Headfirst GDAL Documentation

Release 0.1

Wu Qi

August 26, 2016

Contents

CHAPTER 1

前言

O'Reilly有一个HeadFirst系列，面对完全的初学者的书籍。因为经常用到GDAL，也碰到过一些问题，所以从头开始，写一些经验，给大家分享。网上也有许多GDAL的文章，除了李民录老师写的，大部分不太系统，虽然有系统文档，但都是英文的，很多人都不想看，因此写这个文档，算是总结，也帮新手少走点弯路。本系列主要涉及的是GDAL的各个接口和调用，以及一些基础概念的阐述，不涉及其他应用方面，GDAL更新也比较快，所以本文也只能介绍主要的读写接口以及GDALWarp部分，不涉及更深入的内容。如果想学好，推荐还是认真阅读英文文档。

本文主要注重于栅格影像处理，矢量ogr库暂时没有涉及（以后可能会添加），主要使用 c++ 语言，vs2010 或 vc6 平台下使用。

1.1 GDAL是什么

GDAL 是 `Geospatial Data Abstraction Library` 的缩写，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平台，Google Earth和跨平台的GRASS GIS系统。

1.2 GDAL能做什么

GDAL可以用来对各种栅格数据格式进行读写，也可以用于格式相互转换，图像几何校正、重投影、重采样等。

GDAL中包括很多算法，比如surf匹配算法，TPS纠正等几何纠正算法，OGR库中也可以进行拓扑运算，因为是开源的，所以部分算法可以参照源码，如TPS中矩阵运算可以添加Aramdillo库，匹配算法也可以借鉴学习。

1.3 参考文档

文档参考的文章和示例来自以下位置：

- GDAL源文档
- 李民录GDAL专栏
- WKT

- 地图投影系列介绍
- 地图投影简明笔记

1.4 如何使用GDAL库

很多新手不知道下载的编译完成的库如何使用，下面以vc6和vs2010中如何使用为例，给大家讲解下。本部分参照 GDAL源码剖析（六）之GDAL开发及其调试 ,其中讲的已经比较清楚,本段简单介绍下载或编译完成的库如何使用。

Note:

- 如果不关心如何编译调试GDAL的话，可以直接使用 框架 ,里面有已经设置好的环境和基础的两影像相加的示例.
- 没有C++基础的新手，推荐阅读下 C++静态库与动态库 ，对库的解说比较好.

1.4.1 目录说明

下载完成的GDAL库，一般有 `lib include html data bin` 几个文件夹，这些是必须文件，其他有些是调试或绑定其他语言用的文件夹 ,下面分别介绍其作用:

- lib 里面包含 gdal_i.lib 库文件
- include 包含头文件
- html 包含html文档说明
- data 包含GDAL的数据文件，例如epsg文件、dxf头文件、S-57 定义文件等，可以设置 `GDAL_DATA` 环境变量来改变其路径
- bin 工具集，包括一个dll文件。

1.4.2 安装

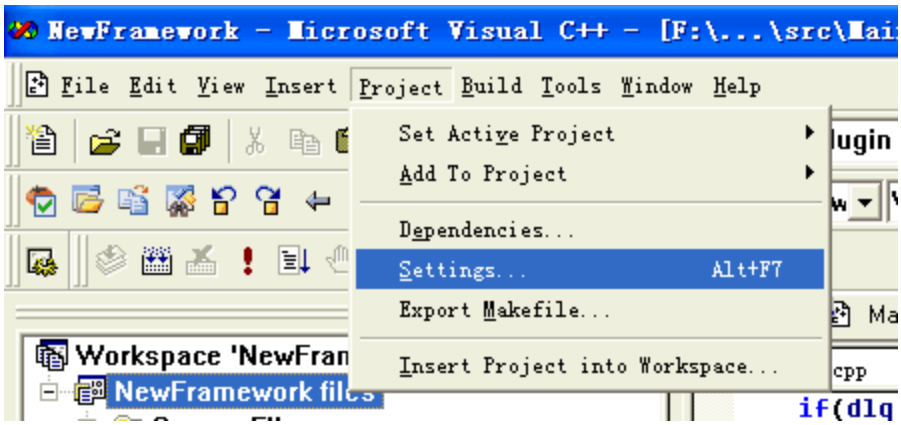
新建工程，在新建工程中添加 `GDAL` 目录，将上述文件夹中的 `lib include` 文件夹拷贝到新建的 `GDAL` 目录中，至此完成安装。

Attention:
- 也可以在工程设置中设置所在路径，本步骤并非必须步骤，仅为了方便下面介绍以及新手理解

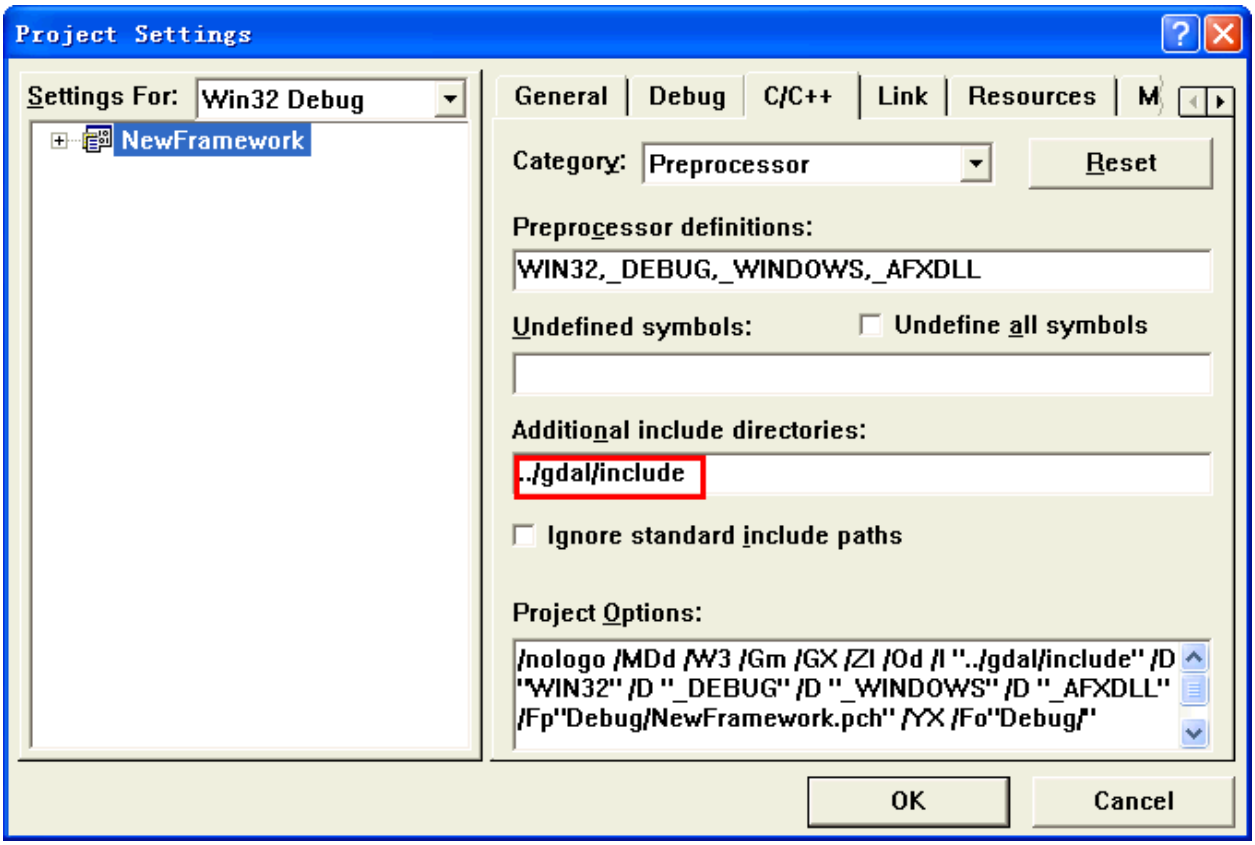
1.4.3 工程设置

vc6

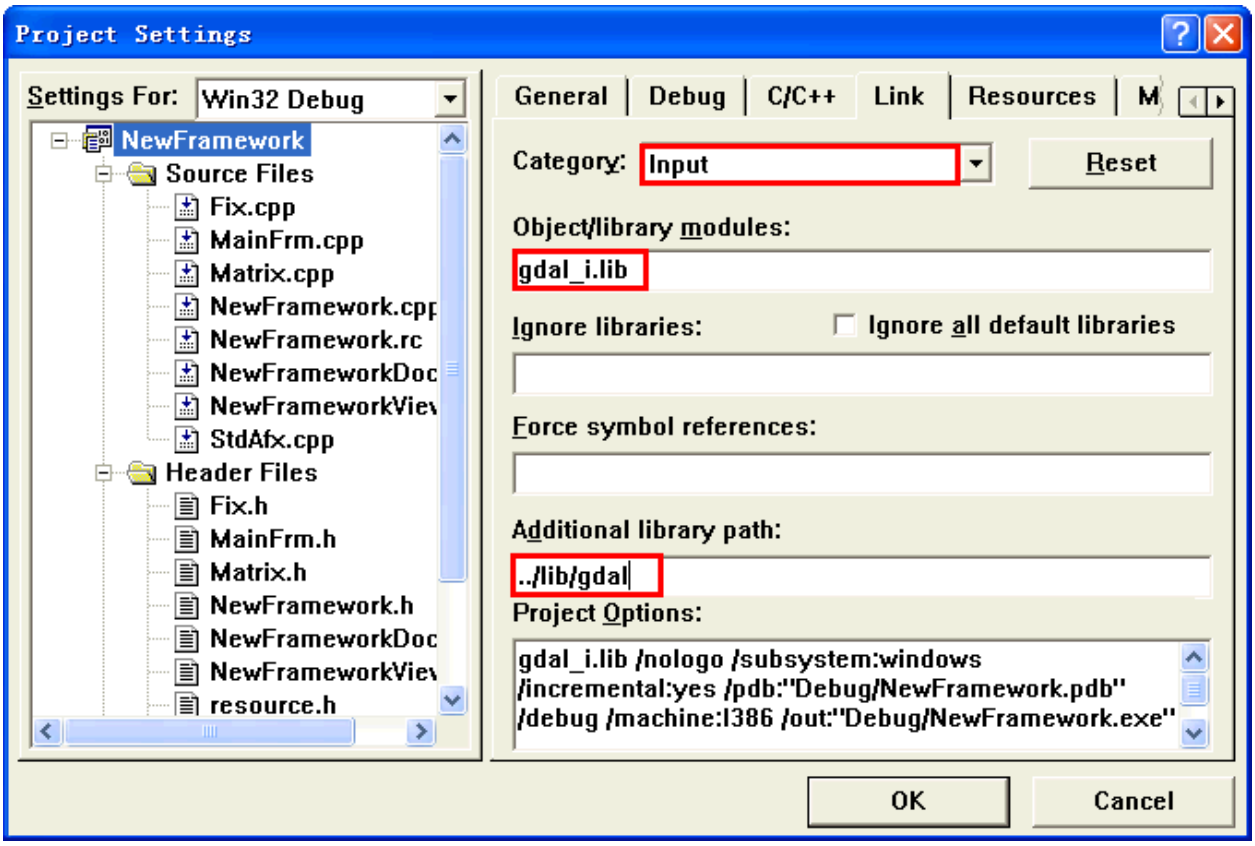
打开或新建vc6工程，在菜单中找到工程->设置(Project->Settings)，如下所示:



设置Include路径，点击 c++ Tab页,在 Category 中选择 Preprocessor,第三项,添加Include目录中,添加gdal的include文件夹路径:



设置Lib路径,点击 Link Tab页,在 Category 中,选择 Input ,第一项中填入 gdal_i.lib ,第四项中添加gdal中的lib文件夹路径:

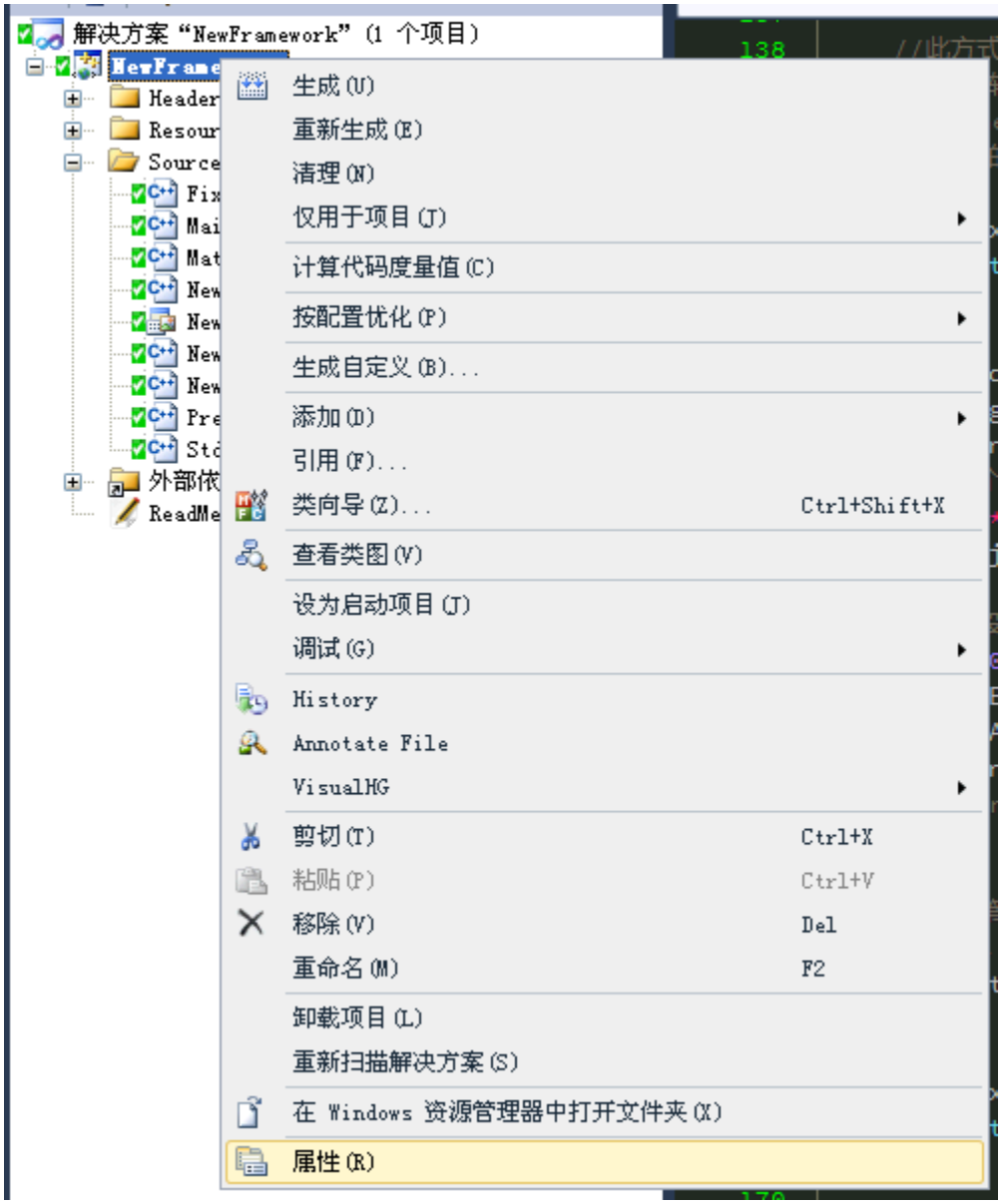


Attention:

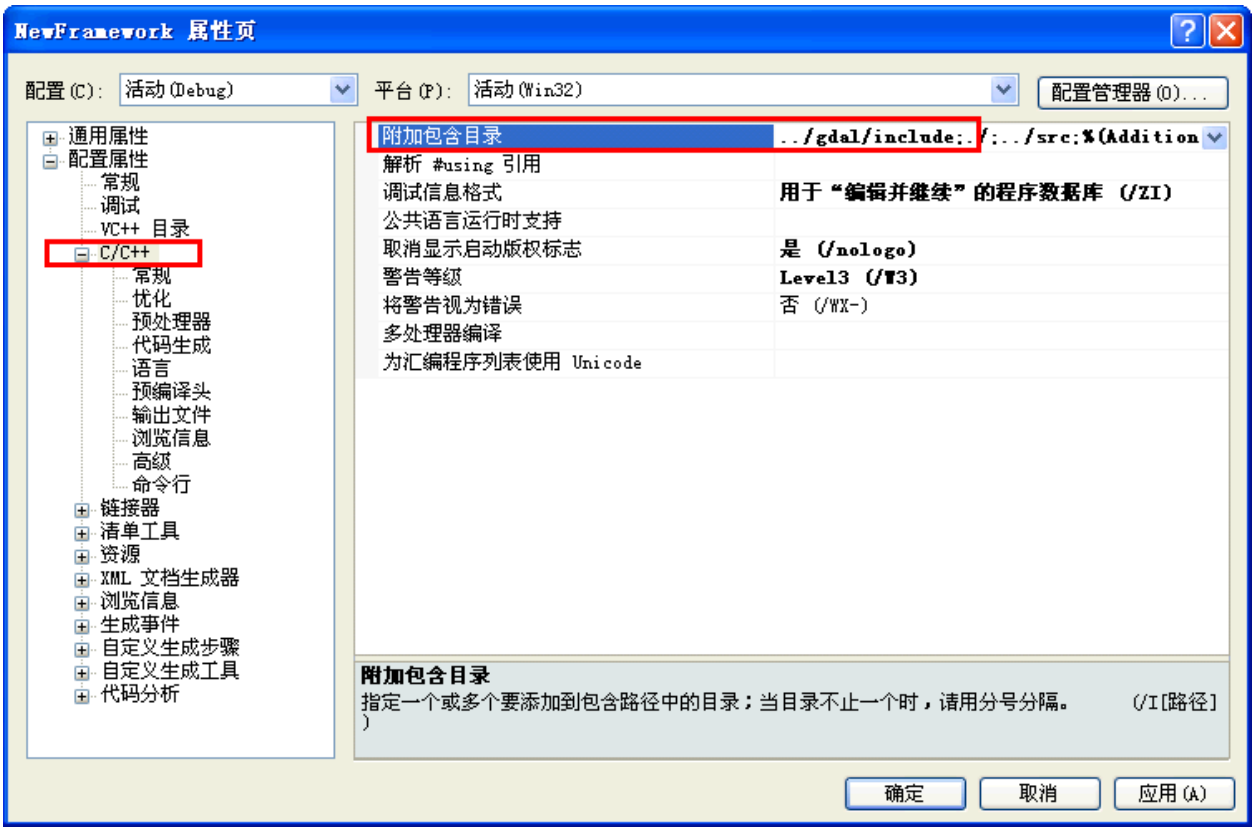
- 编译完成后，需要将 `bin` 或者 `lib` 文件夹下面的DLL文件全部拷贝到输出目录中（Debug或Release文件夹中）

vs2010

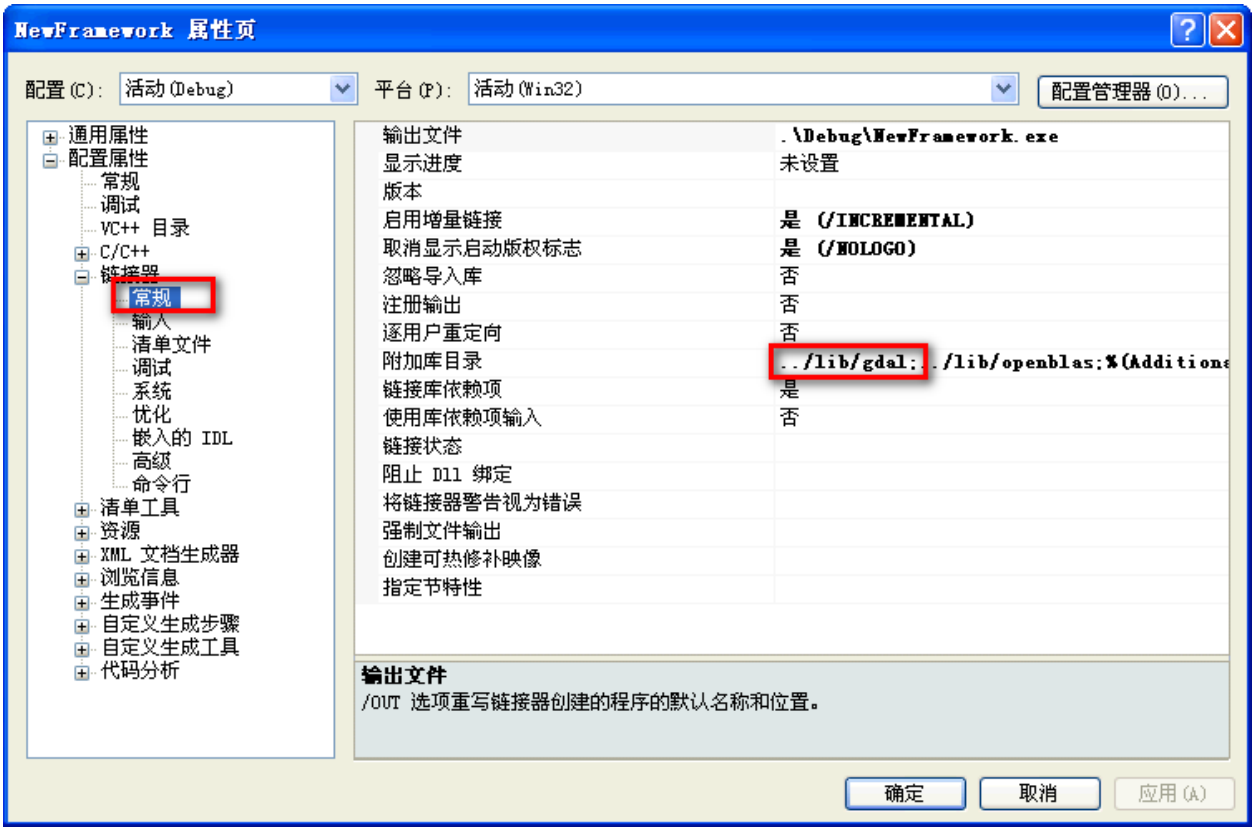
打开或新建vs工程，右键设置项目属性，如下所示：



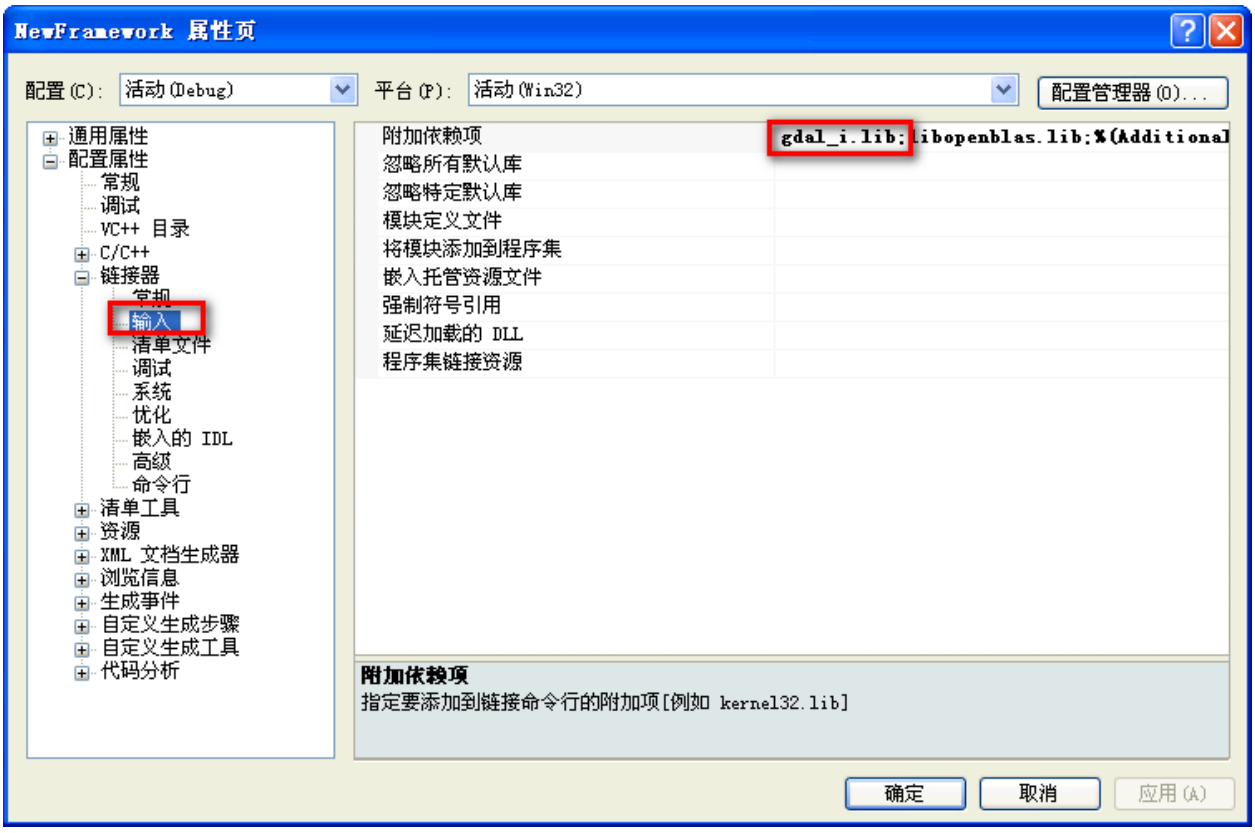
设置Include路径，点击 C++ 填写第一项，在附加包含目录中添加gdal的Include文件夹路径：



设置Lib路径,点击 链接器 ,在 常规 中, 附加库目录 里添加gdal中的lib文件夹路径:



设置Lib库，点击 链接器 ,在 输入 中, 附加依赖项 里添加填入 gdal_i.lib:



Attention:

- 编译完成后，需要将 bin 或者 lib 文件夹下面的DLL文件全部拷贝到输出目录中（Debug或Release文件夹中）

这样就完成了环境设置，接下来就可以按照 栅格数据读写 的例子读写数据了。

1.5 OGR库说明

现在仅初步使用 OGR 库，大部分只是翻译文档,没有深入使用，有些翻译并不准确。如有不确定的地方，请参照文档。在 GDAL2.0 中， OGR 库和 GDAL 库的数据集已经合并，其他部分基本没变，且 v2.0 版本尚未发布，现有翻译参照 v.1.11 来写。

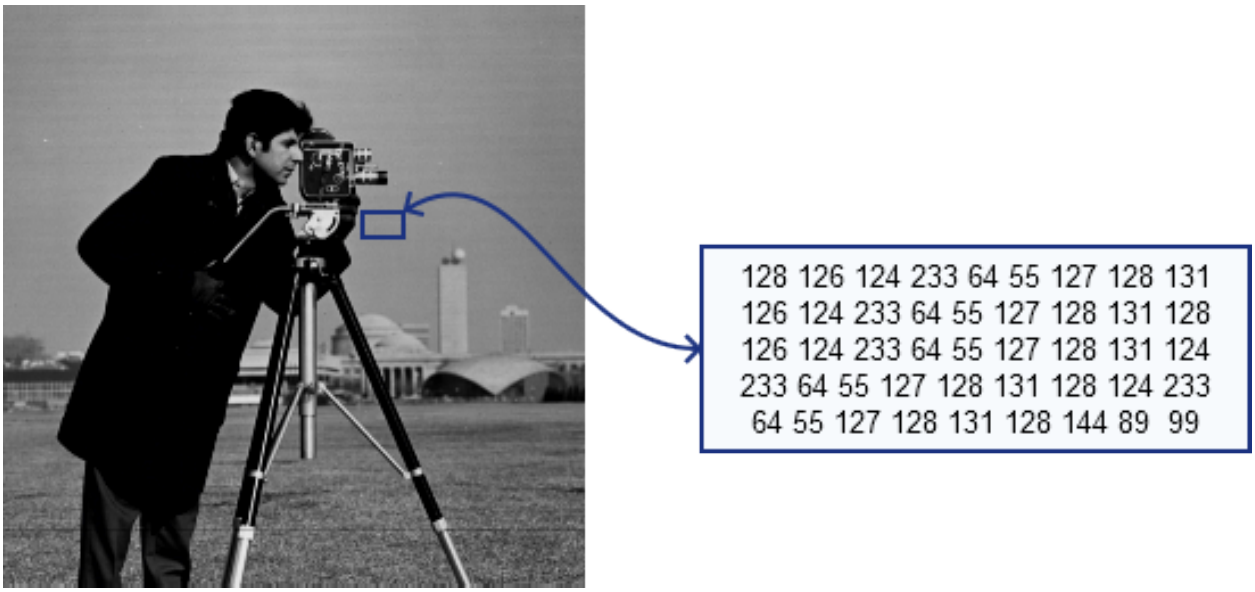
1.6 使用的工具

文档使用 sphinx 编写，演示图片使用 Pencil 绘制。pdf使用TexLive2013生成。

CHAPTER 2

影像是什么

从真实世界中获取数字影像有很多方法，比如数码相机、扫描仪、CT或者磁共振成像。无论哪种方法，我们（人类）看到的是影像，而让数字设备来“看“的时候，则是在记录影像中的每一个点的数值。[1]



比如上面的影像，在标出的蓝框中你见到的只是一块灰色区域，但是计算机中识别为一个矩阵，该矩阵包含了所有像素点的强度值。如何获取并存储这些像素值由我们的需求而定，最终在计算机世界里所有影像都可以简化矩阵信息。当然，影像还有其他信息，下面我们即将介绍。

2.1 影像元数据

影像元数据是指影像中除了信息矩阵以外的一些参考数据。例如影像的长、宽、波段数、数据类型、存储方式、投影坐标、太阳高度角、获取时间、传感器信息等内容。使用 *gdalinfo*文件信息工具 ，我们就可以看到影像的元数据信息：

[1] 来自OpenCV文档

```
CMD
E:\GDAL\bin> gdalinfo teapot.tiff
Driver: GTiff/GeoTIFF
Files: E:\GDAL\bin\teapot.tiff
Size is 256, 256
Coordinate System is `'
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left (   0.0,   0.0)
Lower Left (   0.0, 256.0)
Upper Right ( 256.0,   0.0)
Lower Right ( 256.0, 256.0)
Center      ( 128.0, 128.0)
Band 1 Block=256x8 Type=Byte, ColorInterp=Red
Band 2 Block=256x8 Type=Byte, ColorInterp=Green
Band 3 Block=256x8 Type=Byte, ColorInterp=Blue
Band 4 Block=256x8 Type=Byte, ColorInterp=Undefined
```

