 11](#_Toc485579541)

[仿射地理变换 12](#_Toc485579542)

[GCPs 12](#_Toc485579543)

[元数据（Metadata） 13](#_Toc485579544)

[SUBDATASETS 域 13](#_Toc485579545)

[IMAGE\_STRUCTURE 域 14](#_Toc485579546)

[xml: 域 14](#_Toc485579547)

[Raster Band 14](#_Toc485579548)

[GDAL目录结构 17](#_Toc485579549)

[GDAL编译说明 24](#_Toc485579550)

[一、简单的编译 24](#_Toc485579551)

[二、自定义编译 26](#_Toc485579552)

[1、集成GEOS 27](#_Toc485579553)

[2、集成Proj4 28](#_Toc485579554)

[3、集成HDF数据读取 29](#_Toc485579555)

[三、其他方面 30](#_Toc485579556)

[1、makegdal\_gen.bat使用 30](#_Toc485579557)

[2、编译64位系统下的GDAL 30](#_Toc485579558)

[3、GDAL1.8后打开中文路径失败问题 31](#_Toc485579559)

[三、GDAL部分类说明 32](#_Toc485579560)

[GDALMajorObject 32](#_Toc485579561)

[GDALDriverManager 32](#_Toc485579562)

[GDALDriver 32](#_Toc485579563)

[GDALDataSet 32](#_Toc485579564)

[GDALRasterBand 48](#_Toc485579565)

[四、GDAL实用工具介绍 48](#_Toc485579566)

[五、开源栅格/矢量空间数据转换库-GDAL/OGR 49](#_Toc485579567)

[架构 49](#_Toc485579568)

[1、GDAL 50](#_Toc485579569)

[2、OGR 51](#_Toc485579570)

[1)OGR体系结构 51](#_Toc485579571)

[2) OGR的Geometry模型 52](#_Toc485579572)

[3）OGR的API使用范例 53](#_Toc485579573)

[六、GDAL源码剖析之Swig编译和帮助文档生成 54](#_Toc485579574)

[一、Swig编译 54](#_Toc485579575)

[1、Swig介绍 54](#_Toc485579576)

[2、编译C#版本GDAL 55](#_Toc485579577)

[3、编译Java版本GDAL 56](#_Toc485579578)

[4、编译Python版本GDAL 56](#_Toc485579579)

[二、开发帮助文档生成 57](#_Toc485579580)

[1、Doxygen介绍 58](#_Toc485579581)

[2、生成GDAL开发帮助文档 58](#_Toc485579582)

[3、Doxygen脚本配置以及代码注释书写规范 60](#_Toc485579583)

[七、GDAL源码剖析之命令行程序说明 61](#_Toc485579584)

[一、GDAL工具通用命令 61](#_Toc485579585)

[二、GDAL工具说明 64](#_Toc485579586)

[1. gdalinfo 输出文件信息 64](#_Toc485579587)

[2. gdal\_translate 格式转换 67](#_Toc485579588)

[3. gdaladdo 建立金字塔 70](#_Toc485579589)

[八、GDAL源码剖析之GDAL开发及其调试 72](#_Toc485579590)

[一、简单的调用 72](#_Toc485579591)

[二、GDAL源代码调试 76](#_Toc485579592)

[三、GDAL使用示例 77](#_Toc485579593)

[九、GDAL源码剖析之GDAL RasterIO使用说明 77](#_Toc485579594)

[RasterIO使用方法示例 81](#_Toc485579595)

# 一、简介

## 1.GDAL

简单地说，GDAL是一个操作各种栅格地理数据格式的库。包括读取、写入、转换、处理各种栅格数据格式（有些特定的格式对一些操作如写入等不支持）。它使用了一个单一的抽象数据模型就支持了大多数的栅格数据（GIS对栅格，矢量，3D数据模型的抽象能力实在令人叹服）。当然除了栅格操作，这个库还同时包括了操作矢量数据的另一个有名的库[ogr](http://ogr.maptools.org/)（ogr这个库另外介绍），这样这个库就同时具备了操作栅格和矢量数据的能力。

最重要的是这个库是跨平台的，开源的！如今这个库对各种数据格式的支持相当强大，几乎所有的GIS、RS所要处理的数据格式类型都包括在内。它的主页最后那些使用了它作为底层数据处理的软件列表中，不仅有[GRASS](http://grass.itc.it/index.html)、[Quantum GIS (QGIS)](http://qgis.sourceforge.net/)，更有[Google Earth](http://earth.google.com/)。

学习这个库是十分有必要的。该库在C/C++/python/ruby/VB/java/C#(这个暂时不完全支持)下都能用，而且都一样用。

GDAL(Geospatial Data Abstraction Library)是一个在X/MIT许可协议下的开源栅格空间数据转换库。它利用抽象数据模型来表达所支持的各种文件格式。它还有一系列命令行工具来进行数据转换和处理。

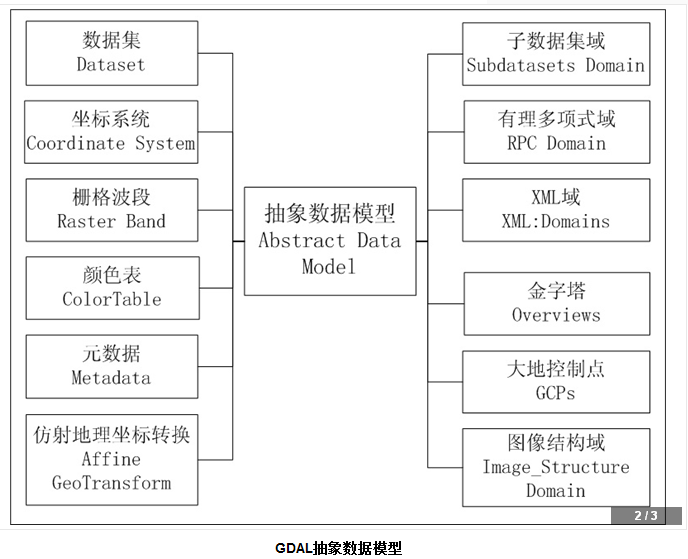
## 2.Ogr

OGR是GDAL项目的一个分支，功能与GDAL类似，只不过它提供对矢量数据的支持。 　　有很多著名的GIS类产品都使用了GDAL/OGR库，包括ESRI的ArgGIS 9.2，Google Earth和跨平台的GRASS GIS系统。

利用GDAL/OGR库，可以使基于Linux的地理空间数据管理系统提供对矢量和栅格文件数据的支持。

# 二、大体结构

GDAL使用抽象数据模型(abstract datamodel)来解析它所支持的数据格式，抽象数据模型包括数据集(dataset)，坐标系统，仿射地理坐标转换(Affine GeoTransform)， 大地控制点(GCPs)， 元数据(Metadata)，栅格波段(Raster Band)，颜色表(ColorTable)，子数据集域(Subdatasets Domain)，图像结构域(Image\_StructureDomain)，XML域(XML:Domains)。



## GDAL基础类

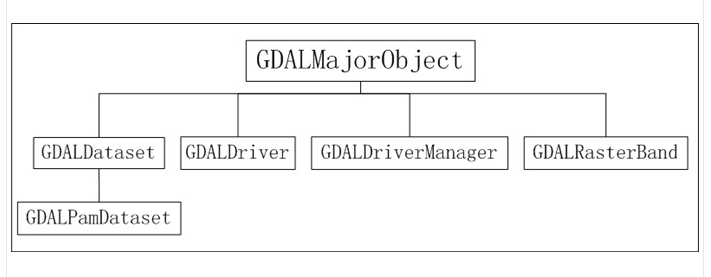
### 抽象类

GDALMajorObject类：带有元数据的对象。 GDAL数据处理结构

GDALDdataset类：通常是从一个栅格文件中提取的相关联的栅格波段集合和这些波段的元数据；GDALDdataset也负责所有栅格波段的地理坐标转换(georeferencing transform)和坐标系定义。

GDALDriver类：文件格式驱动类，GDAL会为每一个所支持的文件格式创建一个该类的实体，来管理该文件格式。

GDALDriverManager类：文件格式驱动管理类，用来管理GDALDriver类。



### 具体描述

Here are the classes, structs, unions and interfaces with brief descriptions:

以下是类，结构体，联合体和接口及其简要介绍。

\_CPLList

*列出元素构造。*

CPLHTTPResult

*Describe the result of a CPLHTTPFetch() call*

*描述了CPLHTTPFetch（）调用的结果*

CPLMimePart

Describe a part of a multipart message

描述了部分多部件信息

CPLODBCDriverInstaller

*A class providing functions to install or remove ODBC driver*

*运行安装或移除ODBC驱动的一个类*

CPLODBCSession

*A class representing an ODBC database session*

*Includes error collection services.*

*描述了一个ODBC 数据库会话的类*

CPLODBCStatement

*Abstraction for statement, and resultset*

*statement, and resultset（SQL的相关概念）的抽象*

CPLXMLNode

*Document node structure*

*节点结构文档*

GDAL\_GCP

*Ground Control Point*

*地面控制点*

GDALAsyncReader

*Class used as a session object for asynchronous requests*

*用来进行异步请求的会话对象的类*

GDALColorEntry

*Color tuple*

*颜色组*

GDALColorTable

*A color table / palette*

*颜色表/调色盘*

GDALDataset

*A set of associated raster bands, usually from one file*

一套关联栅格波段的集合，通常来自一个文件。

GDALDriver

*Format specific driver*

*格式的具体驱动*

GDALDriverManager

*Class for managing the registration of file format drivers*

*管理已注册的文件格式的驱动的类*

GDALGridDataMetricsOptions

*Data metrics method control options*

*数据metrics方法的控制选项*

GDALGridInverseDistanceToAPowerOptions

*Inverse distance to a power method control options*

*反距离加权插值方法控制选项*

GDALGridMovingAverageOptions

*Moving average method control options*

*移动平均插值法控制选项*

GDALGridNearestNeighborOptions

*Nearest neighbor method control options*

*最近邻近点插值法*

GDALMajorObject

*Object with metadata*

*元数据对象*

GDALPamDataset

*A subclass of GDALDataset which introduces the ability to save and restore auxilary information (coordinate system, gcps, metadata, etc) not supported by a file format via an "auxilary metadata" file with the .aux.xml extension*

*GDALDataset的一个子类*

GDALRasterAttributeTable

*Raster Attribute Table container*

*栅格属性表的容器*

GDALRasterBand

*A single raster band (or channel)*

*单个栅格波段（或频段）*

GDALRasterBlock

*A single raster block in the block cache*

*单个栅格区块在区块缓存区*

GDALWarpKernel

*Low level image warping class*

*底层图像变形类*

GDALWarpOperation

*High level image warping class*

*高层图像变形类*

GDALWarpOptions

*Warp control options for use with GDALWarpOperation::Initialize()*

*变形控制选项*

## OGR体系结构

OGR包括如下几部分：

Geometry：类Geometry(包括OGRGeometry等类)封装了OpenGIS的矢量数据模型，并提供了一些几何操作，WKB(Well KnowsBinary)和WKT(Well Known Text)格式之间的相互转换，以及空间参考系统(投影)。

Spatial Reference：类OGRSpatialReference封装了投影和基准面的定义。

Feature：类OGRFeature封装了一个完整feature的定义，一个完整的feature包括一个geometry和geometry的一系列属性。

Feature Definition：类OGRFeatureDefn里面封装了feature的属性，类型、名称及其默认的空间参考系统等。一个OGRFeatureDefn对象通常与一个层(layer)对应。

Layer：类OGRLayer是一个抽象基类，表示数据源类OGRDataSource里面的一层要素(feature)。 　　Data Source：类OGRDataSource是一个抽象基类，表示含有OGRLayer对象的一个文件或一个数据库。

Drivers：类OGRSFDriver对应于每一个所支持的矢量文件格式。类OGRSFDriver由类OGRSFDriverRegistrar来注册和管理。

## Gdal数据模型

在看python-gdal的相关文档时，里面提到的GDAL Data Model，其URL为http://www.gdal.org/gdal\_datamodel.html

这样重要的文档想来应该已经有人翻译了，于是找了一下，果不其然。还是在这儿记录一下，备查。

--------------------------------------------------------------------------------

该文档简要描述了GDAL的数据模型，该模型可以容纳其他各种信息。

### 数据集（Dataset）

一个dataset（对应GDALDataset类）是一个光栅数据以及和它有关系的信息的集合。 特别地dataset包含了光栅数据的大小（像素、线等）。dataset同时也为对应的 光栅数据指定了坐标系统。dataset本身还可以包含元数据，它们以一种键/值对 的方式来组织。

GDAL的数据集是基于OpenGIS Grid Coverages的格式定义的。

### 坐标系统

Dataset的坐标系统由OpenGIS WKT字符串定义，它包含了：

一个全局的坐标系名称。

一个地理坐标系名称。

一个基准标识符。

椭球体的名字。长半轴(semi-major axis)和反扁率(inverse flattening)。

初子午线(prime meridian)名和其与格林威治子午线的偏移值。

投影方法类型（如横轴莫卡托）。

投影参数列表（如中央经线等）。

一个单位的名称和其到米和弧度单位的转换参数。

轴线的名称和顺序。

在预定义的权威坐标系中的编码（如EPSG）。

更多信息请参考OpenGIS WKT坐标系统定义，以及osr教程文档和 OGRSpatialReference类的描述文档。

在GDAL中，返回坐标系统的函数是GDALDataset::GetProjectionRef()。 它返回的坐标系统描述了地理参考坐标，暗含着仿射地理参考转换，这地理参考转换是由GDALDataset::GetGeoTransform()来返回。由GCPs地理参考坐标描述的坐标系统是由 GDALDataset::GetGCPProjection()返回的。

注意，返回的坐标系统字符串“”表示未知的地理参考坐标系统。

### 仿射地理变换

GDAL数据集有两种方式描述栅格位置（用点/线坐标系）以及地理参考坐标系之间的关系。 第一种也是比较常用的是使用仿射转换，另一种则是GCPs。

仿射变换由6个参数构成，它们由GDALDataset::GetGeoTransform()返回它们把点/线坐标， 用下面的关系转将点/线影射到地理坐标：

Xgeo = GT(0) + Xpixel\*GT(1) + Yline\*GT(2)

Ygeo = GT(3) + Xpixel\*GT(4) + Yline\*GT(5)

假设影像上面为北方，GT2和GT4参数为0，而GT1是象元宽，GT5是象元高， （GT0，GT3）点位置是影像的左上角。

注意，上面所说的点/线坐标系是从左上角(0,0)点到右下角，也就是坐标轴从 左到右增长，从上到下增长的坐标系（即影象的行列从左下角开始计算）。 点/线位置中心是(0.5,0.5)。

### GCPs

数据集可以由一系列控制点来定义空间参考坐标系。所有的GCPs共用一个地理参考坐标系， 由GDALDataset::GetGCPProjection()返回。每个GCP（对应GDAL\_GCP类）包含下面内容：

typedef struct

{

char \*pszId;

char \*pszInfo;

double dfGCPPixel;

double dfGCPLine;

double dfGCPX;

double dfGCPY;

double dfGCPZ;

} GDAL\_GCP;

字符串pszid是本GCP在数据集中一系列GCP点中惟一的标示字符串（通常是数字）。 字符串pszInfo通常为空，但是也可以包含用户针对GCP定义的一些文本信息。甚至还 可能是GCP状态中包含机器可分析信息，虽然现在还支持。

坐标（dfGCPPixel,dfGCPLine）是栅格中的GCP位置。坐标（dfGCPX，dfGCPY，dfGCPZ）是 联合的地理参考位置（dfGCPZ通常是0）。

GDAL数据模型没有实现由GCPs产生坐标系的变化的机制，而是把具体的操作留给用户实现。 通常1到5阶多项式是常用的方法。

通常一个数据集会包含仿射地理变换，或GCPS中的一个，或者两个都没有。两个都有很少见， 而且无法用权威坐标系定义。

### 元数据（Metadata）

GDAL元数据是一种辅助的数据格式，并且以键/值对序列的方式呈现。一般键的 名称有严格的定义（没有空白、或某些特殊字符等）。键所对应的值 可以是任意的长度， 包含任意的内容（NULL字符除外）。元数据处理系统一般不能处理很大的数据，例如一 个大于100K的数据很可能导致处理的终止。

虽然某些键现在可能没有，但是以后也可能会被定义。 一些格式的数据可以支持一些基本的元数据（可以由用户自己定义）， 可以使用驱动程序访问这些键/值。例如，如果数据含有date/time标志， TIFF格式的驱动程序可能只是简单地返回基本的信息：

TIFFTAG\_DATETIME=1999:05:11 11:29:56

元数据的键/值对还可以被组织到某些域中，默认的域是没有名字的。当然， 为了保存某些特殊的信息可能需要定义特殊的域。目前虽然不能列举出一个对象所需要 的所有域，但是程序可以测试任何我们已经知道确切含义的域。

#### SUBDATASETS 域

SUBDATASETS域保存了一个子datasets列表。这个列表通常是对应一个复合影象中每个 子影象的位置（像HDF或NITF）。例如一个由4个影象组成的NITF可能含有类似下面的 子datasets列表：

SUBDATASET\_1\_NAME=NITF\_IM:0:multi\_1b.ntf

SUBDATASET\_1\_DESC=Image 1 of multi\_1b.ntf

SUBDATASET\_2\_NAME=NITF\_IM:1:multi\_1b.ntf

SUBDATASET\_2\_DESC=Image 2 of multi\_1b.ntf

SUBDATASET\_3\_NAME=NITF\_IM:2:multi\_1b.ntf

SUBDATASET\_3\_DESC=Image 3 of multi\_1b.ntf

SUBDATASET\_4\_NAME=NITF\_IM:3:multi\_1b.ntf

SUBDATASET\_4\_DESC=Image 4 of multi\_1b.ntf

SUBDATASET\_5\_NAME=NITF\_IM:4:multi\_1b.ntf

SUBDATASET\_5\_DESC=Image 5 of multi\_1b.ntf

以\_NAME为后缀的键所对应的值可以传递给GDALOpen()用于访问文件。以\_DESC为 后缀的键所对应的值可以以一种友好的方式显示给用户。

#### IMAGE\_STRUCTURE 域

默认域中的元数据一般和影象有关，但是和影象的具体格式无关。换言之， 与影象格式无关的信息在复制dataset的时候一般也会被复制。为了处理那些与 影象格式相关的数据，一般将相关的键/值对放到IMAGE\_STRUCTURE域中，并且在复制 dataset的时候可以不被复制。

IMAGE\_STRUCTURE域中的一个项是数据的压缩方式，对应的键名为COMPRESSION， 对应的值说明的压缩的方式。

#### xml: 域

任何以"xml:"为前缀名的域都不是一个普通的键/值对方式的元数据。它是一个 单一的XML文档，以一个大的字符串方式保存

#### Raster Band

一个波段的光栅数据对应GDAL中的 GDALRasterBand类。它描述了单个波段的 band/channel/layer。它并不是一次描述整个影象。例如，一个24bit的RGB影象 在一个dataset中一般被描述为三个波段，分别对应red/green/blue三中颜色。

一个波段的光栅数据具有以下性质：

行列数。如果没有抽样的话，该值和dataset定义的相同。

数据类型 (GDALDataType)。可以是Byte、UInt16、Int16、UInt32、Int32、Float32、Float64，或者是复合类型CInt16、CInt32、CFloat32或 CFloat64。

块的大小。这是建议的缓冲块的大小。对于块式存储的影象来说，可能是一个块的大小。对于以扫描线方式的影象，可能是一个扫描线的大小。

元数据中的name/value列表，和dataset中相同，但是可能更侧重当前波段的特征。

可选择的参数（字符串）。

An optional list of category names (effectively class names in a thematic image).

An optional minimum and maximum value.

An optional offset and scale for transforming raster values into meaning full values (ie translate height to meters)

可选的光栅名称。这个特性可以用来指定一些比较重要的数据。

每个波段解释，具体为：

GCI\_Undefined: 默认，未知信息。

GCI\_GrayIndex: 为一个独立的灰度影象。

GCI\_PaletteIndex: this raster acts as an index into a color table

GCI\_RedBand: RGB或RGBA影象的红色波段。

GCI\_GreenBand: RGB或RGBA影象的绿色波段。

GCI\_BlueBand: RGB或RGBA影象的蓝色波段。

GCI\_AlphaBand: RGBA影象的alpha通道。

GCI\_HueBand: HLS影象的色调。

GCI\_SaturationBand: HLS影象的饱和度。

GCI\_LightnessBand: HLS影象的光强。

GCI\_CyanBand: CMY或CMYK影象的cyan波段。

GCI\_MagentaBand: CMY或CMYK影象的magenta波段。

GCI\_YellowBand: CMY或CMYK影象的yellow波段。

GCI\_BlackBand: CMYK影象的black波段。

颜色表，描述光栅数据的更多的细节。

Knowledge of reduced resolution overviews (pyramids) if available.

颜色表

一个颜色表包含0个或多个颜色结构。结构体定义如下：

typedef struct

{

/- gray, red, cyan or hue -/

short c1;

/- green, magenta, or lightness -/

short c2;

/- blue, yellow, or saturation -/

short c3;

/- alpha or blackband -/

short c4;

} GDALColorEntry;

颜色表同时还对应一个调色板（GDALPaletteInterp），GDALColorEntry中的c1/c2/c3/c4的 值可以作为调色板索引得到真正的颜色值。

GPI\_Gray: c1作为灰度值。

GPI\_RGB: c1/c2/c3依次为Red/Green/Blue，c4对应alpha通道。

GPI\_CMYK: c1为cyan，c2为magenta，c3为yellow，c4为black。

GPI\_HLS: c1为hue，c2为lightness，c3为saturation。

用颜色表表示时，每个像素保存的只是像素颜色在颜色表的位置。每个颜色的索引从 0开始递增。这里并没有提供针对颜色表的缩放机制。

总结

A band may have zero or more overviews. Each overview is represented as a "free standing" GDALRasterBand. The size (in pixels and lines) of the overview will be different than the underlying raster, but the geographic region covered by overviews is the same as the full resolution band.

The overviews are used to display reduced resolution overviews more quickly than could be done by reading all the full resolution data and downsampling.

Bands also have a HasArbitraryOverviews property which is TRUE if the raster can be read at any resolution efficiently but with no distinct overview levels. This applies to some FFT encoded images, or images pulled through gateways (like OGDI) where downsampling can be done efficiently at the remote point.

## GDAL目录结构

以下摘自：

<http://blog.csdn.net/liminlu0314/article/details/6933772>

首先对于GDAL的目录结构进行一个简单的介绍。GDAL源代码下载地址：http://trac.osgeo.org/gdal/wiki/DownloadSource，或者安装svn从源代码服务器下载，svn地址是：<http://svn.osgeo.org/gdal/trunk>。

如果是使用下载的压缩包，其目录结构如下图：

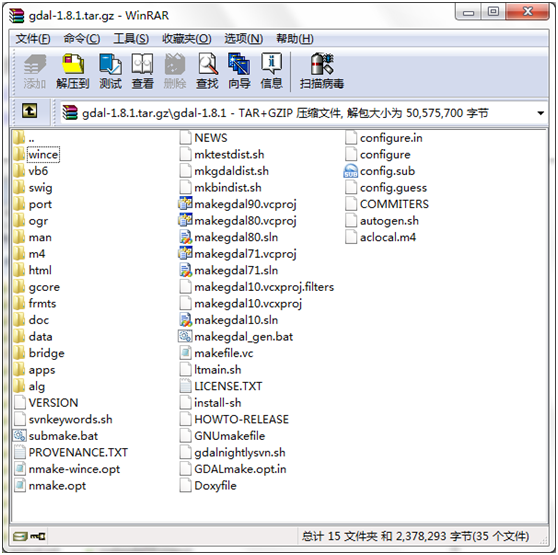


图1 GDAL源码压缩包文件目录结构

如果使用svn下载的源代码，目录结构如下：

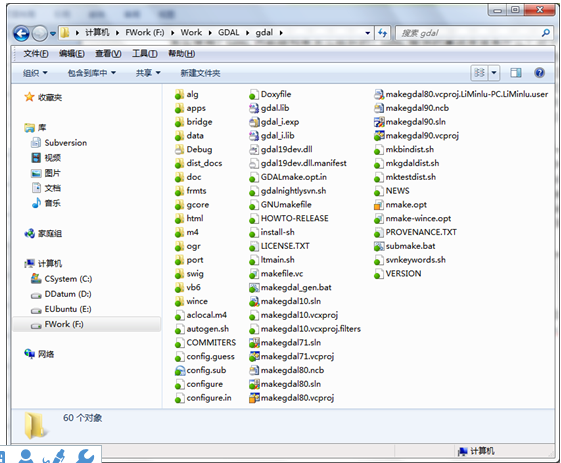


图2 GDAL SVN源码文件目录结构

从上面两张图中可以看出，GDAL的目录结构不管是用什么方式获取的源代码，它的目录结构都是一样的，下面就针对目录结构中的每个文件夹和文件做一个简单的说明。（按照字母顺序来进行说明）

1、alg文件夹：alg文件夹中存放的是GDAL库中提供的一些算法的源代码，这些算法包括但不限于：DEM生成等高线算法；图像纠正算法（几何纠正，TPS纠正，正射RPC纠正）；栅格矢量化算法；矢量栅格化算法；格网计算算法；PCT和RGB互转算法；分类图的小碎斑块去除算法等。

2、apps文件夹：apps文件夹中存放的是GDAL库中提供的一些命令行工具集的源代码，这些工具集的介绍可以参考http://gdal.org/gdal\_utilities.html，将来我会对这些工具做一个简单的说明。其中有些工具非常的有用，比如gdalinfo，可以使用该工具来查看图像的元数据信息等。

3、bridge文件夹：bridge文件夹中存放的是用来连接GDAL抽象类的定义以及GDAL自己的结构体定义和实现的源代码。具体到后面涉及到GDAL的实现原理时会对该文件夹作一个比较详细的介绍。

4、data文件夹：data文件夹中存放的是GDAL库中需要用到的一些“配置文件”（此处用配置文件可能不太准确），这些文件主要有ESRI的投影文件，ESPG的投影文件，PCI的投影和椭球体文件，autoCAD的头文件，以及其他的一些文件。在GDAL库中有很多时候会自动读取该文件夹中的文件，一般是通过环境变量来查找该文件夹，环境变量的名字叫GDAL\_DATA，值就是data文件夹的路径，或者可以在你的程序中使用函数 CPLSetConfigOption("GDAL\_DATA","C:\GDAL\data");来进行设置该文件夹的目录，如果没有设置GDAL会自动从环境变量中查找，如果还是没有找到，那么GDAL可能会提示错误，比如如果不设置GDAL\_DATA，那么在写如atuocad的dxf格式的时候就提示创建不成功，后面遇到的话会再进行说明。

5、doc文件夹：doc文件夹存放的是用来生产GDAL帮助文档的一些dox文件，dox文件是使用doxygen工具来进行生成的，后面会对doxygen工具作一个简单的介绍和说明，以及在自己的工程中怎么使用doxygen生成自己的程序的开发帮助文档等。总之一句话，这个文件夹就是用来生成GDAL库的帮助文档的一些东西。后面会告诉大家如何生成一份GDAL的帮助文档，当然你也可以把gdal.org整个网站抓下来，J

6、frmts文件夹：这个文件夹可以说是GDAL代码中东西最多的一个文件夹了，每次更新GDAL的版本后都会发现这个文件夹中会多出几个文件夹，同时GDAL也会中支持的文件格式中多出来几个新的文件格式。没错，这个文件夹存放的就是GDAL针对每种不同的特定的图像格式解析的源代码，可以举几个简单的例子，比如bmp文件夹就是解析BMP图像的，hfa文件夹就是用来解析Erdas的img图像格式，raw文件夹用来读取ENVI的hdr文件，还有pcidsk文件夹就是读取PCI的pix格式的等等。所以这个文件夹存放的是解析各个文件格式的源代码。

7、gcore文件夹：通过名字大家也应该知道这个文件夹是做什么的了，叫core的肯定都是很核心的东西了，这个文件夹就是GDAL的灵魂所在，主要存放的GDAL抽象类的数据集，波段，图像读写接口等都在这个里面实现的。如果要知道GDAL的抽象类是怎么对图像格式进行抽象的，可以看看这个里面的代码。

8、html文件夹：html文件夹如果使用压缩包的话，里面应该是空的，这个文件夹主要是用来存放GDAL的生成的帮助文档的地方，主要是使用前面介绍的doc文件夹中的dox脚本，使用doxygen工具生成的GDAL帮助文档会出现在这个文件夹中。后面会和doc文件夹一起进行一个详细介绍。

9、m4文件夹：m4文件夹存放的有好几个已m4为后缀名的文件，m4文件叫MacroProcessor Library，m4文件是编译基础中最核心的文件，这个文件主要是用autoconf来产生configure配置文件，继而自动生成Makefile文件。这个文件夹中Windows平台下貌似没什么作用，可能是我还不知道吧，在此略过。

10、man文件夹：man文件夹貌似是用来生成linux或者其他平台下的帮助文件，估计是可以使用linux下的man帮助吧。Windows平台下貌似也没什么用，略过。

11、ogr文件夹：用过GDAL的肯定知道ogr库吧，在很久很久以前，GDAL和OGR是两个库，GDAL专门负责读取栅格数据，OGR库负责读取矢量数据，然后可能是因为两个库分开有些不方便，比如GDAL的算法库中经常会用到矢量数据的读取，或者还有别的原因吧，现在将这两个库整合在了一起，目前OGR库就是GDAL库的一个子集。其实OGR库还是可以单独编译出来的。Ogr文件夹就是存放OGR库源代码的文件夹。这个文件夹里面也是有很多东西的，后面再详细进行介绍。

12、port文件夹：port文件夹中存放的是port库的东西，port库对于GDAL库来说是一个底层的支持库，port库中定义了一些字符串的操作，文件处理，网页请求，数据库连接，哈希表，字符加密文件压缩等基础的函数。比如GDAL中所有的导出函数符号CPL\_DLL就是在这个port文件夹中定义的，还有frmts文件夹中，打开文件，打开数据库，打开网络路径等都是用的port库，以及字符串的处理等。

13、swig文件夹：swig文件夹主要是存放swig的脚本。Swig全称叫SimplifiedWrapper and Interface Generator，网址是www.swig.org， swig的作用就是可以将C/C++写的库封装为Python，C#，Java，Perl和 Ruby等其他语言的访问接口。网上GDAL的C#版本就是使用swig来编译出来实现的，很强大吧。后面有时间的话，会写一篇关于swig编译GDAL的文章。

14、vb6文件夹：这个文件夹中用来将GDAL编译成一个VB6的模块，对于里面具体的文件说明，以及如何编译参考文件夹中的readme.txt，由于我对VB6的使用，还是仅限于拖个按钮，写个单击事件的基础，对于模块之间的调用，实在是不太懂，所在就不说明了。想用VB6使用GDAL的可以自己研究一下，应该也不是很难。

15、wince文件夹：顾名思义，这个文件夹中的东西就是用来编译Windows CE平台下的GDAL库用的，具体编译请参考其中的说明文档。

文件夹介绍完了，下面对文件进行一个大概说明：

1、aclocal.m4：同上面的m4文件夹

2、autogen.sh：Linux平台下的shell文件，用来调用autoconf来产生configure配置文件的。

3、COMMITERS：该文件中的内容是GDAL开发人员的信息，姓名，联系邮箱以及各自负责开发的模块说明等。

4、config.guess，config.sub，configure，configure.in：这四个文件貌似都是linux平台下的配置文件，中Windows下没啥用，略过。