2.2 GDAL中的影像

GDAL中，影像也可以称为 数据集 ，大致分为两种类型，多波段影像和多数据集影像。 *GDAL数据模型* 中我们将详细介绍GDAL对影像数据如何解析和读写，这章中，我们只简单的分析一下两种数据的共性和区别。

2.2.1 多波段数据

多波段数据是我们比较熟悉的，例如tiff格式、jpg格式、png格式、img格式等，都是多个波段的数据。我们一般处理的也都是多波段的数据。

多波段的数据实际上可以看为一个二维的数组，在GDAL中就是包含了多个 波段 的一个 数据集 ，里面保存了影像的所有数据以及元数据。其中数据存在 波段 中，每个波段可以当作一维数组进行处理。如下图所示:

2.2.2 多数据集数据

这种数据遥感影像中也比较常见，例如hdf格式、he5格式、netcdf格式等。这些数据在GDAL中，与上文介绍的多波段数据不同，是包含了多个 数据集 的 数据集 。可以简单的理解为：一个多数据集数据包含了多个 多波段数据 ，当然，实际上，这些 多波段数据 里大部分只有一个波段。如下图所示:

一般我们会将多数据集数据转化为多波段的数据，方便处理。下面就是一个多数据集数据转换为多波段数据的函数，其中的部分内容我们将在后文中陆续展开讲解:

```
/**
  GDALAllRegister,仅此一次
*/
void RegisterAll()
{
  if(GDALGetDriverByName("GTiff") == NULL) {
    GDALAllRegister();
```

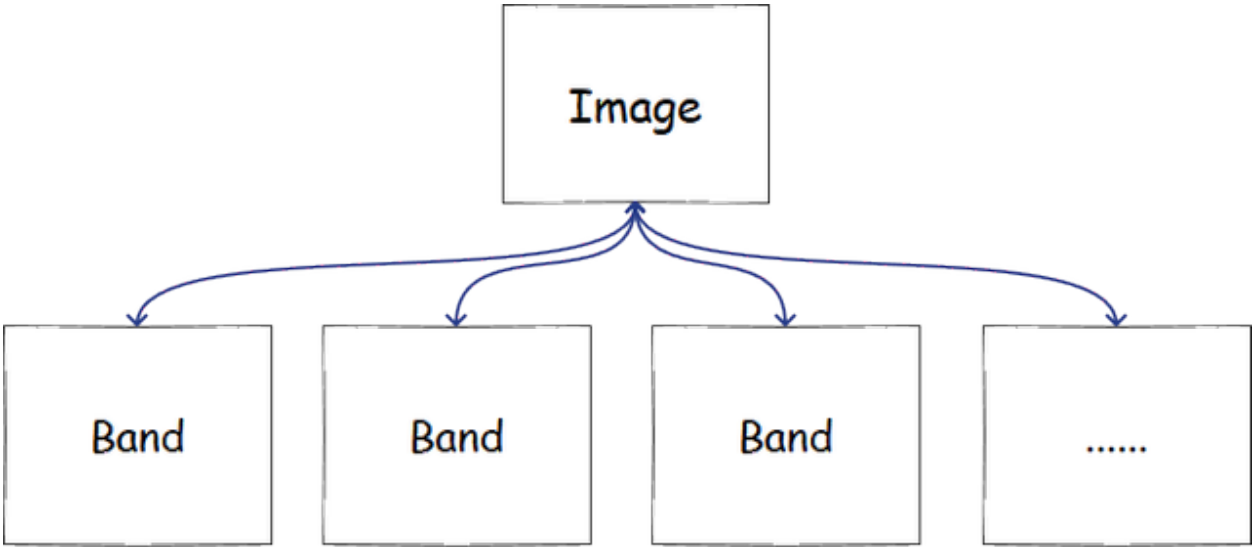
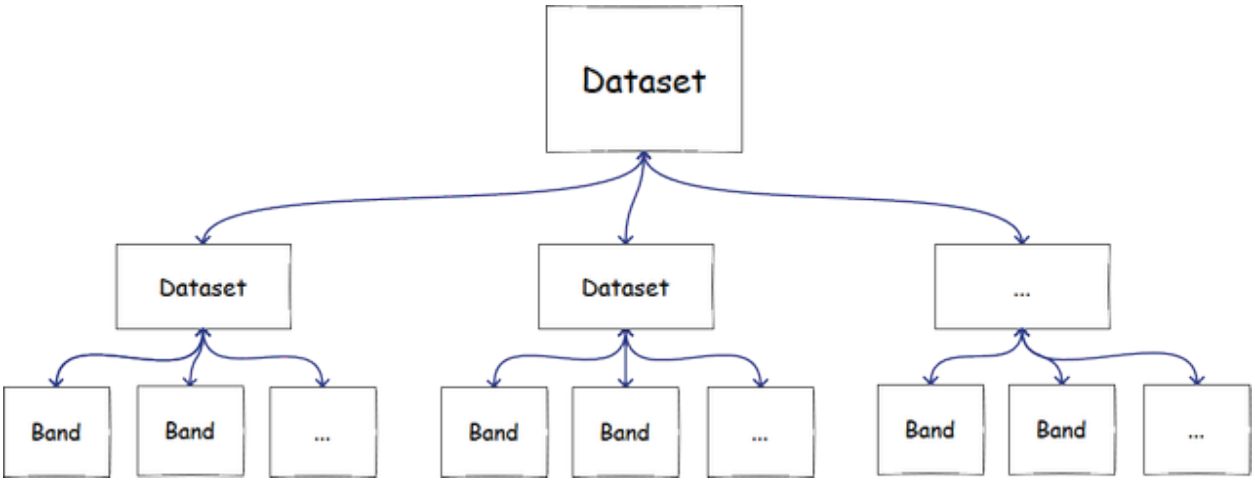


Image
Band
Band
Band
.....


Dataset
Dataset
Dataset
...
Band
Band
...
Band
Band
...
Band
Band
...

```
  }
}

/**
所有subset转成一个tif
  @param fileName    输入文件名
  @param outfile     输出文件名
  @param pBandIndex  所需输出的数据集，用数组表示，从1开始，例如[1,3,5]，传入NULL表示全部数据集
  @param pBandCount  所需输出数据集个数，如果pBandIndex设置为NULL，此参数无作用
  @return
 */
void subsets2tif(const char *fileName, const char *outfile, \
                 int *pBandIndex, int pBandCount)
{
  int i, j;
  char *outFileName = NULL; //输出文件名

  if(outfile == NULL) {
    int charNum = strlen(fileName);
    charNum += 5;
    outFileName = new char[charNum];
    strcpy(outFileName, fileName);
    strcat(outFileName, ".tif") ;
  } else {
    int charNum = strlen(outfile);
    outFileName = new char[charNum + 1];
    strcpy(outFileName, outfile);
  }

  RegisterAll();//注册类型，打开影像必须加入此句
  GDALDataset *pDataSet = (GDALDataset *) GDALOpen(fileName, GA_ReadOnly);

  if(pDataSet == NULL) {
    printf("不能打开该文件，请检查文件是否存在！");
    return ;
  }

  //获取子数据集
  char **papszSUBDATASETS = pDataSet->GetMetadata("SUBDATASETS");
  int subdsNumber =  papszSUBDATASETS == NULL ? 1 : CSLCount(papszSUBDATASETS) / 2;
  char **vSubDataSets = new char*[subdsNumber]; //子数据集名称列表
  char **vSubDataDesc = new char*[subdsNumber]; //子数据集描述列表
  char *papszMetadata = NULL;

  //如果没有子数据集，则其本身就是一个数据集
  if(papszSUBDATASETS == NULL) {
    const char *Metadata = GDALGetDriverShortName((GDALDriverH)pDataSet);
    vSubDataSets[0] = new char[strlen(Metadata) + 1];
    vSubDataDesc[0] = new char[strlen(Metadata) + 1];
    strcpy(vSubDataSets[0], Metadata);
    strcpy(vSubDataDesc[0], Metadata);
  } else {
    int iCount = CSLCount(papszSUBDATASETS);  //计算子数据集的个数

    if(iCount <= 0) {    //没有子数据集,则返回
      GDALClose((GDALDriverH)pDataSet);
      return;
    }
```

```
    //将子数据集压入列表
    for(i = 0, j = 0; papszSUBDATASETS[i] != NULL; i++) {
      if(i % 2 != 0) {
        continue;
      }

      char *setInfo = papszSUBDATASETS[i];
      setInfo = strstr(setInfo, "=");

      if(setInfo[0] == '=') {
        memmove(setInfo, setInfo + 1, strlen(setInfo)); //提取元数据中子数据集名称
        vSubDataSets[j] = new char[strlen(setInfo) + 1];
        strcpy(vSubDataSets[j], setInfo);
      }

      char *descptn = papszSUBDATASETS[i + 1];
      descptn = strstr(descptn, "=");

      if(descptn[0] == '=') {
        memmove(descptn, descptn + 1, strlen(descptn)); //提取元数据中子数据集描述
        vSubDataDesc[j] = new char[strlen(descptn) + 1];
        strcpy(vSubDataDesc[j], descptn);
      }

      j++;
    }//end for
  }

  int Width, Height;
  GDALDriver *poDriver;
  const char *pszFormat = "GTiff";
  poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);

  if(poDriver == NULL) {
    return;
  }

  //将每个子数据集转为对应的tif假设所有数据集大小相同
  bool init = false;
  int iBands = pBandIndex != NULL ? pBandCount : subdsNumber;
  int realBandCount = 0;

  for(i = 0; i < subdsNumber; i++) {
    if(pBandIndex != NULL) {
      bool runNext = true;

      for(j = 0; j < pBandCount; j++) {
        if(pBandIndex[j] == (i + 1)) {
          realBandCount = j + 1;
          runNext = false;
          break;
        }
      }

      if(runNext) {
        continue;
      }
    } else {
```

```
    realBandCount = i + 1;
  }

  char *dsName = vSubDataSets[i];
  GDALDataset *subDs = (GDALDataset *) GDALOpen(dsName, GA_ReadOnly);

  if(subDs == NULL) {
    continue;
  }

  Width = subDs->GetRasterXSize();
  Height = subDs->GetRasterYSize();
  void *subData;
  GDALRasterBand *poBand = subDs->GetRasterBand(1);
  GDALDataType eBufType = poBand->GetRasterDataType();

  switch(eBufType) {
    case GDT_Byte:
      subData = new char[Width * Height];
      break;

    case GDT_Int16:
      subData = new short[Width * Height];
      break;

    case GDT_Int32:
      subData = new int[Width * Height];
      break;

    case GDT_Float32:
      subData = new float[Width * Height];
      break;

    case GDT_Float64:
      subData = new double[Width * Height];
      break;

    default:
      subData = new double[Width * Height];
      break;
  }

  poBand->RasterIO(GF_Read, 0, 0, Width, Height, subData,\
                   Width, Height, eBufType, 0, 0);
  //写入影像
  GDALDataset *pDstDs = NULL;

  if(!init) { //创建影像
    char **papszOptions = NULL;
    //设置bsq或者 BIP bsq:BAND,bip:PIXEL
    papszOptions = CSLSetNameValue(papszOptions, "INTERLEAVE", "BAND");
    pDstDs = poDriver->Create(outFileName, Width, Height, \
                              iBands, eBufType, papszOptions);
    double geos[6];
    subDs->GetGeoTransform(geos);//变换参数
    char *pszProjection = NULL;
    char *pszPrettyWkt = NULL;
    //获取ds1坐标
```

```
      OGRSpatialReferenceH  hSRS;

      if(GDALGetProjectionRef(subDs) != NULL) {
        pszProjection = (char *) GDALGetProjectionRef(subDs);
        hSRS = OSRNewSpatialReference(NULL);

        if(OSRImportFromWkt(hSRS, &pszProjection) == CE_None) {
          OSRExportToPrettyWkt(hSRS, &pszPrettyWkt, FALSE);
          pDstDs->SetProjection(pszPrettyWkt);
        }
      }

      //设置坐标
      pDstDs->SetGeoTransform(geos);
      CPLFree(pszPrettyWkt);
      pDstDs->SetMetadata(subDs->GetMetadata());
      pDstDs->FlushCache();
      init = true;
    } else {
      //打开影像
      pDstDs = (GDALDataset *) GDALOpen(outFileName, GA_Update);
    }

    GDALRasterBand *poBandOut;
    poBandOut = pDstDs->GetRasterBand(realBandCount);
    poBandOut->SetScale(poBand->GetScale());
    poBandOut->SetOffset(poBand->GetOffset());
    poBandOut->SetUnitType(poBand->GetUnitType());
    poBandOut->SetColorInterpretation(poBand->GetColorInterpretation());
    poBandOut->SetDescription(poBand->GetDescription());
    poBandOut->SetNoDataValue(poBand->GetNoDataValue());
    //GDT_Float32和 **OutputImg 类型要对应!
    poBandOut->RasterIO(GF_Write, 0, 0, Width, Height, \
                        subData, Width, Height, eBufType, 0, 0);
    pDstDs->FlushCache();
    //关闭
    GDALClose(subDs);
    GDALClose(pDstDs);
    delete[] subData;
  }

  GDALClose(pDataSet);

  if(papszMetadata) {
    delete papszMetadata;
  }

  for(i = 0; i < subdsNumber; i++) {
    delete[] vSubDataDesc[i];
    delete[] vSubDataSets[i];
  }

  delete[] vSubDataDesc;
  vSubDataDesc = NULL;
  delete[] vSubDataSets;
  vSubDataSets = NULL;
  delete[] outFileName;
}
```

CHAPTER 3

GDAL数据模型

本章主要介绍GDAL的数据模型，通过上一章，我们大致了解数据集和波段的意义，这章中，我们来看GDAL中这些概念的实现。本章基本参考 GDAL_Datamodel 为了方便理解，没有完全按照原文翻译，有部分删改。

3.1 数据集

上章介绍了，简单的说，一个数据集可以看作为一幅影像。它包括了元数据、投影信息、波段等等，GDAL中用 GDALDataset 类来表示数据集。数据集除了内部的波段，还包括以下内容:

3.1.1 投影系统

GDAL中，使用的是 WKT 串来表示投影，具体的表示内容可以参考链接，下面用例子简单的介绍一下，#后面表示注释:

```
PROJCS["WGS 84 / UTM zone 52N",                          #投影名称
    GEOGCS["WGS 84",                                     #地理坐标系统名
        DATUM["WGS_1984",                                #水平基准面
            SPHEROID["WGS 84",6378137,298.257223         #椭球体名称、长半轴、反偏率
                AUTHORITY["EPSG","7030"]],               #外部权威的空间参考系统的编码
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0],                           #中央经线Greenwich，0度标准子午线
        UNIT["degree",0.0174532925199433],               #指定测量使用的单位。在地理坐标系下使用角度。
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],                   #投影方法，这里是通用墨卡托投影
    PARAMETER["latitude_of_origin",0],                   #PARAMETER表示投影参数,0表示纬度起点为0度
    PARAMETER["central_meridian",129],                   #投影带的中央经线是东经129度
    PARAMETER["scale_factor",0.9996],                    #中央经线的长度比是0.9996
    PARAMETER["false_easting",500000],                   #坐标纵轴向西移动500km
    PARAMETER["false_northing",0],                       #横轴没有平移
    UNIT["metre",1,                                      #指定测量使用的单位，指定米为测量单位。
        AUTHORITY["EPSG","9001"]],                       #外部权威的空间参考系统的编码
    AUTHORITY["EPSG","32652"]]
```