5、Doxyfile：Doxyfile就是前面doc文件夹中说明提到的doxygen的工程文件，用来生成帮助文档用的，后面在介绍doxygen的使用是会对该文件进行一个说明。

6、GDALmake.opt.in：这个文件是linux平台下的GDAL库编译配置文件，功能在后面的nmake.opt中介绍。

7、gdalnightlysvn.sh：Linux平台下调用svn获取GDAL源代码的一个shell脚本。

8、GNUmakefile：GNU的make文件。

9、HOWTO-RELEASE：GDAL发布版本的一些说明。

10、install-sh：GDAL的安装shell脚本，Linux平台下。

11、LICENSE.TXT：GDAL的许可说明文件。

12、ltmain.sh：libtool的shell脚本，Linux平台下，Windows下貌似没用到。

13makefile.vc：GDAL的编译文件，用来将源代码编译成dll文件，后面会在GDAL编译中作一个介绍。

14、makegdal\_gen.bat：用来生成VS的工程文件的一个批处理文件，后面在GDAL编译中会对该文件的使用方式做一个说明。

15、makegdal10.sln：文件夹中所有的sln文件都是VS的项目文件，文件名后的数字代表的是VS的版本号。

16、 makegdal10.vcxproj和makegdal71.vcproj：VS的工程文件，该文件可以由makegdal\_gen.bat文件自动生成，后面详细介绍。

17、mkbindist.sh，mkgdaldist.sh和mktestdist.sh：三个shell脚本文件，Windows下没用，略过。

18、NEWS：GDAL的新增功能，以及修复的bug记录等。

19、nmake.opt：GDAL编译选项配置文件，在编译GDAL中，可以指定GDAL绑定的其他库等都在这个里面进行设置。在后面的GDAL编译中会详细介绍说明。

20、nmake-wince.opt：编译wince版本的编译选项配置文件。

21、PROVENANCE.TXT：GDAL目录说明文件，如果上面说明的不够清楚，可以参考这个文件。

22、submake.bat：一个编译的批处理文件，目前没啥用。

23、svnkeywords.sh：又是svn的一个shell脚本。

24、 VERSION：GDAL版本信息。

## GDAL编译说明

### 一、简单的编译

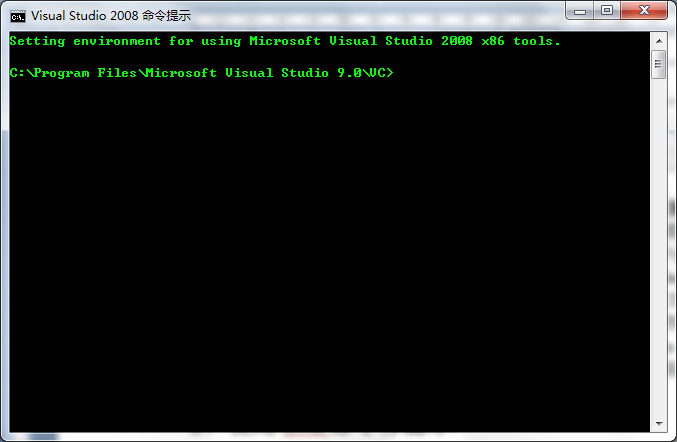
1、使用VisualStudio IDE编译

首先进入GDAL的源代码目录，可以看到有几个sln为后缀的文件名，比如makegdal10.sln，makegdal80.sln，makegdal71.sln，makegdal90.sln 。这些文件是VisualStudio的工程文件，后面的数字对应的VS的版本号，71表示的VS2003，80表示VS2005，90表示VS2008，还有10表示VS2010等。根据自己电脑安装的VS版本，打开对应的文件，如下图所示（使用VS2008SP1版本，打开makegdal90.sln文件）：

然后在左侧解决方案右键，弹出菜单中选择“生成”或者“重新生成”命令，然后GDAL就会开始编译，等待输出窗口中提示，执行完成，生成成功等信息后，就表示GDAL已经完成编译。同时会在GDAL的源代码目录中会出现gdal.lib，gdal\_i.lib，gdal18.dll等文件，如果你没有修改GDAL中的nmake.opt文件的话，那么同时会在你的C盘中会出现一个叫“C:\warmerda\bld”的文件夹，里面会包含三个文件夹，分别是bin，data和html。其中bin文件夹中存放的是编译出来的GDAL的可执行程序，包括GDAL提供的十几个工具集；data文件夹就是在第一节中的介绍的data文件夹；html文件夹中存放的是各种数据格式的说明文档。

2、使用cmd命令行编译

使用cmd命令行编译，首先在“开始菜单\所有程序\Microsoft Visual Studio 2008\Visual Studio Tools\ Visual Studio 2008命令提示”，点击“Visual Studio 2008 命令提示”会弹出下面的界面：



然后使用cd命令，切换到GDAL的源代码目录，如下图所示：

切换到GDAL的源代码目录后，依次敲入下面的命令行后回车，等待编译结束即可。

nmake -f makefile.vc

nmake -f makefile.vc install

nmake -f makefile.vc devinstall

同时还有其他的命令，如：

nmake -f makefile.vc clean

nmake -f makefile.vc MSVC\_VER=1400clean

nmake -f makefile.vc MSVC\_VER=1400DEBUG=1

上面六行的命令含义依次是：

编译GDAL库

编译GDAL库，并安装（这里安装的意思就是将生成的dll，exe等文件拷贝到C:\warmerda\bld目录），

编译GDAL库，并安装开发者模式（安装的意思同上，开发者模式意思是将开发用的include文件夹中的头文件和lib文件一同拷贝到C:\warmerda\bld目录，此时会在C:\warmerda\bld目录中多出来两个文件夹，分别是include和lib，分别存放的是GDAL的头文件和lib文件，用于调用GDAL库使用）。

清理GDAL库，同时会删除编译GDAL库所生成的临时文件，作用相当于在VS环境中的清理命令。

作用同上，但是添加了一个MSVC\_VER=1400，表示使用VS2005编译。

编译GDAL库的debug模式，可以用来调试GDAL源码。

### 二、自定义编译

GDAL的强大之处不单单在于可以读取栅格和矢量数据，同时它的强大之处还在于下面几个方面，第一可以进行矢量图形之间的一些常用操作，比如：求交，求并，缓冲区等等。第二可以进行投影和坐标转换。如果使用GDAL默认的编译方式，那么上述的两个非常强大的功能您将不能使用，因为GDAL这两大功能是基于另外的两个开源库GEOS（Geometry Engine, Open Source）库和PROJ4库来实现的。下面对这两大库分别做一个简单的说明，以及如何修改编译文件，让GDAL能够拥有这两大功能。

#### 1、集成GEOS

关于GEOS库的说明，网上有很多，同时在GEOS的官网http://geos.osgeo.org有详细的说明，简单的来说，GEOS提供了OGC规范中简单几何要素对象操作的C++语言的实现。在地理信息系统领域，拓扑模型是重要的，其计算方法简单但是很难得以实现。使得GEOS不同于其他项目的也正是“空间谓词”与“空间操作”。空间谓词是比较两个空间对象并返回一个布尔变量值作为结果，它表明了存在于两个空间对象之间特殊的关系。比如典型的空间谓词有Contains(), Intersects(), Touches(), andCrosses()函数等。GEOS项目中对该些函数的实现是异常强壮的，即使是奇异几何对象或是临时的坐标系统运算也不能使其运算不正常或计算错误。目前绝大多数的商业软件仍然在最基础的空间谓词处理上相对成熟，这正是GEOS项目的重要意义。“空间操作”则主要是对两个几何对象进行计算并且返回一个新的几何实体。比较典型的操作函数如Difference(), Union()以及Buffer()等。GEOS中的操作算法已经被广泛的经过了测试。GEOS类库被各类开源空间信息软件项目广泛应用，使用GEOS，它们可以基于最新的规范的几何实体来完成，同时也拥有了复杂空间方法的实现。

关于GEOS的说明和编译，后面会单独写一篇文件进行介绍，这里假设已经下载的是编译好的GEOS库。

首先使用记事本或者其他的文本编辑器打开GDAL源代码目录下的nmake.opt文件，找到“# Uncomment for GEOS support”这句，大概在405行左右，将下面三行代码：

#GEOS\_DIR=C:/warmerda/geos

#GEOS\_CFLAGS =-I$(GEOS\_DIR)/capi -I$(GEOS\_DIR)/source/headers -DHAVE\_GEOS

#GEOS\_LIB =$(GEOS\_DIR)/source/geos\_c\_i.lib

修改为：

GEOS\_DIR=F:\Work\3rdPart\geos-3.2.2

GEOS\_CFLAGS =-I$(GEOS\_DIR)/capi -I$(GEOS\_DIR)/source/headers -DHAVE\_GEOS

GEOS\_LIB = $(GEOS\_DIR)/source/geos\_c\_i.lib

其中F:\Work\3rdPart\geos-3.2.2是我本机的GEOS存放的主目录，后面两行设置的是GEOS的头文件目录和lib文件路径。设置好后保存即可。对比结果如下图如

保存完nmake.opt之后，按照第一步中的编译方式进行编译即可。编译后的GDAL就将会支持图形之间的操作等处理。函数主要是在OGR库中，后面会在OGR库中进行详细的介绍说明。编译后，千万别忘记将geos\_c.dll文件拷贝到gdal18.dll的同级目录下，否则会提示你找不到geos\_c.dll文件。

#### 2、集成Proj4

Proj4是一套开源的坐标投影转换类库，它可以完成在两套不同制图投影系统之间的转换，同样不同的椭球体或大地基准面之间也可以成功的完成转换。GDAL中用到的坐标转换，投影转换，几何纠正，正射纠正等算法，都离不开坐标转换，也就是说要使用这些算法，必须有proj4库的支持才行。同GEOS库的配置方法，在nmake.opt文件中，找到proj4库的位置，大概在352行左右。将下面的三行代码：

#PROJ\_FLAGS =-DPROJ\_STATIC

#PROJ\_INCLUDE =-Id:\projects\proj.4\src

#PROJ\_LIBRARY =d:\projects\proj.4\src\proj\_i.lib

修改为：

#PROJ\_FLAGS =-DPROJ\_STATIC

PROJ\_INCLUDE =-IF:\Work\3rdPart\proj-4.7.0\src

PROJ\_LIBRARY =F:\Work\3rdPart\proj-4.7.0\src\proj\_i.lib

其中第一行表示是否使用静态链接的方式，第二行的路径表示，proj库存放的位置，第三行为proj库的lib文件所在路径。修改后保存即可，对比结果如下图如下：

同GEOS库一样，保存完nmake.opt之后，按照第一步中的编译方式进行编译即可。对于Proj库的使用后面会在有一篇文章对其做一个简单的介绍说明。编译后，同样千万别忘记将proj.dll文件拷贝到gdal18.dll的同级目录下，否则会提示你找不到proj.dll文件。

#### 3、集成HDF数据读取

通过上面GEOS和PROJ库的介绍，相信对gdal的配置文件，nmake.opt有一个比较初步的了解了吧，那么下面对于使用GDAL支持hdf数据的读取也是同样，先下载好hdf4和hdf5的库，我用的是HDF4.2.6和HDF5-1.8.7两个库，在hdf的官方网上有编译好的库，直接下载编译好的库即可，对于hdf库的编译，我没有进行编译过，应该和其他的开源库都是差不多吧。同时官网提供了32位的库和64位的库，这里都是按照32位的库进行介绍，后面会有一个gdal的64位库的编译介绍。

将下载好的HDF4.2.6和HDF5-1.8.7两个库解压，然后修改nmake.opt文件中的278行左右，代码如下：

# Uncomment thefollowing and update to enable NCSA HDF Release 4 support.

#HDF4\_PLUGIN = NO

#HDF4\_DIR = D:\warmerda\HDF41r5

#HDF4\_LIB = /LIBPATH:$(HDF4\_DIR)\lib Ws2\_32.lib

# Uncomment thefollowing and update to enable NCSA HDF Release 5 support.

#HDF5\_PLUGIN = NO

#HDF5\_DIR = c:\warmerda\supportlibs\hdf5\5-164-win

#HDF5\_LIB = $(HDF5\_DIR)\dll\hdf5dll.lib

修改为下面的代码：

# Uncomment thefollowing and update to enable NCSA HDF Release 4 support.

HDF4\_PLUGIN = NO

HDF4\_DIR = F:\Work\3rdPart\HDF4.2.6\_win\_x86

HDF4\_LIB = $(HDF4\_DIR)\dll\hd426m.lib$(HDF4\_DIR)\dll\hm426m.lib \

$(HDF4\_DIR)\lib\hd426.lib$(HDF4\_DIR)\lib\hm426.lib Ws2\_32.lib

# Uncomment thefollowing and update to enable NCSA HDF Release 5 support.

HDF5\_PLUGIN = NO

HDF5\_DIR = F:\Work\3rdPart\HDF5-1.8.7\_win\_x86

HDF5\_LIB = $(HDF5\_DIR)\dll\hdf5dll.lib

对比代码如下图：

保存，然后编译gdal即可，同时将hdf库中的dll文件夹下的dll文件拷贝到gdal18.dll的同级目录下。

## 三、其他方面

#### 1、makegdal\_gen.bat使用

对于makegdal\_gen.bat的作用，在上一篇文章中已经进行了介绍，下面对怎么使用该文件生成VS的工程文件做一个说明。

首先打开cmd命令行窗口，使用cd命令切换到GDAL源代码目录，然后输入makegdal\_gen.bat回车，会得到该工具的一个简单实用帮助，如下图所示：

该工具的使用方法是带有命令行参数的一个批处理工具，（在后面对GDAL工具集的介绍中会对带有参数的命令行程序，以及编写带有命令行的程序有一个比较详细的说明）。通过上图可以看出该工具的基本语法是：

makegdal\_gen 7.10 >makegdal71.vcproj

makegdal\_gen 8.00 >makegdal80.vcproj

通过上面的示例可以看出，该工具的命令行参数分别是，首先是VS的版本号，具体版本号参考本文第一小节，然后跟一个大于号“>”，最后是输出的VS的工程的名字。那么现在我要使用该命令行生成一个VS2008版本的工程文件，我可以输入下面的命令，然后回车即可：

makegdal\_gen 9.00 >makegdal90.vcproj

#### 2、编译64位系统下的GDAL

对于GDAL的64位系统的编译，基本和32位系统的编译一样，首先在VS的工程中，打开配置管理器，然后再活动解决方案平台的下拉列表中选择新建，然后弹出，新建解决方案平台对话框，选择新平台为x64（需要在安装VS的时候安装64位的编译环境），然后点击确定即可。最后在VS中选择X64进行编译即可。如下图所示：

对于使用cmd命令行编译，基本同本文开始，不一样的只有，在开始菜单选择的不是“Visual Studio 2008 命令提示”而是“Visual Studio 2008 x64 兼容工具命令提示”，剩下的编译步骤跟前面的一样。

在编译开始之前，还需要打开nmake.opt文件，找到131行处的“#WIN64=YES”，将前面的“#”去掉，保存，然后开始编译。如果就这样编译过去的话，那么恭喜你，如果不能顺利编译过去的，那么需要按照下面的步骤进行一点点设置。

1：在GDAL目录下的nmake.opt文件中，找到SYM\_PREFIX的定义，应该在438行左右

将SYM\_PREFIX=\_ 改为SYM\_PREFIX= 就是将最后的下划线去掉

2：在GDAL目录下的makefile.vc文件中，找到46行左右的代码，如下：

BASE\_INCLUDE =/INCLUDE:\_GDALSimpleImageWarp@36 \

/INCLUDE:\_GDALReprojectImage@48 \

/INCLUDE:\_GDALComputeMedianCutPCT@32 \

/INCLUDE:\_GDALDitherRGB2PCT@28 \

/INCLUDE:\_OCTNewCoordinateTransformation@8$(VB6\_SAFEARRAYSYM)