这部分和地图投影关系很大，如果没有基础，请参考 地图投影系列介绍 和 地图投影简明笔记 。

使用 GDALDataset::GetProjectionRef() 函数可以获取数据集的投影信息，GDAL中使用仿射变换可以将地理坐标和图像坐标进行转换，在下一节中我们将具体展示；使用 GDALDataset::GetGCPProjection() 可以获取GCP点的投影。返回均为WKT字符串，如果返回""，则表示该数据集没有投影或者投影没有被识别。

3.1.2 仿射地理变换

GDAL数据集有两种方式表示栅格数据中像元位置（图像中某个点在影像中的行列号）和投影坐标系（不是经纬度，是投影到二维平面的地图坐标，二者可以通过地图投影进行相互转换）间的关系：仿射变换和GCP点。大部分数据都是用仿射变换描述的，本节中描述仿射变换。

仿射变换由六个参数实现，`GDALDataset::GetGeoTransform()` 可以获取仿射变换参数数组。将像元位置转换为投影坐标的公式如下：

```
/*
六个参数分别是:
    geos[0]  top left x 左上角x坐标
    geos[1]  w-e pixel resolution 东西方向像素分辨率
    geos[2]  rotation, 0 if image is "north up" 旋转角度，正北向上时为0
    geos[3]  top left y 左上角y坐标
    geos[4]  rotation, 0 if image is "north up" 旋转角度，正北向上时为0
    geos[5]  n-s pixel resolution 南北向像素分辨率
    x/y为图像的x/y坐标，geox/geoy为对应的投影坐标
*/
geox = geos[0] + geos[1] * x + geos[2] * y;
geoy = geos[3] + geos[4] * x + geos[5] * y
```

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)

3.1.3 GCP点

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GCP点共用`GDALDataset::GetGCPProjection()` 获取的地理参考坐标系。GCP结构体在GDAL中表示如下：

```
typedef struct
{
    char     *pszId;          //ID，可以用字母+数字表示，不能重复
    char     *pszInfo;        //描述信息,一般为空
    double      dfGCPPixel;     //列号
    double     dfGCPLine;      //行号
    double     dfGCPX;         //投影x坐标
    double     dfGCPY;         //投影y坐标
    double     dfGCPZ;         //投影z坐标
} GDAL_GCP;
```

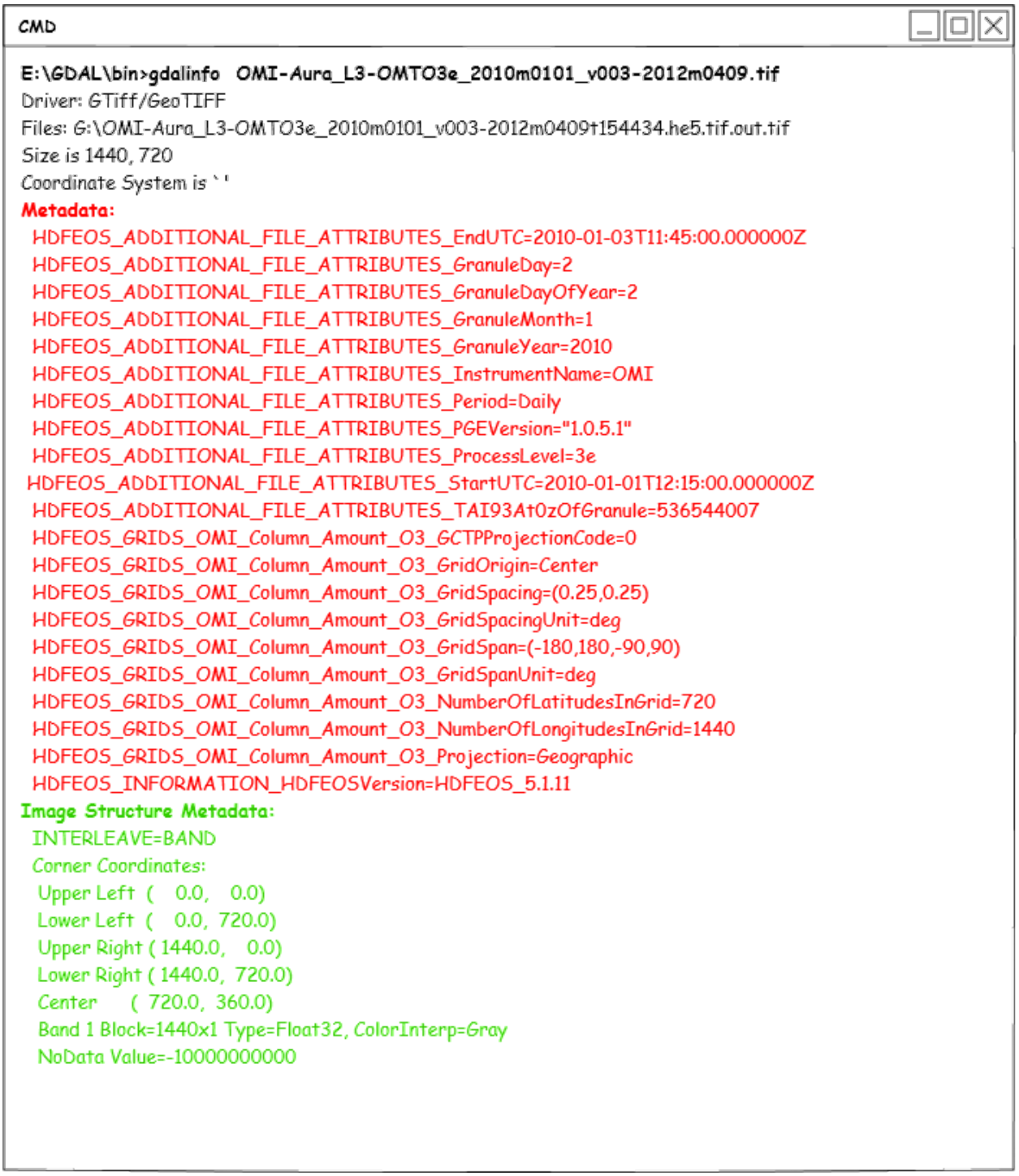
GDAL数据模型没有实现由GCPs产生坐标系的变化的机制，而是把具体的操作留给用户实现,一般采用多项式插值实现。

如果数据集有投影的话，会包含仿射变换或GCPS。两者都有的很少见，如果两者都有，则无法用权威坐标系定义。

3.1.4 元数据

GDAL中，元数据是以键值对呈现的辅助数据，可以使用 *gdalinfo*文件信息工具 获取影像元数据，结果如图所示：

如上图中红色部分所示，在 `MetaData` 栏中，所有的数据都是以 **key=value** 的方式组织的。`key` 中不能设置空格和一些特殊字符，`value` 为非空值的任意长度的内容。一个数据集的元数据一般不超过100kb，否则会导致性能严重下降。


CMD
E:\GDAL\bin>gdalinfo OMI-Aura_L3-OMTO3e_2010m0101_v003-2012m0409.tif
Driver: GTiff/GeoTIFF
Files: G:\OMI-Aura_L3-OMTO3e_2010m0101_v003-2012m0409t154434.he5.tif.out.tif
Size is 1440, 720
Coordinate System is `'
Metadata:
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_EndUTC=2010-01-03T11:45:00.000000Z
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_GranuleDay=2
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_GranuleDayOfYear=2
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_GranuleMonth=1
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_GranuleYear=2010
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_InstrumentName=OMI
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_Period=Daily
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_PGEVersion="1.0.5.1"
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_ProcessLevel=3e
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_StartUTC=2010-01-01T12:15:00.000000Z
HDFEOS_ADDITIONAL_FILE_ATTRIBUTES_TAI93At0zOfGranule=536544007
HDFEOS_GRIDS_OMI_Column_Amount_O3_GCTPProjectionCode=0
HDFEOS_GRIDS_OMI_Column_Amount_O3_GridOrigin=Center
HDFEOS_GRIDS_OMI_Column_Amount_O3_GridSpacing=(0.25,0.25)
HDFEOS_GRIDS_OMI_Column_Amount_O3_GridSpacingUnit=deg
HDFEOS_GRIDS_OMI_Column_Amount_O3_GridSpan=(-180,180,-90,90)
HDFEOS_GRIDS_OMI_Column_Amount_O3_GridSpanUnit=deg
HDFEOS_GRIDS_OMI_Column_Amount_O3_NumberOfLatitudesInGrid=720
HDFEOS_GRIDS_OMI_Column_Amount_O3_NumberOfLongitudesInGrid=1440
HDFEOS_GRIDS_OMI_Column_Amount_O3_Projection=Geographic
HDFEOS_INFORMATION_HDFEOSVersion=HDFEOS_5.1.11
Image Structure Metadata:
INTERLEAVE=BAND
Corner Coordinates:
Upper Left (0.0, 0.0)
Lower Left (0.0, 720.0)
Upper Right (1440.0, 0.0)
Lower Right (1440.0, 720.0)
Center (720.0, 360.0)
Band 1 Block=1440x1 Type=Float32, ColorInterp=Gray
NoData Value=-10000000000

一些数据格式支持用户自定义的基本元数据，一些数据格式会定义特殊的键。

相似的元数据可以组成一个域，如上面的红色部分和绿色部分，就属于不同的域。默认域没有名称，有些特殊用途的元数据有特殊的域。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。

下面介绍些常见的域。

子数据集域

多数据集数据 中一般都会有这个域，用于存储子数据集的信息,一般是一个类似下面的信息列表:

```
Subdatasets:
    SUBDATASET_1_NAME=NETCDF:"example.nc":auditTrail
    SUBDATASET_1_DESC=[2x80] auditTrail (8-bit character)
    SUBDATASET_2_NAME=NETCDF:"example.nc":data
    SUBDATASET_2_DESC=[1x61x172] data (32-bit floating-point)
    SUBDATASET_3_NAME=NETCDF:"example.nc":lat
    SUBDATASET_3_DESC=[61x172] lat (32-bit floating-point)
    SUBDATASET_4_NAME=NETCDF:"example.nc":lon
    SUBDATASET_4_DESC=[61x172] lon (32-bit floating-point)
```

`NAME=` 后面的内容是子数据集名称，用来打开数据集下的子数据集。`DESC=` 后面跟的是描述。ADRG, ECRGTOC, GEORASTER, GTiff, HDF4, HDF5, netCDF, NITF, NTv2, OGDI, PDF, PostGISRaster, Rasterlite, RPFTOC, RS2, WCS, and WMS这些数据类型都有子数据集。

图像结构域

与影像格式相关的元数据信息将存在这个域中，就是上图中绿色部分。这部分元数据在转换格式的时候，一般不会自动拷贝到新数据中。其中有一些典型的条目，例如 `COMPRESSION` 表示压缩程度等。

RPC域

RPC元数据域描述有理多项式系数几何模型，这也是描述像元和地理参考位置之间变换信息用的。更详细的信息参见 RPCs in GeoTIFF 文档。

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。它是一个存有xml字符串的单XML文档。

3.2 波段

波段是真正存储数据的结构，GDAL中用 `GDALRasterBand` 类来表示单个波段。

波段有以下的性质：

- 宽和高：如果是全分辨率的波段，那长和宽与数据集中的定义是一样的。
- 数据类型：有Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, and the complex types CInt16, CInt32, CFloat32, and CFloat64类型.
- 块大小：缓冲块的大小，tiff中一般是以行作为缓冲块

- 描述字串：可选
- nodata数值：可选
- 光栅名称：可选，这个特性可以用来指定一些比较重要的数据。
- **波段的解释信息，枚举型：**
  - GCI_Undefined: 默认值.
  - GCI_GrayIndex: 灰度值波段
  - GCI_PaletteIndex: 颜色表索引
  - GCI_RedBand: 红色波段
  - GCI_GreenBand: 绿色波段
  - GCI_BlueBand: 蓝色波段
  - GCI_AlphaBand: 透明通道
  - GCI_HueBand: 色调
  - GCI_SaturationBand: 饱和度
  - GCI_LightnessBand: 光强
  - GCI_CyanBand: 青色波段
  - GCI_MagentaBand: 品红波段
  - GCI_YellowBand: 黄色波段
  - GCI_BlackBand: t黑色波段.

3.3 颜色表

颜色表在GDAL中，由多个 `GDALColorEntry` 组成， `GDALColorEntry` 结构体描述如下：

```
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。

3.4 金字塔层

一个波段可以有零个或多个金字塔层，这个结构在构建影像金字塔的时候将用到。每个层都相当于独立的 `GDALRasterBand` 。每层的行列数和原始的波段有所不同，但是覆盖的地理区域是相同的。

金字塔层是原始波段的降采样实现的，一般用于快速显示。

波段有 `HasArbitraryOverviews` 属性来判定在某些分辨率上是否有金字塔层。该属性可以应用于fft编码或者是网络传输切片等状况中。

CHAPTER 4

栅格数据读写

本章我们就开始按顺序详细介绍GDAL读写栅格数据的过程。最后将提供完整的读写流程代码。 完整代码 中有整个创建、两影像相加、写入的流程，如果已经大致了解GDAL的读写流程，可以直接参照 完整代码 。

4.1 GdalDriver

首先，GDAL对每种格式提供了一个驱动 `GdalDriver`， `GdalDriver` 将对对应格式的数据进行管理，例如读取、创建、删除、重命名、复制、从已有数据创建新数据集等。所以，我们所有程序开头，都将添加 `GDALAllRegister()` 函数，注册所有GDAL支持的数据驱动。

再次强调，所有的程序都要首先调用GDALAllRegister() 函数，否则将无法打开任何数据。

4.2 数据读取

GDAL中数据读取的步骤如下：

- 打开数据集
- 打开数据集下所需的波段
- 读取数据

下面分步讲解如何操作：

4.2.1 打开数据集

打开 多波段数据 步骤如下：使用 `GDALOpen` 或者 `GDALOpenShared` 函数，传入文件名，打开数据集。 `GDALOpen` 和 `GDALOpenShared` 参数一样，区别在于，在同一线程，如果是相同的文件，多个 `GDALOpenShared` 打开的其实是同一个 `GDALDataset` 的引用（如果在不同线程使用，为了保证线程安全，返回的将是不同的对象）。

```
#include "gdal_priv.h"
#include "cpl_conv.h"

//...中间的程序

//所有程序前先加上
GDALAllRegister();
```

```
//pszFilename代表文件名，GA_ReadOnly表示以只读方式打开
//也可以使用GA_Update
//GDALOpenShared和GDALOpen可以互换
GDALDataset  *poDataset= (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );
if( poDataset == NULL )
{
    ...;//打开失败处理
}
```

打开 多数据集数据 方式与上面稍有区别，因为有多个子数据集，所以需要获取真实数据的话，实际上是获取子数据集中的波段，可以使用 *gdalinfo*文件信息工具 获取如下的 子数据集域