修改为：

BASE\_INCLUDE =/INCLUDE:$(SYM\_PREFIX)GDALSimpleImageWarp \

/INCLUDE:$(SYM\_PREFIX)GDALReprojectImage \

/INCLUDE:$(SYM\_PREFIX)GDALComputeMedianCutPCT\

/INCLUDE:$(SYM\_PREFIX)GDALDitherRGB2PCT \

/INCLUDE:$(SYM\_PREFIX)OCTNewCoordinateTransformation $(VB6\_SAFEARRAYSYM)

就是将后面的@开始，后面的数字删除。现在开始编译吧，呵呵。祝你成功！此处可以参考我的博文：http://liminlu0314.blog.163.com/blog/static/111800520106223508717/

#### 3、GDAL1.8后打开中文路径失败问题

此处可以参考我的博文：http://blog.csdn.net/liminlu0314/article/details/6610069，在此不再赘述。

# 三、GDAL部分类说明

## GDALMajorObject

所有核心类的父类是 GDALMajorObject，它定义了一些操作元数据的属性和方法供子类继承。

元数据：data about date 描述数据的数据，majorobject使用GDALMultiDomainMetadata对象存储元数据。该对象存在一个域名列表和一个元数据内容列表。用户传入域名以及元数据内容已进行数据设置。可序列化为xml。

## GDALDriverManager

管理GDALDriver的类，存储了GDALDriver的数目以及列表。

可以通过序号以及名称来获取处理对象数据对应的driver。

使用一个全局变量static volatile GDALDrvierManager poDM管理，在register数据驱动时，通过GetGDALDriverManage这个全局函数，新建或获取，然后通过RegisterDriver添加对应驱动。

## GDALDriver

数据驱动类，只定义了成员方法，没有成员变量，通过函数指针来实现对于不同的驱动类型采用不同的方式。

有Create方法创建数据集，Delete删除数据集，Rename重命名数据集，CopyFile，CreateCopy

另外定义了其他函数指针，算是接口了。

在注册的时候，根据对象类型创建driver，并设置driver的描述，元数据等，并设置函数指针为对应数据集的静态函数。如poDriver->pfnOpen = JDEMDataset::Open。其中Open方法为JDEMDataset中的静态函数。

## GDALDataSet

一套关联栅格波段，通常来自一个文件。

数据成员包括GDALDriver指针，波段的数目大小，波段列表，引用计数，是否被共享。等信息。以及一个GDALDefaultOverViews对象。提供一个IRasterIO接口给派生类使用，作为读取数据的接口。其他方法以及接口介绍如下：

#include <gdal\_priv.h>

GDALDataset的继承图表：

GDALMajorObject 《 GDALDataset 《 GDALPamDataset

**所有成员列表：**

**公有成员函数**

Virtual ~GDALDataset（）

销毁一个打开的GDALDataset

Int GetRasterXSize（void）

取出栅格宽度，单位像素

Int GetRasterYSize()

取得栅格高度，单位像素

Int GetRasterCount（）

取得该dataset中栅格波段数目

GDAlRasterBand \* GetRasterBand（int）

为一个dataset取得波段对象

Virtual void FlushCache（void）

刷新所有写入的缓存数据到磁盘

Virtual const char \* GetProjectionRef（void）

取得该dataset的投影定义字符

Virtual CPLErr SetProjection (const char \*)

为该dataset 设置投影参考字符

Virtual CPLErr GetGeoTransform (double \*)

取得仿射投影变换系数

Virtual CPLErr SetGeoTransform（double \*）

设置仿射投影变换系数

Virtual CPLErr AddBand （GDALDataType eType， char \*\*papszOptions=NULL）

为dataset增加波段

Virtual void \* GetInternalHandle（const char \*）

取得格式细节的内部有意义的句柄

Virtual GDALDriver \* GetDriver（void）

取得dataset有关系的驱动

Virtual char \*\* GetFileList（void）

取得文件构成数据集

Virtual int GetGCPCount（）

取得GCPs的数目

Virtual const char \* GetGCPProjection（）

得到GCP的输出投影

Virtual const GDAL\_GCP \* GetGCPs（）

取得GCPs

Virtual CPLErr SetGCPs（int nGCPCount, const GDAL\_GCP \* pasGCPList, const char \* pszGCPProjection）

制定GCPs

Virtual CPLErr AdviseRead（int nXOff, int nYOff, int nXSize, int nYSize, int nBufXSize, int nBuffYSize, GDALDataTyoe eDT, int nBandCount, int \* panBandList, char \*\* papszOptions）

向驱动告知即将进行的读请求

Virtual CPLErr CreateMaskBand (int nFlags)

为dataset增加一个掩膜

Virtual GDALAsyncReader \* BeginAsyncReader（int nXOff, int nYOff, int nXSize, int nYsize, void \*pBuf, int nBufXSize, int nBufYSize, GDALDataType eBufType, int nBandCount, int \*panBandMap, int PixelSpace, int nLineSpace, int nBandSpace, char \*\*papszOptions）

设置异步数据请求

Virtual void EndAsyncReader (GDALAsyncReader \*)

结束一个异步请求

CPLerr RasterIO（GDALRWFlag, int, int, int, int, void \*, int, int, GDALDataType, int, int\*, int,int,int）

从多个波段中读写一定区域的图像

Int Reference()

Dataset引用总数加一

Int Dereference ()

Dataset 引用总数减一

GDALAccess GetAcess()

Int GetShared()

返回共享标志

Void MarkAsShared（）

把dataset标记为共享可用

CPLErr BuildOverviews (const char \*, in, int\*, int, int\*,GDALProgressFunc, void \*)

构建栅格预览视图

**静态公有成员函数**

Static GDALDataset\*\* GetOpenDatasets（int \* pnDatasetCount）

取得所有打开的GDAL dataset 的句柄

**保护成员列表**

void RasterInitialize (int, int)

void SetBand (int, GDALRasterBand \*)virtual CPLErr IBuildOverviews (const char \*, int, int \*, int, int \*, GDALProgressFunc, void \*)virtual CPLErr IRasterIO (GDALRWFlag, int, int, int, int, void \*, int, int, GDALDataType, int, int \*, int, int, int)CPLErr BlockBasedRasterIO (GDALRWFlag, int, int, int, int, void \*, int, int, GDALDataType, int, int \*, int, int, int)void BlockBasedFlushCache ()

**保护属性**

GDALDriver \* poDriver

GDALAccess eAccess

int nRasterXSize

int nRasterYSize

int nBands

GDALRasterBand \*\* papoBands

int bForceCachedIO

int nRefCount

int bShared

GDALDefaultOverviews oOvManager

**友元**

class GDALDriver

class GDALDefaultOverviews

class GDALProxyDataset

class GDALRasterBand

GDALDatasetH GDALOpen（const char \*, GDALAccess）

以GDALDataset的格式打开栅格文件

GDALDatasetH GDALOpenShared（const char \*, GDALAccess）

以GDALDataset的格式打开栅格文件

**详细说明**

一套关联栅格波段，通常来自一个文件。

一个dataset包封了一个或多个栅格波段

更多详细讨在GDAL Data Model

用GDALOpen（）或GDALOpenShared（）来创建GDALDataset 通过打开一个有名字的的文件

或者用GDALDriver::Create 或 GDALDriver::CreateCopy()来创建一个新的dataset

**构造函数和析构函数文档**

GDALDataset::~GDALDataset ( ) [virtual]

析构一个打开的GDALDataset。

这是一个可接受的方法来关闭一个GDAL Dataset 并且重新分配所有关联的资源。

错误的单词？Equivelent（Equivalent）

与之等价的是c 中的GDALClose（）。除了GDALClose（）首先先减少Reference总数，然后当总数变为0时再关闭。

对于windows用户，使用在Dataset对象上删除操作是不可被使用的，因为分配和释放空间跨越了模块的界限这个已知的issue. 调用 GDALClose（）是一个更好的选择.

参考 GDALMajorObject::GetDescription（）

**成员函数文档**

CPLErr GDALDataset::AddBand ( GDALDataType eType, char \*\* papszOptions = NULL ) [virtual]

增加一个波段到dataset中.

如果基本格式支持这种操作的话，这个方法将会在dataset中增加一个新的波段。大部分的格式不支持这种操作。

注意新的GDALRasterBand（）不会被返回。 但是在这个操作成功之后可以通过调用函数GDALDataset::GetRasterBand（GDALDataset::GetRasterCount（）-1）来取得，因为最新的波段总是在最后一个波段中。

参数：

eType 新波段中像素的数据类型

papszOptions Name=value选项字符串的列表。 被支持的选项是格式细节。NULL作为默认值也可通过。

返回值：

CE\_None 成功

CE\_Failure 失败

CPLErr GDALDataset::AdviseRead ( int nXOff,

int nYOff,

int nXSize,

int nYSize,

int nBufXSize,

int nBufYSize,

GDALDataType eDT,

int nBandCount,

int \* panBandMap,

char \*\* papszOptions

) [virtual]

告知驱动器即将进行的读请求。

一些GDAL 驱动会更有效率的工作，如果它们事先知道即将进行的读请求的设置。AdviseRead（）方法允许程序通知驱动哪个范围和波段会被读取。

一些驱动会忽略调adviseRead（）的调用，但是它会明显的增加读取熟读在另外一些驱动上。

参数：

nXOff 被读取的波段区域距离左上角的偏离值（单位：像素）。从左边以0开始

nYOff 被读取的波段区域距离左上角的偏离值（单位：行）。 从顶部以0开始。

nXSize 被读取的波段区域宽度（单位：像素）

nYSize 被读取的波段区域高度（单位：行）

nBufXSize 缓冲区图像的宽度

nBufYSize 缓冲区图像的高度

eBufType

nBandCount 将要被读写的波段数目

panBandMap nBandCount个将被读写的波段的列表。注意波段数目至少为1。将会是NULL来选择第一个nBandCount波段

papszOptions 一个name=value的关于特殊控制选项的列表，正常来说为NULL

返回值：

CE\_Failure 如果请求无效

CE\_None 如果见效了或被忽略了

参考GDALRasterBand::AdviseRead(), 和GetRasterBand().

GDALAsyncReader \* GDALDataset::BeginAsyncReader ( int nXOff,

int nYOff,

int nXSize,

int nYSize,

void \* pBuf,

int nBufXSize,

int nBufYSize,

GDALDataType eBufType,

int nBandCount,

int \* panBandMap,

int nPixelSpace,

int nLineSpace,

int nBandSpace,

char \*\* papszOptions

) [virtual]

建立异步数据请求。

这个方法在dataset中指定的窗口建立了一个异步的栅格读取请求到指定的缓冲区。参数为了windowing，缓冲区大小，缓冲类型和缓冲组织和GDALDataset::RasterIO（）的这些类似；

这个调用只会打开请求，填入缓冲区被完成通过调用GetNextUpdateRegion（）在返回GDALAsyncReader会话对象

一旦创建对话的所有进程完成时，或者没有进一步动作的需求时，GDALAsyncReader 对象将会通过GDALDataset::EndAsyncReader（）方法自动被销毁

注意数据缓冲区（pData）在会话的生命周期中可能会潜在的持续更新，但是当对话（GDALAsyncReader）被EndAsyncReader结束时，它不会被重新分配。它会被那个点的程序重新分配。

有关GDAL中异步IO的补充信息可在<http://trac.osgeo.org/gdal/wiki/rfc24_progressive_data_support>上找到。

这个方法和C的GDALBeginAsyncReader（）一样。

CPLErr GDALDataset::BuildOverviews ( const char \* pszResampling,

int nOverviews,

int \* panOverviewList,

int nListBands,

int \* panBandList,

GDALProgressFunc pfnProgress,

void \* pProgressData

)

建立栅格预览图。

如果指定的dataset不支持这个操作，返回值将是CE\_Failure, CPLGtLastErrorNo() 将会返回CPLE\_NotSupported.

这个方法和C的函数GDALBuildOverviews（）相同

CPLErr GDALDataset::CreateMaskBand ( int nFlags ) [virtual]

对dataset增加一个掩膜。

CreateMaskBand（）方法的默认执行是在与.ovr操作工具类似的规则的基础上使用GDALDefualtOverviews对象执行。 一个带有.msk扩展的TIFF文件会被创建成与原始文件相同的基本名称，并且会有一个波段。如果可能的话掩膜图像会是与原始图像有着相同的block size的压缩图像

int GDALDataset::Dereference ( )

从dataset引用计数中减一

实例化之后引用计数为一。一般来说引用计数减到零时这个dataset已经被安全的删除（关闭）了。

这个方法和C的GDALDereferenceDataset（）函数相同。

void GDALDataset::EndAsyncReader ( GDALAsyncReader \* poARIO ) [virtual]

结束异步请求。

这个方法将销毁一个异步IO请求，并恢复与之相关联的资源。

这个方法和C的GDALEndAsyncReader（）相同。

void GDALDataset::FlushCache ( void ) [virtual]

刷新所有写入的缓存到磁盘。

任何栅格（或其他GDAL）数据通过GDAL的调用被写入，但是缓冲区会被内部机制写入到磁盘。

使用这个方法不会阻止调用GDALClose（）来正常关闭一个dataset，并且确保重要的数据不会被FlushCache分配地址写入发哦文件中。

这个方法和C下面的函数GDALFlushCache（）相同。

在GDALPamDataset中执行。

GDALDriver \* GDALDataset::GetDriver ( void ) [virtual]

取得dataset相关联的驱动。

这个方法和C的GDALGetDatasetDriver（）函数相同。

char \*\* GDALDataset::GetFileList ( void ) [virtual]

取得构成dataset的文件。

返回被认为是构成dataset的文件列表。如果返回一个空的文件列表，意味着认为没有本地文件系统的文件和该dataset相关联（例如一个虚拟的dataset）。这个返回的文件列表是归调用者拥有的，并可以被CSLDestory（）所销毁。

返回的文件名通常是相对路径或绝对路径，这取决于打开dataset时使用的路径。

这个方法和C的GDALGetFileList（）函数相同。

int GDALDataset::GetGCPCount ( ) [virtual]

取得GCP的数目。

这个方法和C的GDALGetGCPCount（）相同。

const char \* GDALDataset::GetGCPProjection ( ) [virtual]

取得GCP输出投影。

这个方法和C的 GDALGetGCPProjection（）相同。

投影字符串遵循GetProjectionRef（）的标准规则。

const GDAL\_GCP \* GDALDataset::GetGCPs ( ) [virtual]

取得GCPs。

这个方法和C的GDALDataset::GetGCPs（）相同。

CPLErr GDALDataset::GetGeoTransform ( double \* padfTransform ) [virtual]

取得仿射变换系数。

取得在pixel/line(P, L)栅格空间和投影坐标空间之间变换的系数。

Xp = padfTransform[0] + P\*padfTransform[1] + L\*padfTransform[2];

Yp = padfTransform[3] + P\*padfTransform[4] + L\*padfTransform[5];

在北朝上的图像里，padfTransform[1]是像素宽，padfTransform[5]是像素高。左上像素的左上角就是这个坐标（padfTransform[0]，padfTransform[3]）。

默认的转换是（0,1,0,0,0,1）即使当返回CE\_Failure 错误也会被返回，，例如格式上不支持转换到投影坐标系。

注意：GetGeoTransfrom （）不能够表达处理多种从OGC Grid Coverage pixel/line 到投影变换的方案。最后这个方法将会被让位于更普遍的scheme。

这个方法和C的GDALGetGeoTransfrom（）相同。

void \* GDALDataset::GetInternalHandle ( const char \* ) [virtual]

取得一个格式详尽的内部有意义的句柄。

这个方法和C的GDALGetGeoInternalHandle（）相同。

GDALDataset \*\* GDALDataset::GetOpenDatasets ( int \* pnCount ) [static]

取得所有打开的GDAL dataset 的句柄。

这个方法和C的GDAL GetOpenDatasets（）相同。

const char \* GDALDataset::GetProjectionRef ( void ) [virtual]