```
Subdatasets:
    SUBDATASET_1_NAME=NETCDF:"example.nc":auditTrail
    SUBDATASET_1_DESC=[2x80] auditTrail (8-bit character)
    SUBDATASET_2_NAME=NETCDF:"example.nc":data
    SUBDATASET_2_DESC=[1x61x172] data (32-bit floating-point)
    SUBDATASET_3_NAME=NETCDF:"example.nc":lat
    SUBDATASET_3_DESC=[61x172] lat (32-bit floating-point)
    SUBDATASET_4_NAME=NETCDF:"example.nc":lon
    SUBDATASET_4_DESC=[61x172] lon (32-bit floating-point)
```

打开子数据集也使用 GDALOpen 或者 GDALOpenShared 函数，但是传入的不是整个数据集的名称，而是需要打开的 SUBDATASET_N_NAME= (_N_中N为第几个子数据集，即 1 2 3 4...)后面的字符串，即子数据集名称。例如需要打开上例中data子数据集，使用如下代码：

```
#include "gdal_priv.h"
#include "cpl_conv.h"

//...中间的程序

//所有程序前先加上
GDALAllRegister();
//也可以使用GA_Update
//GDALOpenShared和GDALOpen可以互换
//pszFilename代表子数据集的NAME，注意双引号需要转义，GA_ReadOnly表示以只读方式打开
char * pszFilename = "NETCDF:\"example.nc\":data";
GDALDataset  *poDataset= (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );
if( poDataset == NULL )
{
    ...;//打开失败处理
}
```

打开了数据集后，两种数据接下来的处理方式都是一样的。

4.2.2 打开波段

GDAL中， **波段起始索引为1** ，所以循环时需要特别注意。使用 GDALDataset->GetRasterBand(int BandIndex) 函数，可以打开波段。

```
#include "gdal_priv.h"
#include "cpl_conv.h"
//...中间的程序

//所有程序前先加上
GDALAllRegister();
GDALDataset  *poDataset= (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );//打开文件
```

```
if( poDataset == NULL )
{
    //...//打开失败处理
}
//打开数据集同上
//获取影像相关信息
int i,j,width,height,bandNum;
width = inDS->GetRasterXSize();//影像宽度
height = inDS->GetRasterYSize();//影像高度
bandNum = inDS->GetRasterCount();//影像波段数量

//将第一波段读入
double *data = new double[width*height];
GDALRasterBand *band = ds1->GetRasterBand(1);//获取第一波段，波段从1开始
//如果是在循环内，需要注意，波段从1开始;
//for(i=0;i < bandNum;i++){
// GDALRasterBand *band = ds1->GetRasterBand(i+1);//这里注意下
// ......对波段处理
//}
```

4.2.3 读取内容

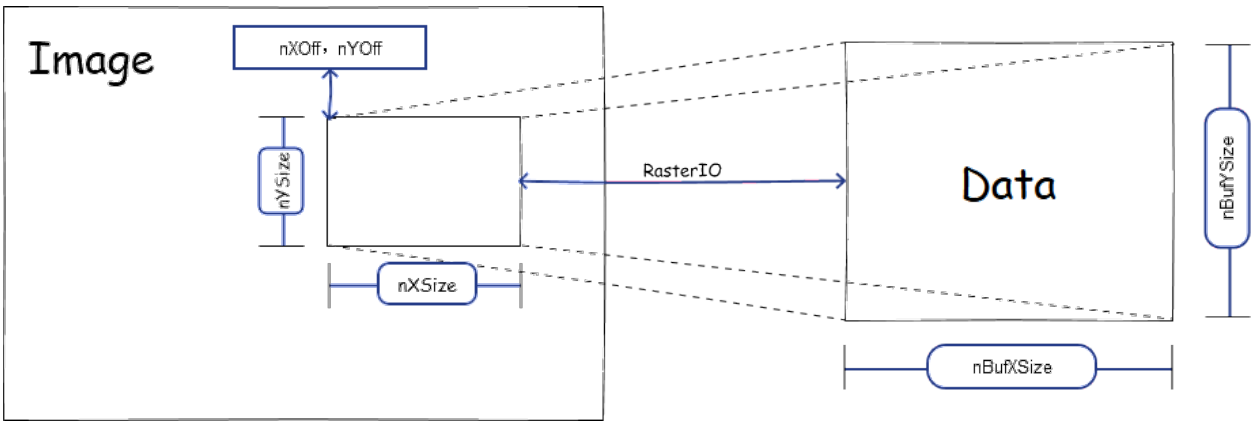
使用 RasterIO 函数获取波段中的具体内容，该函数的功能强大，参数较多,我们先用代码演示,然后用图来表示。

函数说明:

```
///CPLErr GDALRasterBand::RasterIO (
///@param eRWFlag,      //读取或者写入,GF_Read或GF_Write
///@param nXOff,        //起始点x坐标
///@param nYOff,        //起始点y坐标
///@param nXSize,       //所需读取(写入)块宽度
///@param nYSize,       //所读(写)块高度
///@param * pData,      //所读(写)数据,指针,
///@param nBufXSize,    //一般跟nXSize一致，用于缩放图像,
///                       图像将按nBufXSize/nXsize在x尺度缩放（会自动重采样）
///                      ，一般不需要调整
///@param nBufYSize,    //一般跟nYSize一致
///@param eBufType,     //与pData的实际类型一致,GDT_Float64
///                       代表double,其他的可以跳到定义查看
///@param nPixelSpace,
///                       设置为0为自动判断，一般设为0
///                       表示的是当前像素值和下一个像素值之间的间隔，单位是字节
///                       例:byte类型，就是1，double类型，就是8
///@param nLineSpace
///                       设置为0为自动判断，一般设为0
///                       表示的是当前行和下一行的间隔，，单位是字节
///                       例如，一行300像素，类型为int,此时就是300*4 = 1200
///@return              //是否成功,成功返回CE_None ,失败返回 CE_Failure
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
)
```

示意图：



读取单幅影像第一波段原始数据代码示例：

```
#include "gdal_priv.h"
#include "cpl_conv.h"
//...中间的程序

//所有程序前先加上
GDALAllRegister();
GDALDataset  *poDataset= (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );//打开文件
if( poDataset == NULL )
{
    //...打开失败处理
}
//打开数据集同上
//获取影像相关信息
int i,j,width,height,bandNum;
width = inDS->GetRasterXSize();//影像宽度
height = inDS->GetRasterYSize();//影像高度
bandNum = inDS->GetRasterCount();//影像波段数量

//将第一波段读入
double *data = new double[width*height];
GDALRasterBand *band = ds1->GetRasterBand(1);//获取第一波段，波段从1开始
band->RasterIO(GF_Read,0,0,width,height,data,width,height,GDT_Float64,0,0);
```

Warning: 使用RasterIO函数时需要注意以下几点：

- 波段索引从一开始计算
- pData数组要有足够的大小
- pData数组数据类型要和eBufType对应
- 读取完成后必须要释放数组和关闭数据集，波段不需要关闭

4.2.4 关闭数据集

程序结束前，必须要关闭数据集，使用 GDALClose() 函数。

```
GDALClose(ds2);
```

4.3 数据写入

数据写入分为几个步骤

- 获取驱动
- 创建数据集
- 写入数据
- 关闭数据集

4.3.1 获取驱动

GdalDriver 中已经说明，不论读取还是写入， GDALDriver 都是很重要的存在。如果需要输出特定格式，我们必须首先获取该格式的驱动，才能创建该格式的数据集。获取方式如下：

```
GDALDriver *poDriver;

//一般使用tif作为输出，如果有特殊需求，请参阅gdal文档，一般修改这里改为其他驱动名称即可，例如PNG
const char *pszFormat ="GTiff";
poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);//获取特殊的驱动。
if(poDriver == NULL) {
    return;
}
```

4.3.2 创建数据集

根据不同输出需要，创建数据集有两种方式：

- 输出的影像与输入的影像大小、投影都相同，只有数据不同，就可以从已经读入的数据集中创建
- 投影或大小不同，需要创建新的数据集

下面分别说明两种方式如何创建数据集，下文都是以输出GeoTiff作为示例，不同的 GdalDriver 中，可以设置的参数不同，具体请参考 GDAL支持格式 中对每种格式的说明。

从已读入的数据集中创建

从已经读入的数据集中创建数据集，可以使用 GDALDriver 的 CreateCopy 函数。

CreateCopy 函数原型如下：

```
//从已有的dataset中创建
//@param pszFilename, 输出文件名
//@param poSrcDS, 已有的dataset
//@param bStrict, TRUE表示严格等价，一般设置为FALSE，表示拷贝副本用作编辑
//@param papszOptions, 参数设置，具体可以参考每个driver的文档
```

```
//@param  pfnProgress 回调函数指针
//@param *pProgressData 传入回调函数的数据
GDALDataset * GDALDriver::CreateCopy     (
    const char *     pszFilename,
    GDALDataset *     poSrcDS,
    int     bStrict,
    char **     papszOptions,
    GDALProgressFunc     pfnProgress,
    void *     pProgressData
)
```

使用 CreateCopy 函数创建数据集代码如下:

```
GDALDataset *OutDs;
char **papszOptions = NULL;//设置压缩、存储方式等，每种格式不同，可以直接设置为NULL
                           //也可以根据需要设置,例如:
//papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
//papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );

//创建与ds1相同的坐标信息、相同大小的文件
OutDs = poDriver->CreateCopy(OutPut1,ds1,FALSE,papszOptions,NULL,NULL);
```

直接创建

直接创建数据集比从已有数据集中创建会多出创建投影的步骤:

创建数据集

直接集中创建数据集，可以使用 GDALDriver 的 Create 函数。

Create 函数原型如下:

```
//Create文件
//@param pszFilename, 输出文件名
//@param nXSize, 文件宽度
//@param nYSize, 文件高度
//@param nBands, 波段数
//@param eType, 数据类型
//@param papszOptions     参数设置，具体可以设置的属性参考每个driver的文档
//@return dataset
GDALDataset * GDALDriver::Create (
    const char *     pszFilename,
    int     nXSize,
    int     nYSize,
    int     nBands,
    GDALDataType     eType,
    char **     papszOptions
)
```

使用 Create 函数创建数据集代码如下:

```
GDALDataset *OutDs;
///设置:
char **papszOptions = NULL;
papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
```

```
///...
///创建512*512，单波段的，byte类型的数据
OutDs = poDriver->Create( pszDstFilename, 512, 512, 1, GDT_Byte,
                          papszOptions );
```

创建投影

在 投影系统 和 仿射地理变换 中已经介绍过GDAL中数据如何投影，下面我们将用代码示例：

```
///设置仿射变换参数
double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
OutDs->SetGeoTransform( adfGeoTransform );

//设置投影
OGRSpatialReference oSRS;
char *pszSRS_WKT = NULL;
oSRS.SetUTM( 11, TRUE );
oSRS.SetWellKnownGeogCS( "NAD27" );
oSRS.exportToWkt( &pszSRS_WKT );
poDstDS->SetProjection( pszSRS_WKT );
CPLFree( pszSRS_WKT );//使用完后释放
```

4.3.3 写入数据

写入数据与读取数据基本相同，都是使用 GDALRasterBand 的 RasterIO 函数，只是第一个参数变为 GF_Write ，下面就直接给出代码，不再另作说明。

```
float * outData = new float [512*512];

//对outData进行计算、处理
/*************************************************************************/
/*********************************处理*********************************/
/*************************************************************************/

GDALRasterBand *outBand = OutDs->GetRasterBand(1);
outBand->RasterIO(GF_Write,0,0,512,512,outData,512,512,GDT_Float32,0,0);
```

4.3.4 关闭数据集

数据集必须关闭后，数据才会写入到输出文件中，否则数据将在缓存中，使用 GDALClose() 函数关闭数据集。

```
GDALClose(OutDs);
```

4.3.5 分块读写

数据较大时,分块读写将提高效率,实际上大部分的图像格式中包含分块内容,一般是以行分块为主,所以块宽度为行宽可能会提高分块读取速度

```
GDALAllRegister();//注册类型，打开影像必须加入此句
GDALDataset *ds1;
ds1 = (GDALDataset *) GDALOpen(input1,GA_ReadOnly);//input1为文件名
```

```
if(ds1 == NULL) {  //读取失败
    AfxMessageBox("cant open the unReferard image!");
    return ;
}

WidthAll = ds1->GetRasterXSize(); //影像宽度
HeightAll = ds1->GetRasterYSize();//影像高度
BandNum = ds1->GetRasterCount();//影像波段数
//读取波段数据
GDALRasterBand *poBand;//不需要释放,只需要最后是否GDALDataset
double **Inputimg1 = new double*[BandNum];

int blockx = 512;//分块大小
int blocky = 512;
//分块处理.将影像分成很多blockx*blocky大小的块，通过循环对每一块进行处理
int nxNum = WidthAll/blockx+1;//计算列方向上块数
int nyNum = HeightAll/blocky+1;//计算行方向块数

int i,j,k;
int Width,Height;//块的实际宽和高
for (i = 0; i < nxNum; ++i) {
    for ( j = 0; j < nyNum; ++j) {
        //确定实际块大小
        Width = (WidthAll - (i+1)*blockx) > 0? blockx : (WidthAll - i * blockx);
        Height = (HeightAll - (j+1)*blocky) > 0? blocky : (HeightAll - j * blocky);
        //分块读取数据
        for ( k = 0; k < BandNum; ++k) {
            Inputimg1[j] = new double[Width*Height];
            //注意,获取波段数从1开始计数!!
            poBand = ds1->GetRasterBand(k+1);
            //将数据写入Inputimg1[j],RasterIOd中，参数具体含义见RasterIO:
            poBand->RasterIO(GF_Read,i*blockx, j*blocky, Width, Height,Inputimg1[j], \
                             Width,Height,GDT_Float64,0,0);
        }
        /**
        * 处理
        * 处理
        * 处理
        */
    }
}

//用完之后,关闭dataset即可,不需要释放GDALRasterBand
GDALClose(ds1);
```

4.4 GDAL2.0

GDAL2.0版本中,RasterIO添加了 `GDALRasterIOExtraArg` 结构体作为参数,默认为空,所有GDAL1的代码可以不用修改直接编译. `GDALRasterIOExtraArg` 中可以选择重采样方式/偏移/进度条等功能,定义和结构如下:

class GDALRasterIOExtraArg
向RasterIO中传入额外信息

```
/** Structure to pass extra arguments to RasterIO() method
  * @since GDAL 2.0
```

```
  */
typedef struct
{
    /*! 版本号,方便以后扩展 */
    int                 nVersion;
    /*! 重采样算法 */
    GDALRIOResampleAlg  eResampleAlg;
    /*! 进度条 callback 函数*/
    GDALProgressFunc    pfnProgress;
    /*! 进度条 callback user data */
    void               *pProgressData;
    /*! Indicate if dfXOff, dfYOff, dfXSize and dfYSize are set.
        Mostly reserved from the VRT driver to communicate a more precise
        source window. Must be such that dfXOff - nXOff < 1.0 and
        dfYOff - nYOff < 1.0 and nXSize - dfXSize < 1.0 and nYSize - dfYSize < 1.0 */
    int                 bFloatingPointWindowValidity;
    /*! 像素相对左上角点x轴偏移量,仅在bFloatingPointWindowValidity = TRUE时有效 */
    double              dfXOff;
    /*! 像素相对左上角点y轴偏移量,仅在bFloatingPointWindowValidity = TRUE时有效 */
    double              dfYOff;
    /*! 感兴趣区域的像素宽度 bFloatingPointWindowValidity = TRUE时有效 */
    double              dfXSize;
    /*! 感兴趣区域的像素高度 bFloatingPointWindowValidity = TRUE时有效 */
    double              dfYSize;
} GDALRasterIOExtraArg;
```

enum GDALRIOResampleAlg

重采样方式