取得这个dataset的投影定义字符串。

这个方法和C的GDAL GetProjectionRef（）相同。

这个返回的字符串使用Open GIS WKT format定义了该图像的投影坐标系。它也适合于和OGRSpatialReference类使用。

当投影定义不可用时，将返回一个空的字符串（但不是NULL）。

GDALRasterBand \* GDALDataset::GetRasterBand ( int nBandId )

取得dataset的一个波段对象。

相当于C的GDALGetRasterBand()

int GDALDataset::GetRasterCount ( void )

取得dataset的栅格波段数量。

和C的函数GDALGetRasterCount()相同

int GDALDataset::GetRasterXSize ( void )

取得栅格宽度（单位像素）

等同于C的GDAL GetRasterXSize（）

int GDALDataset::GetRasterYSize ( void )

取得栅格高度（单位像素）

等同于C的GDAL GetRasterYSize（）

int GDALDataset::GetShared ( )

返回共享标志

CPLErr GDALDataset::RasterIO ( GDALRWFlag eRWFlag,

int nXOff,

int nYOff,

int nXSize,

int nYSize,

void \* pData,

int nBufXSize,

int nBufYSize,

GDALDataType eBufType,

int nBandCount,

int \* panBandMap,

int nPixelSpace,

int nLineSpace,

int nBandSpace

)

从多个波段中读写一个区域的图像数据。

这个方法允许从dataset中把一个或多个GDAL栅格波段的区域数据读取到缓冲区，或者把缓冲区数据写入到GDAL栅格波段的一个区域中。如果缓冲区的数据类型（eBufType）和GDAL栅格波段的数据类型不同的话，它会自动处理数据类型转换。如果缓冲区大小（nBufXSize x nBufYSize）和被存取的区域大小（nXSize x nYSize）不同的话，这个方法同样也会处理图像的抽取和复制。

nPixelSpace, nLineSpace 和 nBandSpace 参数允许从多种缓冲区组织结构中读和写。

为了最高精确性的全分辨率数据存取，在“区块分界线”（block boundaries）的读写被GetBlockSize（）的返回值，也可以使用ReadBlock（）和WriteBlock（）方法。

这个方法和C GDALDatasetRasterIO（）相同

int GDALDataset::Reference ( )

dataset资源计数加一。

实例化之后资源数量为一。

这个方法和C GDALReferenceDataset（）相同。

CPLErr GDALDataset::SetGCPs ( int nGCPCount,

const GDAL\_GCP \* pasGCPList,

const char \* pszGCPProjection

) [virtual]

分配GCPs。

这个方法和C的GDALSetGCPs（）相同。

这个方法给该Dataset和它们的坐标系统分配已经通过的设置。内部拷贝是有坐标系统和一个点的列表构成，如果适当调用者将会遗留下重新分配这些参数的责任。

大部分的格式不支持GCPs的设置，即使格式能处理GCPs。这些格式将会返回CE\_Failure。

LErr GDALDataset::SetGeoTransform ( double \* ) [virtual]

设置仿射变换参数。

参考GetGeoTransform()获得更多细节有关padfTransform系数。

这个方法和C GDALSetGeoTransform（）一样。

CPLErr GDALDataset::SetProjection ( const char \* ) [virtual]

为dataset设置投影参考字符串

这个在OGC WKT或PROJ.4 格式。因为错误的投影字符串，dataset不可写或者dataset不支持标出投影，会导致一个错误出现。许多格式不支持写入投影。

这个方法和C的GDALSetProjection（）相同。

**友类和相关函数文档**

GDALDatasetH GDALOpen ( const char \* pszFilename,

GDALAccess eAccess

) [friend]

用GDALDataset方式打开栅格文件。

这个函数将尝试着打开通过的文件或者轮流通过援引每个已注册的GDALDriver的打开方法来打开虚拟dataset名字。第一次成功的打开将会返回一个dataset。如果所有的驱动都失败了，将会返回一个NULL。

一些建议：

如果你用GA\_Update 存取方式打开一个dataset对象，不建议在同一个基础的文件上打开一个新的dataset。

返回的dataset每次仅仅被单线程存取。如果你想通过不同线程打开dataset，你必须增加必要的代码（mutexes等等）来避免同时使用这个对象。（一些驱动像GeoTIFF支持内态变量，每次更新时一个新的区域被读取，这样防止同时使用。）

GDALDatasetH GDALOpenShared ( const char \* pszFilename,

GDALAccess eAccess

) [friend]

用GDALDataset方式打开栅格文件。

这个函数工作时和GDALOpen（）一样，但是允许为其他的调用者dataset分享GDALDataset句柄。

尤其，GDALOpenShared（）将首先考虑当前打开和共享的GDALDataset的列表，并且如果GetDescription（）名称和pszFilename精确的匹配的话，将会被引用和返回。

开始于GDAL1.6.0, 如果GDALOpenShared（）被两个不同的线程在一样的pszFilename上调用，一个不同的differentGDALDataset对象将被返回，通过不同的线程使用同一个dataset是不安全的，除非用户在代码中明确使用了互斥。

## GDALRasterBand

一个单一的栅格数据波段

存在一个GDALRasterBlock友元

存在一个GDALDataset对象（GDAL貌似很喜欢把外层对象的指针存在分类的对象里面。如在学生的信息里面存储班级的信息， 班级的信息里面要有一个年级的指针，当然，班级类是学生类的友元，年级类是班级类的友元。难道是学生犯错了，然后由班级处置的意思？）

Rasterband中定义了比rasterdataset更细致的数据类型。

# 四、GDAL实用工具介绍

[gdalinfo](http://wiki.westgis.ac.cn/index.php/Gdalinfo).exe 显示GDAL支持的各种栅格文件的信息。

[gdal\_translate](http://wiki.westgis.ac.cn/index.php/Gdal_translate).exe 在不同的格式间进行转换。同时，潜在的执行了一些切割、重采样和使像素比例变化的任务。

[gdalwarp](http://wiki.westgis.ac.cn/index.php/Gdalwarp).exe 投影转换和投影绑定。同时也可以进行图像镶嵌。这个程序可以重新投影所支持的投影，而且如果图像("raw" with)控制信息也可以把GCPs 和图像存储在一起

[gdaltindex](http://wiki.westgis.ac.cn/index.php/Gdaltindex).exe 创建栅格索引的SHP 文件。它能为每个栅格数据建立一个记录：一个包含栅格名称的属性，以及这个栅格的外边界所组成的多边形。

[gdal\_contour](http://wiki.westgis.ac.cn/index.php/Gdal_contour).exe 从一个栅格DEM生成矢量的等值线。

[gdaltransform](http://wiki.westgis.ac.cn/index.php/Gdaltransform).exe 与PROJ相似。进行投投影转换。

[nearblack](http://wiki.westgis.ac.cn/index.php/Nearblack).exe 将栅格中接处黑\白的像元置成黑色或白色。这可以用来修补压缩丢失信息而造成的不准确，以便于在镶嵌时将其视为透明。

[gdal\_grid](http://wiki.westgis.ac.cn/index.php?title=Gdal_grid&action=edit&redlink=1).exe 从分散的点生成栅格，有插值的意思。

[ogrinfo](http://wiki.westgis.ac.cn/index.php/Ogrinfo).exe 显示OGR支持的矢量数据的信息

[ogr2ogr](http://wiki.westgis.ac.cn/index.php/Ogr2ogr).exe 进行不同简单矢量格式转换。

[gdaladdo](http://wiki.westgis.ac.cn/index.php/Gdaladdo) - 在文件内建立金字塔，提高文件打开和浏览速度.

[gdalbuildvrt](http://wiki.westgis.ac.cn/index.php?title=Gdalbuildvrt&action=edit&redlink=1) - Build a VRT from a list of datasets.

[gdaldem](http://wiki.westgis.ac.cn/index.php/Gdaldem) - DEM分析与可视化工具,可进行坡度、坡向和山影分析，以及高程分色渲染功能.

[rgb2pct](http://wiki.westgis.ac.cn/index.php?title=Rgb2pct&action=edit&redlink=1).py - 将24bit的RGB图像转换为8bit颜色表图像.

[pct2rgb](http://wiki.westgis.ac.cn/index.php?title=Pct2rgb&action=edit&redlink=1).py - 将8bit颜色表图像转换为24bit的RGB图像.

[gdal\_merge](http://wiki.westgis.ac.cn/index.php/Gdal_merge).py - Build a quick mosaic from a set of images.

[gdal2tiles](http://wiki.westgis.ac.cn/index.php?title=Gdal2tiles&action=edit&redlink=1).py - 生成 TMS瓦片结构，可用Google Earth(KML)或一般web浏览器进行浏览.

[gdal\_rasterize](http://wiki.westgis.ac.cn/index.php?title=Gdal_rasterize&action=edit&redlink=1) - 将矢量数据栅格化.

[gdal\_retile](http://wiki.westgis.ac.cn/index.php?title=Gdal_retile&action=edit&redlink=1).py - Retiles a set of tiles and/or build tiled pyramid levels.

[gdal\_proximity](http://wiki.westgis.ac.cn/index.php?title=Gdal_proximity&action=edit&redlink=1).py - Compute a raster proximity map.

[gdal\_polygonize](http://wiki.westgis.ac.cn/index.php?title=Gdal_polygonize&action=edit&redlink=1).py - Generate polygons from raster.

[gdal\_sieve](http://wiki.westgis.ac.cn/index.php?title=Gdal_sieve&action=edit&redlink=1).py - Raster Sieve filter.

[gdal\_fillnodata](http://wiki.westgis.ac.cn/index.php?title=Gdal_fillnodata&action=edit&redlink=1).py - Interpolate in nodata regions.

gdal-config - Get options required to build software using GDAL.

# 五、开源栅格/矢量空间数据转换库-GDAL/OGR

## 架构

GDAL(Geospatial Data Abstraction Library)是一个在X/MIT许可协议下的开源栅格空间数据转换库。它利用抽象数据模型来表达所支持的各种文件格式。它还有一系列命令行工具来进行数据转换和处理。

OGR是GDAL项目的一个分支，功能与GDAL类似，只不过它提供对矢量数据的支持。

有很多著名的GIS类产品都使用了GDAL/OGR库，包括ESRI的ArgGIS 9.2，Google Earth和跨平台的GRASS GIS系统。

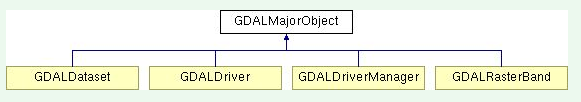
利用GDAL/OGR库，可以使基于Linux的地理空间数据管理系统提供对矢量和栅格文件数据的支持。

## 1、GDAL

GDAL提供对多种栅格数据的支持，包括Arc/Info ASCII Grid(asc)，GeoTiff (tiff)，Erdas Imagine Images(img)，ASCII DEM(dem) 等格式。

GDAL使用抽象数据模型(abstract data model)来解析它所支持的数据格式，抽象数据模型包括数据集(dataset)，坐标系统，仿射地理坐标转换(Affine Geo Transform)， 大地控制点(GCPs)， 元数据(Metadata)，栅格波段(Raster Band)，颜色表(Color Table)，子数据集域(Subdatasets Domain)，图像结构域(Image\_Structure Domain)，XML域(XML:Domains)。

GDAL的核心类结构设计如图所示：



其中的类说明如下：

GDALMajorObject类：带有元数据的对象。

GDALDdataset类：通常是从一个栅格文件中提取的相关联的栅格波段集合和这些波段的元数据；GDALDdataset也负责所有栅格波段的地理坐标转换(georeferencing transform)和坐标系定义。

GDALDriver类：文件格式驱动类，GDAL会为每一个所支持的文件格式创建一个该类的实体，来管理该文件格式。

GDALDriverManager类：文件格式驱动管理类，用来管理GDALDriver类。

## 2、OGR

OGR提供对矢量数据格式的读写支持，它所支持的文件格式包括：ESRI Shapefiles， S-57， SDTS， PostGIS，Oracle Spatial， Mapinfo mid/mif ， Mapinfo TAB。

### 1)OGR体系结构

OGR包括如下几部分：

Geometry：类Geometry (包括OGRGeometry等类)封装了OpenGIS的矢量数据模型，并提供了一些几何操作，WKB(Well Knows Binary)和WKT(Well Known Text)格式之间的相互转换，以及空间参考系统(投影)。

Spatial Reference：类OGRSpatialReference封装了投影和基准面的定义。

Feature：类OGRFeature封装了一个完整feature的定义，一个完整的feature包括一个geometry和geometry的一系列属性。

Feature Definition：类OGRFeatureDefn里面封装了feature的属性，类型、名称及其默认的空间参考系统等。一个OGRFeatureDefn对象通常与一个层(layer)对应。

Layer：类OGRLayer是一个抽象基类，表示数据源类OGRDataSource里面的一层要素(feature)。

Data Source：类OGRDataSource是一个抽象基类，表示含有OGRLayer对象的一个文件或一个数据库。

Drivers：类OGRSFDriver对应于每一个所支持的矢量文件格式。类OGRSFDriver由类OGRSFDriverRegistrar来注册和管理。

### 2) OGR的Geometry模型

OGR的Geometry模型是建立在OpenGIS的简单要素数据模型之上的。如下图所示：

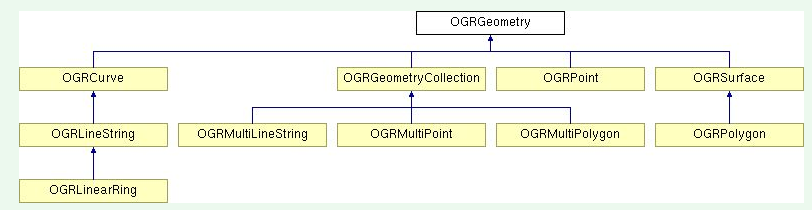


图-OGR的Geometry模型关系图

OpenGIS的简单要素数据模型，其关系图如下所示：

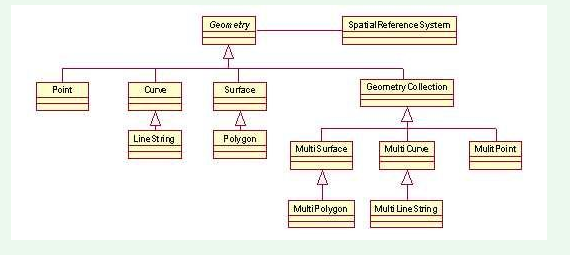


图-OpenGIS的简单要素数据模型

由上面两图的对比，可以清楚的看到，OGR的Geometry模型是严格遵循OpenGIS的简单要素数据规范的。OGR的Geometry模型不仅在继承体系上与OpenGIS的简单要素数据模型一致，在函数接口上也向其靠拢，从基本的获取Geometry对象信息的方法如Dimension ( )、GeometryType ( )、SRID ( )、Envelope( )、AsText( )、Boundary( )等到判定空间未知关系的方法如Equals(anotherGeometry:Geometry)、Disjoint(anotherGeometry:Geometry)、

Intersects(anotherGeometry:Geometry)、Touches(anotherGeometry:Geometry)等都是符合其标准的。

### 3）OGR的API使用范例

以下C++示例代码展示了使用OGR的API来读取矢量数据。

#include "ogrsf\_frmts.h"

int main()

{

//注册所有的文件格式驱动

OGRRegisterAll();

//打开point.shp文件

OGRDataSource \*poDS = OGRSFDriverRegistrar::Open( "point.shp", FALSE );

//获取点层

OGRLayer \*poLayer = poDS->GetLayerByName( "point" );

OGRFeature \*poFeature;

//重置该层，确保从层的开始读取数据

poLayer->ResetReading();

while ( (poFeature = poLayer->GetNextFeature()) != NULL )

{

//获取该层的属性信息

OGRFeatureDefn \*poFDefn = poLayer->GetLayerDefn();

int iField;

for ( iField = 0; iField < poFDefn->GetFieldCount(); iField++ )

{

//获取某一个字段的信息

OGRFieldDefn \*poFieldDefn = poFDefn->GetFieldDefn( iField ); if( poFieldDefn->GetType() == OFTInteger )

printf( "%d,", poFeature->GetFieldAsInteger( iField ) );

}

OGRGeometry \*poGeometry;

//获取feature

poGeometry = poFeature->GetGeometryRef();

//用wkbFlatten宏把wkbPoint25D类型转换为wkbPoint类型

if( poGeometry != NULL && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )

{

OGRPoint \*poPoint = (OGRPoint \*) poGeometry;

printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );

}

else

printf( "no point geometry\n" );

//销毁feature

OGRFeature::DestroyFeature( poFeature );

}

//销毁数据源，以便关闭矢量文件

OGRDataSource::DestroyDataSource( poDS );

}

# 六、GDAL源码剖析之Swig编译和帮助文档生成

## 一、Swig编译

### 1、Swig介绍

SWIG全称是Simplified Wrapper and Interface Generator，官方网站：http://www.swig.org/。SWIG是个帮助使用C或者C++编写的软件能与其它各种高级编程语言进行嵌入联接的开发工具。SWIG能应用于各种不同类型的语言包括常用脚本编译语言例如Perl, PHP, Python, Tcl, Ruby and PHP。支持语言列表中也包括非脚本编译语言，例如C#, Common Lisp (CLISP, Allegro CL, CFFI, UFFI), Java, Modula-3, OCAML以及R，甚至是编译器或者汇编的计划应用（Guile, MzScheme, Chicken）。SWIG普遍应用于创建高级语言解析或汇编程序环境，用户接口，作为一种用来测试C/C++或进行原型设计的工具。SWIG还能够导出XML或Lisp s-expressions格式的解析树。SWIG可以被自由使用，发布，修改用于商业或非商业中。[摘自SWIG官网http://www.swig.org/translations/chinese/index.html]。

下载安装Swig的时候注意下载Swigwin（我下的是swigwin-2.0.4.zip），不要下载源代码，否则不能在windwos下用。下载后解压，将swigwin-2.0.4的解压目录也添加到环境变量Path中去，否则会出现一些些该配置文件的麻烦。检验swig是否成功设置到环境变量Path中的最简单的方式就是在运行中输入swig后回车，如果提示windows找不到swig，那么说明没有设置成功；如果出现一个黑屏一闪而过，那么说明你设置成功了。

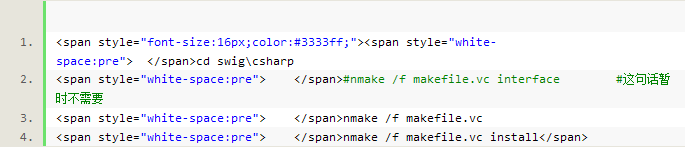
### 2、编译C#版本GDAL

首先，打开nmake.opt文件，找到SWIG=swig.exe这一句，假如没有将swig的目录添加到环境变量中，那么将这句后面的swig.exe修改为swig.exe的全路径，如F:\Work\3rdPart\swigwin-2.0.4\swig.exe。如果设置了环境变量，那么就不需要进行修改了。

然后按照第二篇中的使用cmd命令编译GDAL的方式来进行编译，打开“Visual Studio 2008命令提示”并定位到GDAL源代码目录，然后依次执行下面三行命令：



执行完之后会在GDAL\_HOME的目录下生成编译好的dll和exe文件以及include、lib等文件夹。接下来，使用cd命令，进入swig\csharp文件夹中：



执行完上面两句后，会在csharp文件夹下生成8个dll文件，并将这8个dll文件拷贝到GDAL输出目录下的csharp文件夹中。然后就可以在C#工程中引用这几个dll了，需要注意的是，在使用这几个dll的时候需要将gdal18.dll以及其依赖的其他dll都要拷贝到同一个目录中才能正常运行。

### 3、编译Java版本GDAL

首先需要JDK环境，没有的可以下载安装一个。然后打开nmake.opt文件，找到大概86行左右的位置：



将后面的路径修改为JDK的目录后保存。在打开的命令行中使用cd命令定位到swig目录后，输入下面的



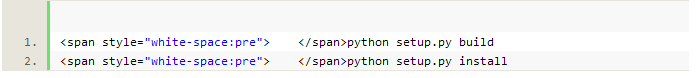
### 4、编译Python版本GDAL

首先在编译Python版本之前，确保自己的电脑中安装了Python，相信大家都装ArcGIS了吧，那么恭喜你，你不用安装Python了，因为在安装ArcGIS的时候必须要安装Python的，同样，将Python的bin目录添加到环境变量Path中去。确保这步完成之后，接下来就是编译Python版本的GDAL。

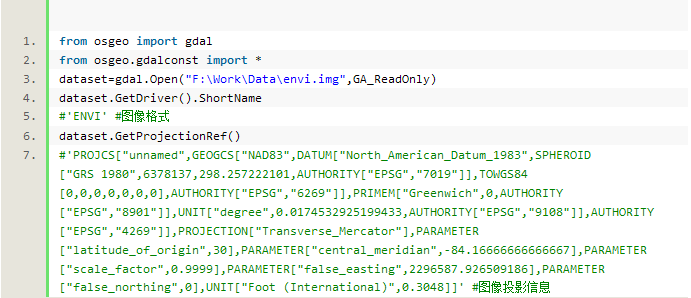
打开nmake.opt文件，找到大概76行左右的：



将后面的路径修改为Python的安装路径后保存。使用cd命令将命令行定位到swig\python目录后，依次输



执行完上述命令后，会在python的site-packages目录看到多了gdal和ogr的文件以及一个osgeo的文件夹。我本机的目录是C:\Python26\Lib\site-packages。然后将编译出来的gdal18.dll以及它依赖的所有的dll都拷贝到site-packages文件夹中的osgeo文件夹中，最后就可以在Python中使用imort gdal来使用GDAL了。测试Python版本的gdal是否安装成功，可以使用下面命令：



注意，上面命令中开头为#号的表示的是Python输出的。如果能输出上面的信息，那么GDAL for Python就安装成功了。

## 二、开发帮助文档生成

### 1、Doxygen介绍

以下内容摘自维基百科，地址http://zh.wikipedia.org/wiki/Doxygen。

Doxygen 是一个 C++, C, Java, Objective-C、Python、IDL (CORBA 和 Microsoft flavors)、Fortran、VHDL、PHP、C#和D语言的文档生成器。可以运行在大多数类Unix系统，以及Mac OS X操作系统和Microsoft Windows 。 初始版本的Doxygen借鉴了一些老版本DOC++的代码；随后，Doxygen源代码由Dimitri van Heesch重写。

Doxygen是一个编写软件参考文档的工具。 该文档是直接写在代码中，因此比较容易保持更新。 Doxygen 可以交叉引用文档和代码，使文件的读者可以很容易地引用实际的代码。

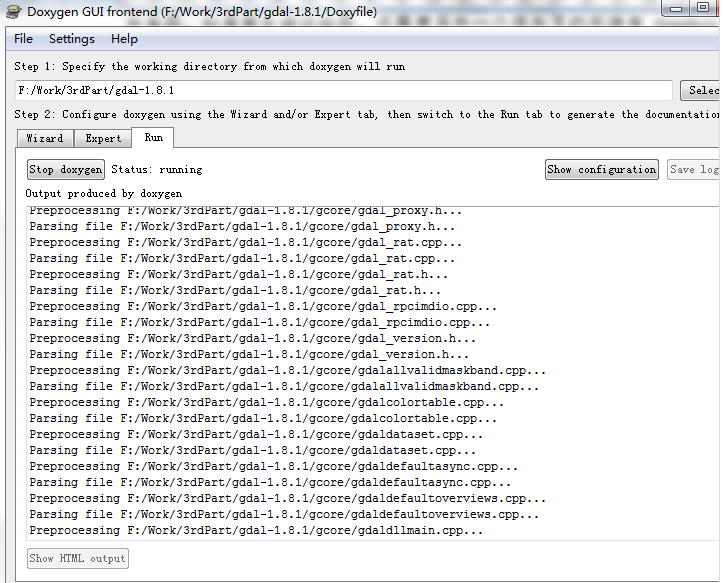
KDE 使用Doxygen作为其部分文档且 KDevelop 具有内置的支持。 Doxygen的发布遵守GNU General Public License，并且是自由软件。

需要注意的是在使用doxygen的时候，会自动生成一些类图，以及函数调用关系图，如果要生成这些图，还需要另外一个很牛叉的开源库Graphviz。Graphviz （Graph Visualization Software的缩写）是一个由AT&T实验室启动的开源工具包，用于绘制DOT语言脚本描述的图形。它也提供了供其它软件使用的库。Graphviz是一个自由软件，其授权为Common Public License。其Mac版本曾经获得2004年的苹果设计奖。更多关于Graphviz的介绍参考其官方网站：http://www.graphviz.org/。

### 2、生成GDAL开发帮助文档

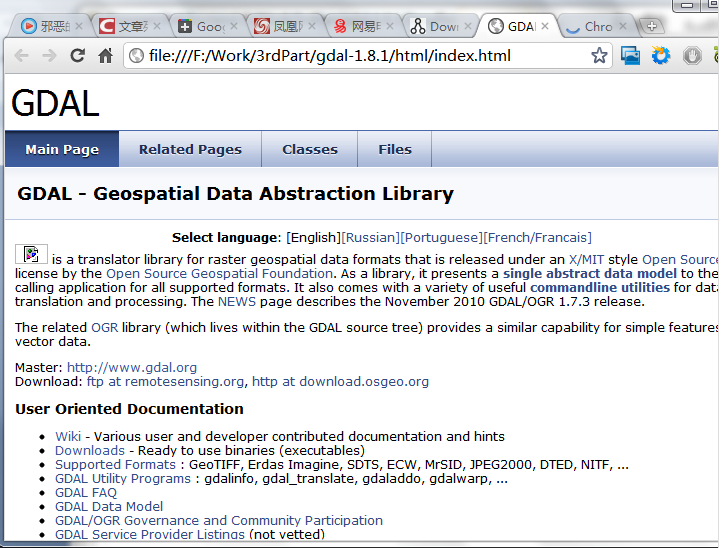
首先下载安装Doxygen和Graphviz，安装完之后最好将Doxygen和Graphviz的bin目录添加到系统环境变量Path中去。负责要设置一些参数，比较麻烦，还是放到Path中方便。

安装完Doxygen后，会在开始菜单中有个doxywizard.exe的程序，打开，然后在File->Open…菜单中选择GDAL源代码目录下的Doxyfile文件，然后，切换到Run标签，点击Run Doxygen按钮，接下来就会自动提取源代码中的注释生成一份gdal的帮助文档，默认的输出目录是GDAL目录下的html目录。等待生成结束后，点击左下角的Show Html Output后会打开生成的帮助文档。如下图：



再次，基本上GDAL的帮助文档生成完成，打开后会发现和GDAL的官方网站一模一样。但是还有个问题，这里只是生成的是GDAL的帮助文档，没有OGR的帮助文档，同样按照上面的步骤，打开GDAL目录下的OGR文件夹下的Doxyfile。然后点击生成，生成的目录默认为ogr文件夹下有个html文件夹，将该文件夹重命名为ogr，然后整个拷贝到上一层的html中，同时将GDAL目录中的doc文件夹中的文件除dox文件以外的文件全部拷贝到html文件夹中。

最后，还记得编译GDAL后生成的html文件夹吗？将这两个文件夹中的内容进行合并，然后你就得到了一份完完整整的GDAL的帮助文档。首页是html文件夹下的index.html。如下图：



### 3、Doxygen脚本配置以及代码注释书写规范

对于Doxygen的脚本配置，可以打开doxywizard后，在Wizard和Expert 标签中在每一项中，鼠标移动到上面，在左下角的试图区域会显示很详细的帮助信息（英文的，不是很难），这里有篇说明文档《 Doxygen配置使用指南》，http://read.pudn.com/downloads151/doc/653112/Doxygen\_Using\_Manual.pdf。文档中包含了所有的东西，很不错的东西。

# 七、GDAL源码剖析之命令行程序说明

## 一、GDAL工具通用命令

下面的工具主要参考的GDAL官方网站中提供的帮助文档说明，此外还有我的一些经验，GDAL官方具体地址为：http://gdal.org/gdal\_utilities.html。

在所有的GDAL工具集中都会支持下面的通用命令行参数，其形式一般是以两个减号（--）开始，下面详细介绍：

1. –version

输出GDAL的版本信息，即版本号。

2. --formats

输出GDAL支持的所有图像格式说明。包括只读和读写。格式支持描述如下：“ro”是只读驱动；“rw”是读写驱动（比如支持CreateCopy方法）；“rw+”是读写和更新驱动（比如支持Create方法），支持所有的读写更新操作。

3. --format format

输出GDAL单个格式驱动的细节信息。格式名需要是在--formats 后列出所要输出的格式名。比如GTiff，HFA,PCISK等。

4. --optfile file

读取指定名称的文件并把其中的内容当成参数传入命令行列表。如果行首以#开头的行将被忽略。多字组成的参数（即中间有空格隔开的参数）需要用双引号来保正其为单一的参数。

5. --config key value

设置配置，把指定键设置为某个值，从而不必把他们设置为环境变量。一些命令参数键是GDAL\_CACHEMAX(用于缓存的内存有多少M)以及GDAL\_DATA（gdal的数据路径）。比如在GDAL1.8之后，经常会发现打不开中文路径的文件，那么可以用这个来设置，具体为“—configGDAL\_FILENAME\_IS\_UTF8 NO”。同时对于每一种驱动都会有各自的配置，具体参考各个驱动的说明。更多的配置选项参考该网址：http://trac.osgeo.org/gdal/wiki/ConfigOptions。

6. --debug value

控制调试信息的打印输出。ON值表示允许调试信息输出，OFF值表示不要输出调试信息。

7. --help-general

输出各个工具的命令行参数帮助信息。不同的命令输出的内容不同。

下面还有一些通用的命令，是用来创建文件来使用的。创建不同的格式需要的参数都是不相同的，尤其是在特殊的情况下，比如创建的Erdas的img格式需不需要使用压缩等特殊的需求。这些参数一般使用一个减号（-）开始。下面对这些参数进行一个简单的说明。

8. -of format

选择要创建新的文件的格式。这个格式被指定为类似GTiff（GeoTIFF格式）或者HFA（ERDAS格式）。所有的支持格式列表可以用--formats 参数列出来。但是只有格式列表“(rw)”可以被写入和创建。许多工具如果没有指定，默认是创建GeoTIFF格式的文件。文件扩展名不会自动添加，如果没有指定文件名的后缀名，gdal一般不会添加任何扩展名。各个工具的命令行参数帮助信息。不同的命令输出的内容不同。

9. -co NAME=VALUE

创建文件选项，许多格式会有一个或者更多的创建参数来控制文件创建的细节。比如GeoTIFF或者Erdas的img格式可以用创建参数控制压缩，或者控制是否用分片还是分带来进行存储。

可以使用的创建参数根据格式驱动不同而不同。而一些简单的格式根本就没有创建参数。虽然某个格式可以用"--format <format>"参数列出所有可用的参数列表，但是更详细的信息可以在格式介绍网页中查到。对于不同的文件格式，请参考对应文件格式说明网页。

10. -a\_srs SRS

指定输出文件的投影信息（坐标系统）。输出各个工具的命令行参数帮助信息。不同的命令输出的内容不同。有几个工具(如gdal\_translate、gdalwarp)可以在命令行中通过类似-a\_srs(分配输出SRS)、-s\_srs(源SRS)、-t\_srs(目标SRS)来指定坐标系统。这些工具允许以一系列格式定义坐标系统（SRS就是空间参考系统spatial reference system）。SRS通常可以使用下面几种方式来指定：

NAD27/NAD83/WGS84/WGS72:这些常见的地理坐标系统可以通过名字来直接使用。

EPSG:n：坐标系统（投影或者地理坐标）可以通过EPSG码来选择。例如EPSG 27700是英国国家网格。更多的EPSG坐标系统可以在GDAL数据文件gcs.csv和pcs.csv中找到（位于GDAL目录中的data文件夹中）。