```
typedef enum
{
    /*! Nearest neighbour 默认 */                       GRIORA_NearestNeighbour = 0,
    /*! Bilinear (2x2 kernel) */                        GRIORA_Bilinear = 1,
    /*! Cubic Convolution Approximation (4x4 kernel) */ GRIORA_Cubic = 2,
    /*! Cubic B-Spline Approximation (4x4 kernel) */    GRIORA_CubicSpline = 3,
    /*! Lanczos windowed sinc interpolation (6x6 kernel) */ GRIORA_Lanczos = 4,
    /*! Average */                                      GRIORA_Average = 5,
    /*! Mode (selects the value which appears most often of all the sampled points) */
                                                        GRIORA_Mode = 6,
    /*! Gauss blurring */                               GRIORA_Gauss = 7
    /* NOTE: values 8 to 12 are reserved for max,min,med,Q1,Q3 */
} GDALRIOResampleAlg;
```

INIT_RASTERIO_EXTRA_ARG

初始化GDALRasterIOExtraArg的宏

```
#define INIT_RASTERIO_EXTRA_ARG(s)  \
    do { (s).nVersion = RASTERIO_EXTRA_ARG_CURRENT_VERSION; \
         (s).eResampleAlg = GRIORA_NearestNeighbour; \
         (s).pfnProgress = NULL; \
         (s).pProgressData = NULL; \
         (s).bFloatingPointWindowValidity = FALSE; } while(0)
```

GDAL2.0中重采样读取完整代码如下:

```
#include "gdal_priv.h"
#include "cpl_conv.h"
//...中间的程序
```

```
//所有程序前先加上
GDALAllRegister();
GDALDataset  *poDataset= (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );//打开文件
if( poDataset == NULL )
{
    //...打开失败处理
}
//打开数据集同上
//获取影像相关信息
int i,j,width,height,bandNum;
width = inDS->GetRasterXSize();//影像宽度
height = inDS->GetRasterYSize();//影像高度
bandNum = inDS->GetRasterCount();//影像波段数量

//将第一波段读入
double *data = new double[width*height];
GDALRasterBand *band = ds1->GetRasterBand(1);//获取第一波段，波段从1开始
GDALRasterIOExtraArg exterArg;
INIT_RASTERIO_EXTRA_ARG(exterArg);
exterArg.eResampleAlg = GDALRIOResampleAlg::GRIORA_Cubic;//配置插值方法
band->RasterIO(GF_Read,0,0,width,height,data,width,height,GDT_Float64,0,0,&exterArg);
```

4.5 完整代码

```
#include "gdal_priv.h"
int main(int argc, char *argv[])
{
  GDALAllRegister();//注册类型，读取写入任何类型影像必须加入此句
  //以只读方式打开文件
  GDALDataset *ds1 = (GDALDataset *) GDALOpen("Input1.tif", GA_ReadOnly);

  if(ds1 == NULL) {
    printf("不能打开第一个文件，请检查文件是否存在！");
    return -1;
  }

  int WidthAll = ds1->GetRasterXSize();
  int HightAll = ds1->GetRasterYSize();
  int BandNum = ds1->GetRasterCount();
  float **InBuf1 = new float*[BandNum];
  int i, j, k;

  //分波段读取
  for(i = 0; i < BandNum; i++) {
    InBuf1[i] = new float[WidthAll * HightAll];
    GDALRasterBand *band = ds1->GetRasterBand(i + 1); //获取波段，波段从1开始
    band->RasterIO(GF_Read,
                   0,
                   0,
                   WidthAll,
                   HightAll,
                   InBuf1[i],
                   WidthAll,
                   HightAll,
                   GDT_Float32,
                   0,
```

```
                    0);
}

//打开第二个文件，假设第一个文件和第二个文件大小、波段数相同
//以只读方式打开文件
GDALDataset *ds2 = (GDALDataset *) GDALOpen("Input2.tif", GA_ReadOnly);

if(ds2 == NULL) {
  printf("不能打开第二个文件，请检查文件是否存在！");
  return ;
}

//第二个文件中的数据
float **InBuf2 = new float*[BandNum];

for(i = 0; i < BandNum; i++) {
  InBuf2[i] = new float[WidthAll * HightAll];
  GDALRasterBand *band = ds2->GetRasterBand(i + 1);
  band->RasterIO(GF_Read,
                 0,
                 0,
                 WidthAll,
                 HightAll,
                 InBuf2[i],
                 WidthAll,
                 HightAll,
                 GDT_Float32,
                 0,
                 0);
}

//写入的数据(可以最后创建写入文件,需要先创建写入数据)
float **outBuf = new float*[BandNum];

for(i = 0; i < BandNum; i++) {
  outBuf[i] = new float[WidthAll * HightAll];
}

////////////////////////////////////////////////////
//////////////////////处理//////////////////////////
////////////////////////////////////////////////////
//简单相加示例，实际上，这部分最好写入函数，或者类中单独封装起来，方便使用
for(j = 0; j < BandNum; j++) {
  for(k = 0; k < WidthAll * HightAll; k++) {
    outBuf[j][k] = InBuf1[j][k] + InBuf2[j][k];
  }
}

////////////////////////////////////////////////////
//////////////////////处理//////////////////////////
////////////////////////////////////////////////////
//创建写入图像:
GDALDriver *poDriver;
const char *pszFormat = "GTiff";
poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);

if(poDriver == NULL) {
  return;
```

```
}

GDALDataset *OutDs;
char **papszOptions = NULL;
char **papszMetadata;
//这里的参数全是指tif格式的参数，如果是其他格式，请把这里所有注释掉，或者参照文档，自行设定
//设置压缩类型，envi只认得packbits压缩.
//papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
//设置压缩比,可以不用,有些时候压缩反而更大,因为是无损的,除非图片很大
//papszOptions = CSLSetNameValue( papszOptions, "ZLEVEL", "9" );
//设置bsq或者BIP存储 bsq:BAND,bip:PIXEL
papszOptions = CSLSetNameValue(papszOptions, "INTERLEAVE", "BAND");
OutDs = poDriver->Create("output.tif",
                         WidthAll,
                         HightAll,
                         BandNum,
                         GDT_Float32,
                         papszOptions);
double geos[6];
//获取第一幅图的地理变化信息，这里如果与原图不同，请自己计算
ds1->GetGeoTransform(geos);
char *pszProjection = NULL;
char *pszPrettyWkt = NULL;
//设置为第一幅图的投影，如果与原图不同，请跳过这个if部分，到下个部分直接设置投影
//设置投影，如果需要特殊投影，请找到wkt串，自行建立投影后传入
OGRSpatialReferenceH  hSRS;

if(GDALGetProjectionRef(ds1) != NULL) {
  pszProjection = (char *) GDALGetProjectionRef(ds1);
  hSRS = OSRNewSpatialReference(NULL);

  if(OSRImportFromWkt(hSRS, &pszProjection) == CE_None) {
    OSRExportToPrettyWkt(hSRS, &pszPrettyWkt, FALSE);
    OutDs->SetProjection(pszPrettyWkt);
  }
}

//pszPrettyWkt实际上是ogc wkt串或者是proj4
//如果指定投影的，可以在http://spatialreference.org/ 网站中搜索
//找到所需要的投影后，例如西安80  3度带，中央经线117E，就是EPSG:2384，点开后
//点击ogc wkt或者proj4，抄下来就行,proj4比较短，而且没有双引号不需要转义，比较简单
//pszPrettyWkt = "+proj=tmerc +lat_0=0 +lon_0=117 +k=1 +x_0=500000
//                +y_0=0 +a=6378140 +b=6356755.288157528 +units=m +no_defs";
//OutDs->SetProjection(pszPrettyWkt);
//设置坐标
OutDs->SetGeoTransform(geos);
CPLFree(pszPrettyWkt);

//写入数据到outds中
for(i = 0; i < BandNum; i++) {
  GDALRasterBand *outBand = OutDs->GetRasterBand(i + 1);
  outBand->RasterIO(GF_Write,
                    0,
                    0,
                    WidthAll,
                    HightAll,
                    outBuf[i],
                    WidthAll,
```

```
                        HightAll,
                        GDT_Float32,
                        0,
                        0);
  }

  //关闭dataset之后，数据才会真正写入到文件中，否则都是在缓存中！！！
  GDALClose(OutDs);
  GDALClose(ds1);
  GDALClose(ds2);

  //清理环境，删除new的数组、变量等
  for(i = 0; i < BandNum; i++) {
    delete[] InBuf1[i];
    delete[] InBuf2[i];
    delete[] outBuf[i];
  }

  delete[] InBuf1;
  delete[] InBuf2;
  delete[] outBuf;
  getchar();
  return 0;
}
```

4.6 压缩文件与网络文件读取

GDAL可以直接读取压缩文件或者网络中的影像数据,不过需要在文件名前加上特定前缀,GDAL工具和函数都可以使用.

4.6.1 压缩文件

GDAL对zip文件有只读支持,对gz文件支持读写,注意随机读写压缩文件效率相当低.

对于tar.gz/.tgz/.tar 等gzip文件来说,打开时,使用如下方式:

```
GDALDataset  *poDataset= (GDALDataset *) GDALOpen( \
                          "/vsigzip/D:/test/1.tgz/aaa/cc.tif",\
                          GA_ReadOnly );//打开文件
```

注意打开时,添加 `/vsigzip/` 前缀

类似,打开zip文件时,添加 `/vsizip/` 前缀:

```
gdalinfo /vsizip/d:/file.zip/test.tif
```

4.6.2 网络文件

在前面添加 `/vsicurl/` 前缀即可:

```
ogrinfo -ro -al -so /vsicurl/http://example.org/gdal/trunk/data/poly.shp
```

混合vsizip:

```
ogrinfo -ro -al -so /vsizip/vsicurl/http://example.com/data/poly.zip
```

用账户访问ftp:

```
ogrinfo -ro -al -so /vsizip/vsicurl/ftp://usr:passwd@a.com/b/c.zip/d.shp
```

4.7 其他处理

可以使用 *GdalDriver* 对已有影像数据进行删除、复制、重命名等处理，可以参见 GdalDriver类 中的具体函数说明，下面使用一些例子来说明：

```
GDALAllRegister();//注册类型，读取写入任何类型影像必须加入此句
GDALDriver *poDriver;
const char *pszFormat ="GTiff";
poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);
poDriver->Delete("TempFile.tif");//删除
poDirver->Rename("newName.tif","oldName.tif");//重命名
poDirver->CopyFiles("Replica.tif","original.tif");//复制
```

CHAPTER 5

GDAL工具集

主要来自 GDAL源码剖析（四）之命令行程序说明一 和 GDAL源码剖析（四）之命令行程序说明二 。对部分翻译进行了少量修改和更新。

5.1 通用选项

所有GDAL工具都可以使用以下选项:

```
--version
    返回GDAL版本.
--license
    返回license信息.
--formats
    返回所有支持的格式.
--format [format]
    某种格式的详细信息.
--optfile filename
    将一个文件中的内容作为工具参数列表.
--config key value
    设置系统参数.
--debug [on/off/value]
    设置debug级别.
--pause
    在出发调试器的时候，等待用户输入
--locale [locale]
    为调试设置本机环境(ie. en_US.UTF-8)
--help-general
    显示通用选项.
```

实例:

```
gdalinfo --help-general
```

5.2 gdalinfo文件信息工具

gdalinfo 主要用来显示栅格数据的所有信息,用法如下:

```
gdalinfo [-mm] [-stats] [-hist] [-nogcp] [-nomd]
         [-norat] [-noct] [-nofl] [-checksum] [-proj4]
         [-mdd domain] [-sd subdataset] datasetname
```

参数说明:

-mm
强制计算栅格每个波段的最大最小值

-stats
显示栅格统计值,如果没有,则强制计算统计值(均值,最大最小值,标准差等)

-approx_stats
显示栅格统计值,如果没有,则强制计算.但是是非精确计算,基于缩略图或者部分数据进行计算,如果不需要精确值或者希望快速返回,请使用此参数代替-stats

-hist
显示所有波段的直方图信息

-nogcp
禁止地面控制点显示,某些数据类型有上千的地面控制点,使用此选项可以禁止

-nomd
禁止元数据显示.有些数据集可能包括很多元数据

-nrat
禁止栅格属性表显示

-noct
禁止颜色表显示

-checksum
强制计算所有波段的校验值

-mdd domain
报告指定域的元数据

-nofl
只显示文件列表中第一个文件

-sd subdataset
如果输入数据集包含几个子数据,读取并显示指定的号码（从1开始）的子数据集。

-proj4 (GDAL >= 1.9.0)
将文件的坐标系用proj4格式的字符串显示

gdalinfo默认显示信息:

```
文件类型
文件大小(以像素表示).
坐标信息(用OGC WKT串表示).
与该文件关联的地理转换
左上右下脚点坐标
地面控制点
元数据
波段数据类型(byte int16...)
Band color interpretations.
波段自动分块大小Band block size.
波段描述Band descriptions.
波段最大最小值Band min/max values (internally known and possibly computed).
波段校验值Band checksum (if computation asked).
波段的NODATA值Band NODATA value.
Band overview resolutions available.
波段长度单位Band unit type (i.e.. "meters" or "feet" for elevation bands).
波段伪彩色表Band pseudo-color tables.
```

实例:

```
gdalinfo 1.tif
```

5.3 gdalwarp 图像纠正工具

gdalwarp 工具是一个图像镶嵌、重投影、和纠正的工具，用法如下:

```
gdalwarp [-s_srs srs_def] [-t_srs srs_def] [-to "NAME=VALUE"]
    [-order n | -tps | -rpc | -geoloc] [-et err_threshold]
    [-refine_gcps tolerance [minimum_gcps]]
    [-te xmin ymin xmax ymax] [-tr xres yres] [-tap] [-ts width height]
    [-wo "NAME=VALUE"] [-ot Byte/Int16/...] [-wt Byte/Int16]
    [-srcnodata "value [value...]"] [-dstnodata "value [value...]"] -dstalpha
    [-r resampling_method] [-wm memory_in_mb] [-multi] [-q]
    [-cutline datasource] [-cl layer] [-cwhere expression]
    [-csql statement] [-cblend dist_in_pixels] [-crop_to_cutline]
    [-of format] [-co "NAME=VALUE"]* [-overwrite]
    [-nomd] [-cvmd meta_conflict_value] [-setci]
    srcfile* dstfile
```

参数说明:

-s_srs srs def
: 设置原始空间参考，坐标系统是可以使用函数OGRSpatialReference.SetFromUserInput()调用的就行，包括EPSG PCS，PROJ4或者后缀名为.prf的wkt文本文件。

-t_srs srs_def
: 设置目标空间参考，坐标系统是可以使用函数OGRSpatialReference.SetFromUserInput()调用的就行，包括EPSG PCS，PROJ4或者后缀名为.prf的wkt文本文件。

-to NAME=VALUE
: 设置转换参数选项，具体选项参考函数GDALCreateGenImgProjTransformer2()支持的选项。

-order n
: 多项式纠正次数（1到3），默认的多项式次数根据输入的GCP点个数自动计算。

-tps
: 强制使用TPS（thin plate spline）纠正方法来纠正图像。

-rpc:
: 强制使用RPC参数纠正。

-geoloc
: 强制使用Geolocation数组。（这个没用过，不太清楚）

-et err_threshold
: 指定变换的近似误差阈值，默认为0.125个像元大小（使用像元为单位）。

-refine_gcps tolerance minimum_gcps
: 细化控制点，自动消除离群值。离群值将被淘汰，直到剩下的控制点为minimum_gcps时或检测不到异常值时。通过调整tolerance，去除离群的GCP。不只适用于多项式插值GCP细化。如果没有投影，tolerance以像素为单位，否则它是SRS单位。如果不设置minimum_gcps，根据多项式模型需要的最少控制点来确定（1次2个点，2次三个点，三次6个点）。

-te xmin ymin xmax ymax
: 设置输出文件的地理范围（在目标空间参考中）。

-tr xres yres
设置输出图像的分辨率。

-tap
(GDAL >= 1.8.0) (target aligned pixels) align the coordinates of the extent of the output file to the values of the -tr, such that the aligned extent includes the minimum extent.

-ts width height
设置输出文件的宽高。如果宽或者高有一个为0，那么将自动计算一个值，注意-ts和-tr不能同时使用。

-wo "NAME=VALUE"
设置纠正选项。具体参考GDALWarpOptions::papszWarpOptions的帮助文档。

-ot type
指定输出波段的数据类型。

-wt type
计算的数据类型。

-r resampling_method
重采样方式，主要有下面几种方式：

- near: 最邻近采样方法（默认值，算法较快，但是质量较差）。
- bilinear: 双线性内插采样。
- cubic: 立方卷积采样。
- cubicspline: 立方样条采样。
- lanczos: Lanczos 窗口辛克采样。
- average: average resampling, computes the average of all non-NODATA contributing pixels. (GDAL >= 1.10.0)
- mode: mode resampling, selects the value which appears most often of all the sampled points. (GDAL >= 1.10.0)

-srcnodata value [value...]
设置输入波段的Nodata值，可以为不同的波段指定不同的值。。如果有多个值，就需要把他们用双引号括起来，以保持在命令参数中作为单一参数输入。掩膜值不会在内插中处理。

-dstnodata value [value...]
设置输出波段的Nodata值。

-dstalpha
创建一个Alpha波段在输出文件中。

-wm memory_in_mb
设置纠正API使用的内存大小，以MB为单位。

-multi
是否使用多线程纠正图像，多线程用来分块处理，同时在读取和写入图像均使用多线程技术。

-q
不在控制台输出提示信息。

-of format
输出文件格式，默认为GeoTiff。

-co "NAME=VALUE"
指定创建图像选项，具体参考不同的格式说明。

-cutline datasource
使用使用OGR支持的矢量数据进行裁切图像。

-cl layername
指定裁切矢量的图层名称。

-cwhere expression
从裁切矢量中根据属性表查询指定的要素来裁切图像。

-csql query
使用SQL语句来从裁切矢量的属性表中查询要素来裁切图像。

-cblend distance
Set a blend distance to use to blend over cutlines (in pixels).（这个参数不太清楚，没用过）

-crop_to_cutline
(GDAL >= 1.8.0) 使用矢量边界的外接矩形大小作为输出影像的范围。

-overwrite
(GDAL >= 1.8.0) 如果结果数据存在，那么覆盖结果数据。

srcfile
输入数据文件名（可以为多个，使用空格隔开）。

dstfile
输出数据文件名。 如果输出文件已经存在，那么镶嵌到这个文件是可以的。但是数据的空间范围等信息不会被修改，如果要修改为新的数据的空间信息，那么需要使用-overwrite选项来覆盖原文件。

5.4 gdal_translate格式转换工具

gdal_translate 主要用来转换数据格式 ,用法如下:

```
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] [-epo] [-eco]
    [-a_srs srs_def] [-a_ullr ulx uly lrx lry] [-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:

-ot: type
输出文件波段数据类型{Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/CInt16/CInt32/CFloat32/CFloat64}中的一个，默认与元数据集相同

-strict
严格模式，如果数据不匹配或者数据丢失，可能导致输出失败

-of format
选择输出格式,默认是GTiff格式

-b band
选择需要输出的波段,波段起始值为1,如果需要多个波段,或者调整波段顺序,请设置多个-b参数，例如 `-b 3 -b 2 -b 1`.

-mask band
选择波段作为掩膜波段

-expand gray|rgb|rgba
将一个带颜色表的波段分解为3个（rgb）或4个波段（rgba）。主要用来输出JPEG, JPEG2000, MrSID, ECW一类不支持颜色索引的数据格式。gray表示将一个含有颜色表的数据集分解为一个灰度索引的数据集。

-outsize xsize[%] ysize[%]
输出文件大小,单位为像素,带%则为相对源图像的比率

-scale [src_min src_max [dst_min dst_max]]
输入像素值的比例范围,从dst_min dst_max到src_min src_max。如果省略了输出范围是0到255。如果省略了输入范围,将自动从源数据计算

-unscale
将波段中使用元数据中的按比例缩放的值转换为实际值，经常与 -ot 选项一起使用来重置输出类型。

-srcwin xoff yoff xsize ysize
从原图中裁切一个子窗口，起点坐标为 xoff,yoff ,长为 xsize ,宽为 ysize 。

-projwin ulx uly lrx lry
从原图中裁切一个子窗口，使用地理投影坐标， ulx,uly 为左上角坐标， lrx,lry 为右下角坐标

-epo
Error when Partially Outside 。如果 -srcwin 或 -projwin 裁剪的子窗口部分超出原图，将提示错误

-eco
Error when Completely Outside 。基本和 -epo 相同，除了如果子窗口完全超出原图，错误提示不同。

-a_srs srs_def
指定输出文件投影，投影字符串可以是wkt、proj4、EPSG号等格式

-a_ullr ulx uly lrx lry
指定输出文件的地理投影边界，使用地理投影坐标， ulx,uly 为左上角坐标， lrx,lry 为右下角坐标

-a_nodata value
指定nodata值。

-mo "META-TAG=VALUE"
设置元数据

-co "NAME=VALUE"
设置创建选项，，参考 创建数据集 一节代码示例，可以设置多个。

-gcp pixel line easting northing elevation
设置GCP点，可以设置多个

-q
静默输出，禁止显示进度条和非错误信息

-sds
将每个子数据集到单独的输出文件中。主要用于HDF等有子数据集的数据转换

-stats
强制重计算统计信息

src_dataset
源数据集名称

dst_dataset
输出数据集名称

5.5 gdalmanage文件管理工具

gdalmanage 主要用来管理栅格数据,用法如下:

```
gdalmanage mode [-r] [-u] [-f format]
              datasetname [newdatasetname]
```

参数说明:

mode
操作模式

- identify datasetname: 显示数据格式
- copy datasetname newdatasetname: 拷贝数据
- rename datasetname newdatasetname: 重命名数据
- delete datasetname: 删除数据

-r
递归扫描文件/文件夹

-u
如果文件无法执行操作，显示操作失败信息

-f format
指定数据格式，例如GTiff

datasetname
需要操作的文件名

newdatasetname
复制或重命名操作时，需要设置此参数。

实例:

```
gdalmanage identify -r 50m_raster/

gdalmanage copy NE1_50M_SR_W.tif ne1_copy.tif

gdalmanage rename NE1_50M_SR_W.tif ne1_rename.tif

gdalmanage identify NE1_50M_SR_W.tif
```

5.6 gdallocationinfo像元查询工具

gdallocationinfo 通过指定一种坐标系中的坐标来进行对其位置的像元值进行输出显示

用法如下:

```
gdallocationinfo [--help-general] [-xml] [-lifonly] [-valonly]
                 [-b band]* [-overview overview_level]
                 [-l_srs srs_def] [-geoloc] [-wgs84]
                 srcfile [x y]
```

参数说明:

--help-general
显示帮助

-xml
输出xml信息

-lifonly
输出LocationInfo 中的信息。

-valonly
仅输出指定位置的每个波段的像元值

-l_srs srs def
指定输入的x,y坐标的坐标系

-geoloc
表示输入的x,y坐标是地理参考坐标系

-wgs84
表示输入的x,y是WGS84坐标系下的经纬度坐标

srcfile
输入栅格图像的名称

x
查询的X坐标。默认为图像列号，如果使用-l_srs,-wgs84或者-geoloc时按照指定的坐标来处理

y
查询的Y坐标。默认为图像行号，如果使用-l_srs,-wgs84或者-geoloc时按照指定的坐标来处理

该工具的目的是输出一个像素的各种信息。目前支持下面四项:

- 像素的行列号。
- 输出元数据中的LocationInfo 信息，目前只支持VRT文件。
- 全部波段或子文件中的所有波段中的像元值。
- 未缩放的像元值，如果对波段进行了缩放和偏移操作。

输入x和y坐标是，是通过命令行来进行输入的，一般是先x后y，即先列号后行号；如果使用-geoloc, -wgs84, 或-l_srs 选项后，会自动根据输入进行交换。默认的输出信息是纯文本文件，也可以使用-xml选项将其使用xml格式进行输出。将来会添加其他的信息。

实例:

```
gdallocationinfo utm.tif 256 256
Report:
  Location: (256P,256L)
  Band 1:
    Value: 115
```

5.7 gdaltransform坐标系转换工具

gdaltransform 坐标系转换工具,用来转换坐标，从支持的投影，包括GCP点的变换。

用法:

```
gdaltransform [--help-general]
              [-i] [-s_srs srs_def] [-t_srs srs_def] [-to "NAME=VALUE"]
              [-order n] [-tps] [-rpc] [-geoloc]
              [-gcp pixel line easting northing [elevation]]*
              [srcfile [dstfile]]
```

参数说明:

-s_srs srs def
原始空间参考，该值将被OGRSpatialReference.SetFromUserInput()函数调用，支持包括EPSG PCS和GCSes (如EPSG:4296), PROJ.4声明或者一个后缀名为.prf的文件存储的WKT串。

-t_srs srs_def
目标空间参考，该值将被OGRSpatialReference.SetFromUserInput()函数调用，支持包括EPSG PCS和GCSes (如EPSG:4296), PROJ.4声明或者一个后缀名为.prf的文件存储的WKT串。

-to NAME=VALUE
设置转换参数，将会使函数GDALCreateGenImgProjTransformer2()来调用。

-order n
几何多项式变换次数（1到3），默认情况下会根据输入的GCP点的个数自动确定。

-tps
强制将GCP点使用TPS转换方式转换。

-rpc
强制使用RPC参数转换。

-geoloc
强制使用Geolocation数组转换。

-i
逆转换，从目标到原始投影转换。

-gcppixel line easting northing [elevation]
指定GCP点，通常至少需要三个。

srcfile
原始投影下定义的GCP点文件，如果没有指定，需要从命令行-s_srs或者-gcp参数输入。

dstfile
目标投影定义文件。

读取的坐标必须是成对出现（或者三个一组），输出也以同样的方式输出，所有的变换共gdalwarp相同，包括使用GCP的变换。注意输入输出必须使用十进制小数，当前不支持DMS（度分秒）的输入输出。如果指定了输入图像，那么输入的坐标是基于图像的行列号，如果输出图像指定，输出的坐标也是图像的行列号。

实例:

```
下面的命令行使用基于RPC变换，并且使用-i选项来将经纬度坐标转为图像的行列号：
gdaltransform -i -rpc 06OCT20025052-P2AS-005553965230_01_P001.TIF
125.67206 39.85307 50
输出：
3499.49282422381 2910.83892848414 50
```

5.8 gdalsrsinfo格式化SRS工具

gdalsrsinfo 将给定的SRS按照不同的格式显示,用法如下:

```
gdalsrsinfo [--help-general/-h] [-p] [-V] [-o] srs_def
```

参数说明:

--help-general/-h
显示帮助

-p
格式化输出信息 (e.g. WKT)

-v
验证 SRS

-o out_type
输出类型 { default, all, wkt_all, proj4, wkt, wkt_simple, wkt_noct, wkt_esri, mapinfo, xml }

srs_def
可以是一个GDAL或者OGR支持的文件，或者是任何GDAL和OGR支持的SRS格式（包括WKT,PROJ4,EPSG:n等）

实例:

```
gdalsrsinfo   "EPSG:4326"

PROJ.4 : '+proj=longlat +datum=WGS84 +no_defs '

OGC WKT :
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
```

CHAPTER 6

GDAL工具集代码实现(C/C++)

GDAL2.0 开始，可以使用代码实现GDAL工具集的功能，即可以使用代码调用gdal工具集，可以提升编码效率。需要注意,使用参数时,需要逐个添加,不能一行按照空格添加。

下面主要介绍 `gdalinfo` `gdalwarp` `gdal_translate` `ogr2ogr` 几个常用工具的代码实现,其他可用工具见 GDAL官网gdal_utils.h: GDAL Utilities C API 说明,包括 `gdaldem` `gdal_rasterize` `nearblack` `gdal_grid` `gdalbuildvrt` 几个工具，其余工具官方正在添加。

使用前,需要添加以下头文件:

```
#include "gdal_priv.h"
#include "cpl_string.h"
#include "gdal_utils.h"
```

6.1 gdalinfo

*gdalinfo*文件信息工具 工具功能可以使用 `GDALInfo` 函数实现，具体定义如下:

```
/**
* @param hDataset    需要获取信息的数据集
* @param psOptions   gdalinfo命令参数
* @return 返回数据集信息
*/
char* GDALInfo       (       GDALDatasetH    hDataset,
const GDALInfoOptions *      psOptions
)
```

基本使用方式如下:

```
GDALAllRegister();
GDALDatasetH TestDs = GDALOpen( "testfile.tiff", GA_ReadOnly );
char* info = GDALInfo(TestDs,NULL);
std::cout<<info<<std::endl;
//do something
GDALClose(TestDs);
```

如果需要使用命令行参数,则需要创建 `GDALInfoOptions` 代码如下:

```
GDALAllRegister();
//添加命令参数
char **argv = NULL;
argv = CSLAddString( argv, "-json");
argv = CSLAddString( argv, "-hist");
```

```
GDALInfoOptions *opt = GDALInfoOptionsNew(argv,NULL);
GDALDatasetH TestDs = GDALOpen( "testfile.tiff", GA_ReadOnly );
char* info = GDALInfo(TestDs,opt);
GDALInfoOptionsFree(opt);
std::cout<<info<<std::endl;
//do something
GDALClose(TestDs);
```

6.2 gdalwarp

gdalwarp 图像纠正工具 工具功能可以使用 GDALWarp 函数实现，具体定义如下:

```
/**
* @param pszDest       输出路径,或者NULL
* @param hDstDS        输出数据集或为NULL(前两参数必须选一)
* @param nSrcCount     输入数据集个数
* @param pahSrcDS      输入数据集列表
* @param psOptionsIn   命令行参数
* @param pbUsageError 返回值,标识错误信息
* @return  输出数据集(如果hDstDS为NULL,返回的数据集必须使用GDALClose关闭)
*/
GDALDatasetH GDALWarp       (
    const char *    pszDest,
    GDALDatasetH    hDstDS,
    int     nSrcCount,
    GDALDatasetH *  pahSrcDS,
    const GDALWarpAppOptions *      psOptionsIn,
    int *   pbUsageError
)
```