PROJ.4定义：一个PROJ4定义字符串可以用作坐标系统定义。例如“+proj=utm +zone=11 +datum=WGS84”。注意在命令行中要保持Proj4字符串在一起作为一个单独的参数（一般用双引号引起来）。

OpenGIS WKT字符串: OpenGIS标准定义了一个文本格式来描述坐标系统作为简单要素规范的一个部分。这个格式是gdal中使用的坐标系统的内部工作格式。包含wkt坐标系统描述的文件的文件名可以被用来作为坐标系统参数，或者坐标系统元素本身也可以被用来作为命令行参数。

ESRI WKT字符串：ESRI 在他们的ArcGIS产品（ArcGIS中的.prj文件）中使用了一种经过精简OGC WKT的格式，而且这个格式被用在一个和wkt相似的风格的文件中。但是文件名要被加以ESRI::前缀。比如"ESRI::NAD 1927 StatePlane Wyoming West FIPS 4904.prj"。

空间参考网址URLs：可以使用一个空间参考的网址来指定，如：http://spatialreference.org/ref/user/north-pacific-albers-conic-equal-area/。

文件名：可以使用一个包含WKT、Proj.4的字符串，或者XML/GML格式的坐标系统定义的文件。

## 二、GDAL工具说明

### 1. gdalinfo 输出文件信息

用法：

gdalinfo[--help-general] [-mm] [-stats] [-hist] [-nogcp] [-nomd]

[-noct] [-nofl] [-checksum] [-proj4][-mdd domain]\*

[-sd subdataset] datasetname

参数说明：

gdalinfo程序输出gdal支持的栅格格式的一系列信息。

-mm

强制计算栅格每个波段的最大最小值。

-stats

读取和现实图像统计信息，如果指定该参数，将强制计算图像的统计信息，如各个波段的最大值、最小值、均值、标准差等。

-hist

输出所有波段的直方图信息。

-nogcp

禁止地面控制点（GCP）列表打印。这可能对大量的GCP的数据集来说是十分有用的。比如L1B AVHRR或者hdf4MODIS数据，这些数据包含了成千上万的地面控制点。

-nomd

禁止元数据打印，一些数据集可能包含极多的元数据字符串。

-noct

禁止输出颜色表。

-checksum

强制计算数据集中所有波段的checksum。

-mdd domain

输出指定区域的元数据信息。

-nofl

仅显示文件列表中的第一个文件信息。GDAL1.9.0开始支持该参数。

-sd subdataset

如果输入的数据集包含几个子数据集，那么将使用指定的数字来替代（从1开始）完整的子数据集名称。GDAL1.9.0开始支持该参数。

-proj4

输出文件的坐标系统按照PROJ.4类型的字符串输出。GDAL1.9.0开始支持该参数。

gdalinfo同时会输出如下的信息（如果有的话）：

当前文件的格式驱动信息

栅格数据大小(行列数)

文件的坐标系统(OGC WKT形式)

图像关联到地理的转换参数(当前不包含旋转系数)

地理上的边界坐标，如果可能的话还有基于经纬度的完整的地理转换参数（如果是GCPs就没有）

地面控制点(GCPs)

所有的（包括子栅格的元数据）文件元数据

波段数据类型

波段颜色信息（RGB,Gray等）

波段颜色表信息

波段瓦片大小（文件块大小）

波段描述

波段最大最小值（已经经过计算的）

波段CheckSum值（已经经过计算的）

波段无意义值（NODATA值）

波段可获得的略缩图分辨率

波段单位类型(如：波段的高程是米制还是英制)

波段的假颜色列表

举例：

gdalinfoF:/Work/Data/utm.tif

Driver:GTiff/GeoTIFF

Sizeis 512, 512

CoordinateSystem is:

PROJCS["NAD27/ UTM zone 11N",

GEOGCS["NAD27",

DATUM["North\_American\_Datum\_1927",

SPHEROID["Clarke1866",6378206.4,294.978698213901]],

PRIMEM["Greenwich",0],

UNIT["degree",0.0174532925199433]],

PROJECTION["Transverse\_Mercator"],

PARAMETER["latitude\_of\_origin",0],

PARAMETER["central\_meridian",-117],

PARAMETER["scale\_factor",0.9996],

PARAMETER["false\_easting",500000],

PARAMETER["false\_northing",0],

UNIT["metre",1]]

Origin= (440720.000000,3751320.000000)

PixelSize = (60.000000,-60.000000)

CornerCoordinates:

UpperLeft ( 440720.000, 3751320.000) (117d38'28.21"W, 33d54'8.47"N)

LowerLeft ( 440720.000, 3720600.000) (117d38'20.79"W, 33d37'31.04"N)

UpperRight ( 471440.000, 3751320.000)(117d18'32.07"W, 33d54'13.08"N)

LowerRight ( 471440.000, 3720600.000)(117d18'28.50"W, 33d37'35.61"N)

Center ( 456080.000, 3735960.000) (117d28'27.39"W, 33d45'52.46"N)

Band1 Block=512x16 Type=Byte, ColorInterp=Gray

### 2. gdal\_translate 格式转换

用法：

gdal\_translate[--help-general]

[-ot{Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/

CInt16/CInt32/CFloat32/CFloat64}][-strict]

[-of format] [-b band] [-mask band][-expand {gray|rgb|rgba}]

[-outsize xsize[%] ysize[%]]

[-unscale] [-scale [src\_min src\_max[dst\_min dst\_max]]]

[-srcwin xoff yoff xsize ysize][-projwin ulx uly lrx lry]

[-a\_srs srs\_def] [-a\_ullr ulx uly lrxlry] [-a\_nodata value]

[-gcp pixel line easting northing[elevation]]\*

[-mo "META-TAG=VALUE"]\* [-q][-sds]

[-co "NAME=VALUE"]\* [-stats]

src\_dataset dst\_dataset

参数说明：

gdal\_translate工具可以用来在不同格式间转换栅格数据。同时还可以做一些诸如提取子栅格、重采样和数据类型转换等操作。

-ot: type

指定输出波段的数据类型。

-strict:

对于这转换过程中出现丢失数据等错误直接报错，不进行忽略处理。之前的参数叫-not\_strict，表示对其进行宽大处理，不报错。

-of format:

选择输出格式。默认是GeoTiff（GTiff）。注意，指定的时候用格式简称。

-b band:

选择一个波段来输出。波段号从1开始编号。多个 -b 参数可以用于选择输出某几个波段或者重新对波段进行排序。从GDAL1.8.0开始，波段可以设置为“mask,1”（或者直接mask）来讲输入数据集的第一个波段作为一个mask波段来使用。

-mask band:

（从GDAL1.8.0开始）(GDAL >= 1.8.0)选择一个输入波段来作为创建输出数据的掩码波段。波段数是从1开始，band可以设置为none，用来避免复制整个输入数据集作为掩码，否则在默认情况下（“auto”），除非掩码是一个alpha通道，或者使用参数-b mask来进行输出，参数band也可以设置为“mask,1”（或者直接mask）来讲输入数据集的第一个波段作为一个mask波段来使用。

-expand gray|rgb|rgba:

（从GDAL1.6.0开始）将带有颜色表信息的单波段文件展开为RGB三波段文件或者RGBA四波段文件。对于输出格式为JPEG，JPEG2000，MrSID，ECW等不支持颜色表的数据来说很有用。灰度值（从GDAL1.7.0开始）也可以使用颜色表展开为一个数据集，但输出文件中仅仅包含灰度级别的一个索引。

-outsize xsize[%] ysize[%]:

设置输出文件的大小。大小以象元为单位，除非用“%”来标记，这时，表示的是输出为输入图层大小的百分比。

-scale [src\_min src\_max [dst\_mindst\_max]]:

重新组织输入象元的值。将它们从src\_min~src\_max范围内缩放到dst\_min ~ dst\_max范围内。如果省略，输出范围将为0~255。输入范围将由源数据自动计算。

-unscale:

大概意思是，不对波段中的数据进行缩放转换，在使用-ot设置输出文件类型时这个参数往往是很有用的。（这个参数的英文有点绕口，不好翻译，有知道的同学麻烦告诉我一下）。

-srcwin xoff yoff xsize ysize:

选择一个取值窗口，通过该窗口在原图像中的行列位置来拷贝数值。

-projwin ulx uly lrx lry:

选择一个取值窗口，通过该窗口在原图像中地理坐标范围来拷贝数据（类似srcwin）。参数中的四个值，使用的投影坐标。

-a\_srs srs\_def:

给输出文件投影强制指定坐标系。srs\_def可以是任何常用的GDAL/OGR格式的投影信息，如：WKT、Proj4、EPSG:n 或者一个包含着wkt的文件的文件名。

-a\_ullr ulx uly lrx lry:

强制指定输出文件的空间转换边界范围（图像的四至范围）。而将原图像的四至范围忽略掉。

-a\_nodata value:

指定一个无意义值到输出波段。从GDAl1.8.0开始，可以设置为none来使用原文件中的nodata值作为输出文件的nodata值。

-mo "META-TAG=VALUE":

如果可以，给输出数据设置一个元数据的键和其对应的值。

-co "NAME=VALUE":

通过一个创建参数来指定输出格式特殊创建要求。多个-co 参数可以组合起来使用。创建参数可以参考各个数据格式本身说明。

-gcp pixel line easting northingelevation:

添加指定地面控制点到输出数据集。这个选项可以被多次使用，以提供一系列的地面控制点GCPs 。GCP格式为：列号 行号 横坐标 纵坐标 高程值。

-q:

安静模式，不输出进度信息以及其他非错误信息。

-sds:

拷贝文件中所有子数据集到各自的输出文件中。通常这个参数用在HDF或者OGDI这样有子数据集的格式中。

-stats:

强制计算（重新计算）数据的统计信息。自GDAL1.8.0开始支持该参数。

src\_dataset:

输入数据集名称，可以是文件名，或者是一个多数据集文件中的一个子数据集的URL地址（比如HDF数据集中的一个子数据集）。

dst\_dataset:

输出文件名。

举例：

将utm.tif转换为一个以分块存储的GeoTiff文件。

gdal\_translate -of GTiff-co "TILED=YES" utm.tif utm\_tiled.tif

创建一个JPEG压缩的Tiff图像，同时使用内部掩码从一个RGBA数据集中。

gdal\_translate rgba.tifwithmask.tif -b 1 -b 2 -b 3 -mask 4 -co COMPRESS=JPEG -co PHOTOMETRIC=YCBCR--config GDAL\_TIFF\_INTERNAL\_MASK YES

创建一个RGBA图像从一个RGB数据中使用一个掩码。

gdal\_translatewithmask.tif rgba.tif -b 1 -b 2 -b 3 -b mask

### 3. gdaladdo 建立金字塔

用法：

gdaladdo[-r {nearest,average,gauss,cubic,average\_mp,average\_magphase,mode}]

[-ro] [-clean] [--help-general]filename levels

参数说明：

gdaladdo工具可以用于为大多数支持的格式建立或者重建金字塔。可以使用下面几种重采样算法中的一种来进行缩小重采样操作。

-r {nearest(default),average,gauss,cubic,average\_mp,average\_magphase,mode}:

指定重采样方法，分别是最邻近（默认），均值，高斯，立方卷积等。

-ro:

（从GDAL1.6.0开始有效），以只读的方式打开数据，来创建外部金字塔（特别对于GeoTIFF来说）。

-clean:

删除所有的金字塔。（从GDAL1.7.0开始有效）。

filename:

要建立金字塔的文件名。

levels:

要建立略缩图的层号的列表。 如果使用-clean选项是将被忽略。

模式，（从GDAL1.6.0开始有效）选择最常用的重采样方式。average\_mp 是不适合使用的， Average\_magphase用于复数数据空间的图像。Nearest 和 average 用于普通的图像。Nearest 使用最邻近采样（简单采样），它计算所有的有效值的均值来进行计算。Cubic 采样（从GDAL1.7.0开始有效）使用一个4x4的近似立方卷积内核。 Gauss 采样（从GDAL1.6.0开始有效）使用高斯内核计算。这种对于高对比度和图案边界比较明显的图像效果比较好。一般建议的采样比值是2,4,8,…，使用3x3重采样作为高斯采样的计算窗口。

gdaladdo将遵守正确NODATA\_VALUES元组（特殊的数据集元数据），因此，只有一个给定的的RGB三元组（在一个RGB图像的情况下）作为NODATA值，而不是每个波段有独立的NODATA值。

选择一个缩放级别值如2表示略缩图缩放程度是源图像每个维上分辨率的1/2。如果文件在所选缩放级别上已经存在略缩图，那么这个缩放级别上的缩略图将被重新计算并覆盖写入。

一些格式根本不支持金字塔。所以许多格式在文件以外以扩展名.ovr存储金字塔，TIFF就是如此。GeoTIFF格式直接把金字塔存储到原有的文件中。除非使用-ro标记来指定。在TIFF中创建金字塔可以通过用COMPRESS\_OVERVIEW配置参数进行压缩。所有GeoTIFF支持的压缩方法，可以在这里获得（如：--config COMPRESS\_OVERVIEW DEFLATE）。

大多数驱动也支持一个备用的略缩图格式（使用的是Erdas图像格式）。要使用这个备用格式使用USE\_RRD=YES 来设置参数。这样做会把GDAL程序创建的金字塔放到一个辅助的.aux文件中使得可以该金字塔可以直接在Erdas中使用或者也可以在ArcGIS中使用。关于如何使用GDAL创建Erdas格式的金字塔，请参考我的博文：http://blog.csdn.net/liminlu0314/article/details/6127755。

举例：

在所指定的TIFF文件内部创建金字塔：

gdaladdo -r average abc.tif 2 4 8 16

从一个ERDAS.IMG文件中创建一个外部的压缩的金字塔文件：

gdaladdo --config COMPRESS\_OVERVIEWDEFLATE erdas.img 2 4 8 16

为给定JPEG文件创建一个Erdas Imagine 格式金字塔：

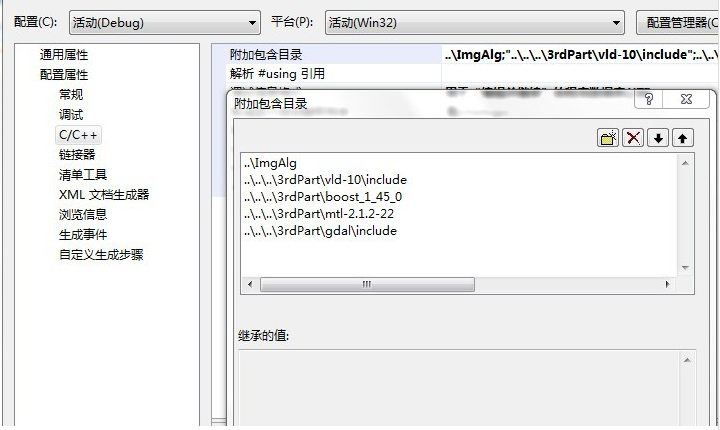
gdaladdo --config USE\_RRD YES airphoto.jpg3 9 27 81

# 八、GDAL源码剖析之GDAL开发及其调试

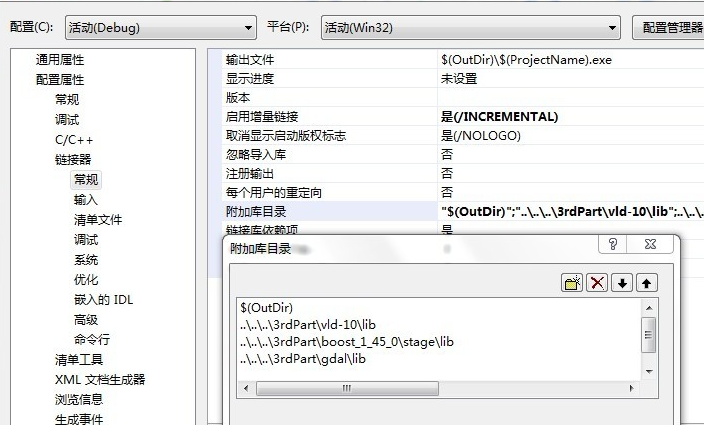
### 一、简单的调用

关于GDAL的使用，网上的资料都很多，主要还是要熟悉GDAL的组织结构，类以及类的函数等，熟悉了，使用GDAL就不在话下了。最常用的就是动态库的GDAL，当然你也可以使用静态库，这里只是简单的介绍使用动态GDAL库来做开发。

首先打开VS，新建一个工程，控制台的就成。然后在工程的属性对话框中，找到【配置属性】-【C/C++】-【常规】，右侧的【附加包含目录】中，将GDAL的include文件夹路径填写到这里，如下图：



第二、继续在属性对话框中，找到【配置属性】-【链接器】-【常规】，右侧的【附加库目录】中，将GDAL的lib文件夹路径填写到这里，如下图：



第三、在【配置属性】-【链接器】-【输入】，右侧的【附加依赖项】中，将gdal\_i.lib填写到此处。然后点击确定即可。至此，使用GDAL的环境全部搭建完成，剩下的就是在您的代码中使用GDAL了。

将下面的代码（代码摘自GDAL官方指南：<http://gdal.org/gdal_tutorial.html>）贴到刚才新建的工程中的cpp文件中，保存后编译，正常情况下会提示生成成功，然后运行，会在控制台上将图像的信息输出。

#include "gdal\_priv.h"

#include "cpl\_conv.h" //for CPLMalloc()

int main()

{

//注册文件格式

GDALAllRegister();

const char\* pszFile = "C:\\Test.img";

GDALDataset \*poDataset;

//使用只读方式打开图像

poDataset = (GDALDataset\*) GDALOpen( pszFile,GA\_ReadOnly );

if( poDataset == NULL )

{

printf( "File: %s不能打开！\n",pszFile);

return 0;

}

//输出图像的格式信息

printf( "Driver:%s/%s\n",

poDataset->GetDriver()->GetDescription(),

poDataset->GetDriver()->GetMetadataItem( GDAL\_DMD\_LONGNAME) );

//输出图像的大小和波段个数

printf( "Size is%dx%dx%d\n",

poDataset->GetRasterXSize(),poDataset->GetRasterYSize(),

poDataset->GetRasterCount());

//输出图像的投影信息

if( poDataset->GetProjectionRef() != NULL )

printf( "Projectionis `%s'\n", poDataset->GetProjectionRef() );

//输出图像的坐标和分辨率信息

double adfGeoTransform[6];

if( poDataset->GetGeoTransform( adfGeoTransform) == CE\_None )

{

printf( "Origin =(%.6f,%.6f)\n",

adfGeoTransform[0], adfGeoTransform[3]);

printf( "PixelSize = (%.6f,%.6f)\n",

adfGeoTransform[1], adfGeoTransform[5]);

}

GDALRasterBand \*poBand;

int nBlockXSize, nBlockYSize;

int bGotMin, bGotMax;

double adfMinMax[2];

//读取第一个波段

poBand = poDataset->GetRasterBand( 1 );

//获取图像的块大小并输出

poBand->GetBlockSize(&nBlockXSize, &nBlockYSize );

printf( "Block=%dx%dType=%s, ColorInterp=%s\n",

nBlockXSize, nBlockYSize,

GDALGetDataTypeName(poBand->GetRasterDataType()),

GDALGetColorInterpretationName(

poBand->GetColorInterpretation()));

//获取该波段的最大值最小值，如果获取失败，则进行统计

adfMinMax[0] = poBand->GetMinimum( &bGotMin);

adfMinMax[1] = poBand->GetMaximum( &bGotMax);

if( ! (bGotMin&& bGotMax) )

GDALComputeRasterMinMax((GDALRasterBandH)poBand, TRUE, adfMinMax);

printf( "Min=%.3fd,Max=%.3f\n", adfMinMax[0], adfMinMax[1] );

//输出图像的金字塔信息

if( poBand->GetOverviewCount() > 0 )

printf( "Band has%d overviews.\n", poBand->GetOverviewCount() );

//输出图像的颜色表信息

if( poBand->GetColorTable() != NULL)

printf( "Band hasa color table with %d entries.\n",

poBand->GetColorTable()->GetColorEntryCount() );

float \*pafScanline;

int nXSize = poBand->GetXSize();

//读取图像的第一行数据

pafScanline = (float\*) CPLMalloc(sizeof(float)\*nXSize);

poBand->RasterIO(GF\_Read, 0, 0, nXSize,1,

pafScanline, nXSize,1, GDT\_Float32, 0, 0 );

CPLFree(pafScanline);

//关闭文件

GDALClose((GDALDatasetH)poDataset);

### 二、GDAL源代码调试

在很多时候我们需要看看GDAL的内部实现，当然可以直接查看GDAL的源代码，但是直接看源代码，不能很好的理解，这时候就需要调试查看源代码中变量的内容。调试GDAL的源代码，需要GDAL的debug版本，以及编译GDAL的时候的pdb等调试文件。当然也可以把GDAL的源代码加入到你的工程中，但是这样太费时费力。

下面就有一个很简单的方法，可以直接调试进GDAL的源代码中，首先编译一下GDAL的debug版本，将编译生成的文件，主要有gdal18.dll，gdal\_i.exp，gdal\_i.lib，gdal.lib，gdal18.pdb，gdal18.ilk，gdal18.exp等文件，将gdal18开头的文件拷贝到自己工程的生成目录中，然后调试自己的程序，在执行到GDALOpen函数（或者其他GDAL的函数）时按F11键，就会进入到GDAL的源代码中进行调试GDAL代码。

### 三、GDAL使用示例

1、使用GDAL进行图像裁切，参考http://blog.csdn.net/liminlu0314/article/details/6136512

2、使用GDAL进行图像重采样，参考http://blog.csdn.net/liminlu0314/article/details/6130064

3、使用GDAL创建金字塔，参考http://blog.csdn.net/liminlu0314/article/details/6127755

# 九、GDAL源码剖析之GDAL RasterIO使用说明

分类： GDAL C++编程技术 2011-12-14 22:59 84人阅读 评论(1) 收藏 举报

一、关于RasterIO

在GDAL中读写图像是最基本的操作，那么RasterIO也就是最基本的函数了，关于RasterIO有很多方式，这个函数的功能相当强大，下面慢慢说明。RasterIO一共有两个，一个是GDALRasterBand::RasterIO，另一个是GDALDataset::RasterIO，这两个RasterIO都可以对图像数据来进行读写，大多数情况下是一样的，但是还是有一些区别的。

二、RasterIO参数说明

下面对两个RasterIO的参数进行一个简单的说明：首先是GDALRasterBand::RasterIO ，该函数的声明如下，具体可以参考下面网址：

http://gdal.org/classGDALRasterBand.html#5497e8d29e743ee9177202cb3f61c3c7

view plaincopy to clipboardprint?CPLErr GDALRasterBand::RasterIO ( GDALRWFlag eRWFlag,

int nXOff,

int nYOff,

int nXSize,

int nYSize,

void \* pData,

int nBufXSize,

int nBufYSize,

GDALDataType eBufType,

int nPixelSpace,

int nLineSpace

)

CPLErr GDALRasterBand::RasterIO ( GDALRWFlag eRWFlag,

int nXOff,

int nYOff,

int nXSize,

int nYSize,

void \* pData,

int nBufXSize,

int nBufYSize,

GDALDataType eBufType,

int nPixelSpace,

int nLineSpace

)

接下来是GDALDataset::RasterIO ，该函数的什么如下，具体形式可以参考下面的网址：http://gdal.org/classGDALDataset.html#e077c53268d2272eebed10b891a05743

view plaincopy to clipboardprint?CPLErr GDALDataset::RasterIO ( GDALRWFlag eRWFlag,

int nXOff,

int nYOff,

int nXSize,

int nYSize,

void \* pData,

int nBufXSize,

int nBufYSize,

GDALDataType eBufType,

int nBandCount,

int \* panBandMap,

int nPixelSpace,

int nLineSpace,

int nBandSpace

)

CPLErr GDALDataset::RasterIO ( GDALRWFlag eRWFlag,

int nXOff,

int nYOff,

int nXSize,

int nYSize,

void \* pData,

int nBufXSize,

int nBufYSize,

GDALDataType eBufType,

int nBandCount,

int \* panBandMap,

int nPixelSpace,

int nLineSpace,

int nBandSpace

)

由于这两个函数的参数基本一致，大多数参数都是一样的，下面就一起进行说明：

第一个参数RWFlag来指定是读数据还是写入数据，其值只能有两个即：GF\_Read 和GF\_Write，分别表示读取数据和写入数据。

第二个和第三个参数nXOff, nYOff表示读取或者写入图像数据的起始坐标图像的左上角坐标为（0，0）。

第四个和第五个参数 nXSize, nYSize表示读取或者写入图像数据的窗口大小，nXSize表示宽度，nYSize表示高度，均使用像素为单位，该宽度和高度是从第二个和第三个参数处开始计算。这两个参数和第二第三个参数一起表示就是，读取和写入图像的窗口位置和大小。

第六个参数pData是指向存储数据的一个指针。如果是写入数据，那么会将pData中的数据写入到栅格图像中去；如果是读取数据，那么会将栅格数据中的数据读入到pData中。pData的真实数据类型是通过后面的eBufType参数来指定的。如GDT\_Byte就是代表的是一个8U的数据类型，如果是GDT\_Float32就表示的是一个32F（float）的数据类型。RasterIO会自动将读入的数据按照参数eBufType指定的数据类型进行转换，需要注意的是，将浮点数转换为整数时，将对数据进行四舍五入处理；而且从一个大的存储单位转换到一个较小的存储单位是所进行的操作是截断操作而不是按照比例缩小操作，比如原来的实际数据中float的存取数据范围超过了缓冲数据指定的数据byte类型的最大存储范围，操作将把超过byte存储的范围外的数据进行丢弃处理，而不是将float缩小到0～255。

第七个和第八个参数nBufXSize和nBufYSize参数指定缓冲区的大小。注意pData的大小应当是nBufXSize×nBufYSize。当读取的数据是完整分辨率的数据（原始数据，没有进行缩放操作），他们应该设置和取值窗口的大小相同，也就是与第四个和第五个参数相等，但是在读取时使用了缩小或者放大系数，那么他们需要根据这个缩放系数进行调整。在这种情况下，RasterIO将会使用缩略图组overviews（金字塔）中某个合适的缩略图来进行读取数据。

第九个和第十个参数（对于GDALDataset来说还有第十一个参数nBandSpace）nPixelSpace和nLineSpace（以及nBandSpace）参数一般情况下是将0作为缺省值。但是，他们可以用于控制存取的内存数据的排列顺序，可以使用这两个参数将图像数据按照另一种组织形式读取内存缓冲区中。也就是说这两个（三个）参数可以读取或者写入非常规组织的缓冲数据。这个首先可以用于在一个缓冲区中包含多个波段数据，并且各个数据之间是交叉排列的，比如一个图像中的数据组织是RGBRGBRGB…，而普通的数据可能是RRR…GGG…BBB…，我们一般读取到的数据就是RGBRGBRGB…这种排列，现在需要使用RRR…GGG…BBB…这样的排列，一般想法就是自己写个for循环之类的，重新组织一次，其实完全没有必要，只要设置这两个（三个）参数就可以达到这个目的。

nPixelSpace表示的是在一个扫描行中一个像元的字节偏移起始点到下一个像元字节偏移起始点之间的字节间隔，如果默认使用0，那么将使用eBufType作为实际的两个像元之间的字节间隔。

nLineSpace表示在一行数据和下一行数据之间的起始字节见的间隔，如果使用0，那么将会使用eBufType\*nBufXSize来表示实际间隔。

此外如果使用的GDALDataset::RasterIO函数,最后还有一个参数叫nBandSpace,同上,这个参数的意思就是一个波段与下一个波段之间的起始字节间的间隔,如果使用0,实际将使用eBufType\*nBufXSize\*nBufYSize来表示。

对于GDALDataset::RasterIO函数还有两个参数nBandCount和panBandMap,分别表示要读取的波段个数和波段序号，尤其是后一个参数波段序号，可以自定义先读取那一个波段，后读取那一个。具体使用方法见下。

## RasterIO使用方法示例

使用示例一,在Windsow位图数据颜色排列是BGR，但是图像存储的可能是按照RGB来存储的，一般的做法是将数据按照每个波段读出来，然后再认为的按照BGR来进行组织，其实完全可以使用后面三个参数来将读出来的数据自动按照BGR的方式组织好。只要将参数设置为：nBandCount=3;panBandMap =new int[]{3,2,1}即可。

view plaincopy to clipboardprint?int panBandMap [3]= {3,2,1}; //按照BGR BGR BGR ... 来读取数据组织

DT\_8U \*pData = new DT\_8U[iWidth\*iHeight\*3];

poDataset ->RasterIO(GF\_Read, 0, 0, iWidth,iHeight, pData,iWidth, iHeight, (GDALDataType)iDataType, 3, panBandMap,iDataType\*3, iDataType\*iMthWidth\*3, iDataType);

int panBandMap [3]= {3,2,1}; //按照BGR BGR BGR ... 来读取数据组织

DT\_8U \*pData = new DT\_8U[iWidth\*iHeight\*3];

poDataset ->RasterIO(GF\_Read, 0, 0, iWidth,iHeight, pData,iWidth, iHeight, (GDALDataType)iDataType, 3, panBandMap,iDataType\*3, iDataType\*iMthWidth\*3, iDataType);

使用示例二，实现的是将7波段图像中的第2 3 4波段按照3 42的顺序读入内存中，逐像素存储：

view plaincopy to clipboardprint? intbandmap[7];

bandmap[0]=3;

bandmap[1]=4;

bandmap[2]=2;

Scanline=(nImagSizex\*8+31)/32\*4;

　BYTE pafScan=( BYTE )CPLMalloc(sizeof(byte)\*nImgSizeX\*nImgSizeY\*3)

　poDataset->RasterIO(GF\_Read, 0, 0,nImgSizeX,nImgSizeY, pafScan,

nImgSizeX,nImgSizeY,GDT\_Byte,3,bandmap,3,Scanline\*3,1 );

intbandmap[7];

bandmap[0]=3;

bandmap[1]=4;

bandmap[2]=2;

Scanline=(nImagSizex\*8+31)/32\*4;

　BYTE pafScan=( BYTE )CPLMalloc(sizeof(byte)\*nImgSizeX\*nImgSizeY\*3)

　poDataset->RasterIO(GF\_Read, 0, 0,nImgSizeX,nImgSizeY, pafScan,

nImgSizeX,nImgSizeY,GDT\_Byte,3,bandmap,3,Scanline\*3,1 );

以这种方式读取之后，直接可构建位图进行显示。这里可以按照自己的需要进行其他方式读取。以上读取方式仅仅为了显示方便，如进行图像处理相关运算，则按波段全部读出会比较方便，即按照常规的方式读取处理：

view plaincopy to clipboardprint?poDataset->RasterIO( GF\_Read, 0,0,nImgSizeX,nImgSizeY, pafScan,

nImgSizeX,nImgSizeY,GDT\_Byte,bandcount,0,0,0,0);

poDataset->RasterIO( GF\_Read, 0,0,nImgSizeX,nImgSizeY, pafScan,

nImgSizeX,nImgSizeY,GDT\_Byte,bandcount,0,0,0,0);

之前申请的内存改为：

view plaincopy to clipboardprint?BYTE pafScan=new byte[nImgSizeX\*nImgSizeY\*bandcount];

BYTE pafScan=new byte[nImgSizeX\*nImgSizeY\*bandcount];

将图像数据读入内存后，即可通过指针pafScan对图像进行你想要进行的操作了。