基本使用方式如下:

```
GDALAllRegister();
//添加命令参数,每次添加一个!!!
char **argv = NULL;
argv = CSLAddString( argv, "-order" );
argv = CSLAddString( argv, "3" );
argv = CSLAddString( argv, "-ts" );
argv = CSLAddString( argv, "1000" );
argv = CSLAddString( argv, "1000" );

//错误实例!!!!
//error!!!!
//////argv = CSLAddString( argv, "-order 3 -ts 1000 1000" );//此写法错误,按照上面写!!!!
//error!!!!

//返回
int bUsageError = FALSE;
//输入列表
GDALDatasetH TestDs = GDALOpen("test.tif", GA_ReadOnly );
//gdalwarp
GDALWarpAppOptions *opt = GDALWarpAppOptionsNew( argv, NULL );
GDALDataset *dst = ( GDALDataset * )GDALWarp( "out.tif", NULL, 1,\
                   &TestDs ,opt, &bUsageError );
GDALWarpAppOptionsFree( opt );
CSLDestroy( argv );
```

```
//do something
//clear env
GDALClose(dst);
GDALClose(TestDs);
```

如果有多个文件输入,使用如下代码:

```
GDALAllRegister();
//添加命令参数,每次添加一个!!!
char **argv = NULL;
argv = CSLAddString( argv, "-order" );
argv = CSLAddString( argv, "3" );
argv = CSLAddString( argv, "-ts" );
argv = CSLAddString( argv, "1000" );
argv = CSLAddString( argv, "1000" );

//错误实例!!!!
//error!!!!
//////argv = CSLAddString( argv, "-order 3 -ts 1000 1000" );//此写法错误,按照上面写!!!!
//error!!!!

//返回
int bUsageError = FALSE;
//输入列表
GDALDatasetH TestDs = GDALOpen("test.tif", GA_ReadOnly );
GDALDatasetH *srcList = NULL;
srcList = ( GDALDatasetH * ) CPLRealloc( srcList, sizeof( GDALDatasetH ) * 1 );
srcList[0] = TestDs;//有多少写多少
//gdalwarp
GDALWarpAppOptions *opt = GDALWarpAppOptionsNew( argv, NULL );
GDALDataset *dst = ( GDALDataset * )GDALWarp( "out.tif", NULL, 1,\
                   srcList ,opt, &bUsageError );
GDALWarpAppOptionsFree( opt );
CSLDestroy( argv );
//do something
//clear env
GDALClose(dst);
GDALClose(TestDs);
CPLFree(srcList);
```

6.3 gdal_translate

*gdal_translate*格式转换工具 工具功能可以使用 GDALTranslate 函数实现，具体定义如下:

```
/**
* @param pszDest      输出路径
* @param hSrcDataset  输入数据集
* @param psOptionsIn  命令行参数,可以为空
* @param pbUsageError 返回值,标识错误信息
* @return  输出数据集(如果hDstDS为NULL,返回的数据集必须使用GDALClose关闭)
*/
GDALDatasetH GDALTranslate  (
    const char *    pszDest,
    GDALDatasetH    hSrcDataset,
    const GDALTranslateOptions *    psOptionsIn,
    int *   pbUsageError
```

```
)
```

基本使用方式如下:

```
GDALAllRegister();
//添加命令参数,每次添加一个!!!
char **argv = NULL;
argv = CSLAddString( argv, "-ot" );
argv = CSLAddString( argv, "UInt16" );

//错误实例!!!!
//error!!!!
//////argv = CSLAddString( argv, "-ot UInt16" );//此写法错误,按照上面写!!!!
//error!!!!

//返回
int bUsageError = FALSE;
//输入列表
GDALDatasetH TestDs = GDALOpen( "test.tif", GA_ReadOnly );
//GDALTranslate
GDALTranslateOptions *opt =  GDALTranslateOptionsNew( argv, NULL );
GDALDataset *dst = ( GDALDataset * )GDALWarp( "out.tif", TestDs ,opt, &bUsageError );
GDALTranslateOptionsFree( opt );
CSLDestroy( argv );
```

6.4 ogr2ogr

ogr2ogr 工具功能可以使用 GDALVectorTranslate 函数实现，具体定义如下:

```
/**
* @param pszDest       输出路径,或者NULL
* @param hDstDS        输出数据集或为NULL(前两参数必须选一)
* @param nSrcCount     输入数据集个数(至2.1.1为止,只能有1个)
* @param pahSrcDS      输入数据集列表
* @param psOptionsIn   命令行参数
* @param pbUsageError 返回值,标识错误信息
* @return  输出数据集(如果hDstDS为NULL,返回的数据集必须使用GDALClose关闭)
*/
GDALDatasetH GDALVectorTranslate    (
    const char *    pszDest,
    GDALDatasetH    hDstDS,
    int     nSrcCount,
    GDALDatasetH *  pahSrcDS,
    const GDALVectorTranslateOptions *      psOptionsIn,
    int *   pbUsageError
)
```

基本使用方式如下:

```
GDALAllRegister();
//添加命令参数,每次添加一个!!!
char **argv = NULL;
argv = CSLAddString( argv, "-f" );
argv = CSLAddString( argv, "GML" );

//错误实例!!!!
```

```
//error!!!!
//////argv = CSLAddString( argv, "-f GML" );//此写法错误,按照上面写!!!!
//error!!!!

//返回
int bUsageError = FALSE;
//输入列表
GDALDatasetH TestDs = GDALOpenEx("test.shp",GDAL_OF_VECTOR, , NULL, NULL, NULL );
//gdalwarp
GDALVectorTranslateOptions *opt = GDALVectorTranslateOptionsNew( argv, NULL );
GDALDataset *dst = ( GDALDataset * )GDALVectorTranslate( "out.gml", NULL, 1,
                    &TestDs ,opt, &bUsageError );
GDALVectorTranslateOptionsFree( opt );
CSLDestroy( argv );
//do something
//clear env
GDALClose(dst);
GDALClose(TestDs);
```

CHAPTER 7

GDAL Cheat Sheet

本文为 GitHub 上的项目，原项目地址： https://github.com/dwtkns/gdal-cheat-sheet

7.1 Vector operations

Get vector information

```
ogrinfo -so input.shp layer-name
```

Or, for all layers

```
ogrinfo -al -so input.shp
```

Print vector extent

```
ogrinfo input.shp layer-name | grep Extent
```

List vector drivers

```
ogr2ogr --formats
```

Convert between vector formats

```
ogr2ogr -f "GeoJSON" output.json input.shp
```

Clip vectors by bounding box

```
ogr2ogr -f "ESRI Shapefile" output.shp input.shp -clipsrc
        <x_min> <y_min> <x_max> <y_max>
```

Clip one vector by another

```
ogr2ogr -clipsrc clipping_polygon.shp output.shp input.shp
```

Reproject vector:

```
ogr2ogr output.shp -t_srs "EPSG:4326" input.shp
```

Merge features in a vector file by attribute ("dissolve")

```
ogr2ogr -f "ESRI Shapefile" dissolved.shp input.shp -dialect sqlite -sql
  "select ST_union(Geometry),common_attribute from input GROUP BY common_attribute"
```

Merge vector files:

```
ogr2ogr merged.shp input1.shp
ogr2ogr -update -append merged.shp input2.shp -nln merged
```

Extract from a vector file based on query

To extract features with STATENAME 'New York','New Hampshire', etc. from states.shp

```
ogr2ogr -where 'STATENAME like "New%"' states_subset.shp states.shp
```

To extract type 'pond' from water.shp

```
ogr2ogr -where "type = pond" ponds.shp water.shp
```

Subset & filter all shapefiles in a directory

Assumes that filename and name of layer of interest are the same...

```
ls -1 *.shp | sed 's/.shp//g' | xargs -n1 -I % ogr2ogr %-subset.shp %.shp
   -sql "SELECT field-one, field-two FROM '%' WHERE field-one='value-of-interest'"
```

7.2 Raster operations

Get raster information

```
gdalinfo input.tif
```

List raster drivers

```
gdal_translate --formats
```

Convert between raster formats

```
gdal_translate -of "GTiff" input.grd output.tif
```

Convert 16-bit bands (Int16 or UInt16) to Byte type (Useful for Landsat 8 imagery...)

```
gdal_translate -of "GTiff" -co "COMPRESS=LZW" -scale 0 65535 0 255
    -ot Byte input_uint16.tif output_byte.tif
```

You can change '0' and '65535' to your image's actual min/max values to preserve more color variation or to apply the scaling to other band types - find that number with:

```
gdalinfo -mm input.tif | grep Min/Max
```

Convert a directory of files to a different raster format

```
ls -1 *.img | sed 's/.img//g' | xargs -n1 -I % gdal_translate
        -of "GTiff" %.img %.tif
```

Reproject raster:

```
gdalwarp -t_srs "EPSG:102003" input.tif output.tif
```

Be sure to add *-r bilinear* if reprojecting elevation data to prevent funky banding artifacts.

Georeference an unprojected image with known bounding coordinates:

```
gdal_translate -of GTiff -a_ullr <top_left_lon> <top_left_lat>
    <bottom_right_lon> <bottom_right_lat> -a_srs EPSG:4269 input.png output.tif
```

Clip raster by bounding box

```
gdalwarp -te <x_min> <y_min> <x_max> <y_max> input.tif clipped_output.tif
```

Clip raster to SHP / NoData for pixels beyond polygon boundary

```
gdalwarp -dstnodata <nodata_value> -cutline input_polygon.shp
    input.tif clipped_output.tif
```

Crop raster dimensions to vector bounding box

```
gdalwarp -cutline cropper.shp -crop_to_cutline input.tif cropped_output.tif
```

Merge rasters

```
gdal_merge.py -o merged.tif input1.tif input2.tif
```

Alternatively,

```
gdalwarp input1.tif input2.tif merged.tif
```

Or, to preserve nodata values:

```
gdalwarp input1.tif input2.tif merged.tif -srcnodata <nodata_value>
    -dstnodata <merged_nodata_value>
```

Stack grayscale bands into a georeferenced RGB

Where LC81690372014137LGN00 is a Landsat 8 ID and B4, B3 and B2 correspond to R,G,B bands respectively:

```
gdal_merge.py -co "PHOTOMETRIC=RGB" -separate LC81690372014137LGN00_B{4,3,2}.tif
    -o LC81690372014137LGN00_rgb.tif
```

Fix an RGB TIF whose bands don't know they're RGB

```
gdal_merge.py -co "PHOTOMETRIC=RGB" input.tif -o output_rgb.tif
```

Export a raster for Google Earth

```
gdal_translate -of KMLSUPEROVERLAY input.tif output.kmz -co FORMAT=JPEG
```

Raster calculation (map algebra)

Average two rasters:

```
gdal_calc.py -A input1.tif -B input2.tif --outfile=output.tif --calc="(A+B)/2"
```

Add two rasters:

```
gdal_calc.py -A input1.tif -B input2.tif --outfile=output.tif --calc="A+B"
```

etc.

Create a hillshade from a DEM

```
gdaldem hillshade -of PNG input.tif hillshade.png
```

Change light direction:

```
gdaldem hillshade -of PNG -az 135 input.tif hillshade_az135.png
```

Use correct vertical scaling in meters if input is projected in degrees

```
gdaldem hillshade -s 111120 -of PNG input_WGS1984.tif hillshade.png
```

Apply color ramp to a DEM First, create a color-ramp.txt file: *(Height, Red, Green, Blue)*

```
0 110 220 110
900 240 250 160
1300 230 220 170
1900 220 220 220
2500 250 250 250
```

Then apply those colors to a DEM:

```
gdaldem color-relief input.tif color_ramp.txt color-relief.tif
```

Create slope-shading from a DEM First, make a slope raster from DEM:

```
gdaldem slope input.tif slope.tif
```

Second, create a color-slope.txt file: *(Slope angle, Red, Green, Blue)*

```
0 255 255 255
90 0 0 0
```

Finally, color the slope raster based on angles in color-slope.txt:

```
gdaldem color-relief slope.tif color-slope.txt slopeshade.tif
```

Resample (resize) raster

```
gdalwarp -ts <width> <height> -r cubicspline dem.tif resampled_dem.tif
```

Entering 0 for either width or height guesses based on current dimensions.

Burn vector into raster

```
gdal_rasterize -b 1 -i -burn -32678 -l layername input.shp input.tif
```

Create contours from DEM

```
gdal_contour -a elev -i 50 input_dem.tif output_contours.shp
```

Get values for a specific location in a raster

```
gdallocationinfo -xml -wgs84 input.tif <lon> <lat>
```

7.2.1 Other

Convert KML points to CSV (simple)

```
ogr2ogr -f CSV output.csv input.kmz -lco GEOMETRY=AS_XY
```

Convert KML to CSV (WKT) First list layers in the KML file

```
ogrinfo -so input.kml
```

Convert the desired KML layer to CSV

```
ogr2ogr -f CSV output.csv input.kml -sql "select *,OGR_GEOM_WKT from some_kml_layer"
```

CSV points to SHP *This section needs retooling* Given input.csv

```
lon_column,lat_column,value
-81,32,13
-81,32,14
-81,32,15
```

Make a .dbf table for ogr2ogr to work with from input.csv

```
ogr2ogr -f "ESRI Shapefile" input.dbf input.csv
```

Use a text editor to create a .vrt file in the same directory as input.csv and input.dbf. This file holds the parameters for building a full shapefile based on values in the DBF you just made.

```
<OGRVRTDataSource>
  <OGRVRTLayer name="output_file_name">
    <SrcDataSource relativeToVRT="1">./</SrcDataSource>
    <SrcLayer>input</SrcLayer>
    <GeometryType>wkbPoint</GeometryType>
    <LayerSRS>WGS84</LayerSRS>
    <GeometryField encoding="PointFromColumns" x="lon_column" y="lat_column"/>
  </OGRVRTLayer>
</OGRVRTDataSource>
```

Create shapefile based on parameters listed in the .vrt

```
mkdir shp
ogr2ogr -f "ESRI Shapefile" shp/ inputfile.vrt
```

The VRT file can be modified to give a new output shapefile name, reference a different coordinate system (LayerSRS), or pull coordinates from different columns.

MODIS operations

First, download relevant .hdf tiles from the MODIS ftp site: ftp://ladsftp.nascom.nasa.gov/; use the MODIS sinusoidal grid for reference.

Create a file containing the names of all .hdf files in the directory

```
ls -1 *.hdf > files.txt
```

List MODIS Subdatasets in a given HDF (conf. the MODIS products table)

```
gdalinfo longFileName.hdf | grep SUBDATASET
```

Make TIFs from each file in list; replace 'MOD12Q1:Land_Cover_Type_1' with desired Subdataset name

```
mkdir output
cat files.txt | xargs -I % -n1 gdalwarp -of GTiff
    'HDF4_EOS:EOS_GRID:%:MOD12Q1:Land_Cover_Type_1' output/%.tif
```

Merge all .tifs in output directory into single file

```
cd output
gdal_merge.py -o Merged_Landcover.tif *.tif
```

BASH functions

Size Functions

This size function echos the pixel dimensions of a given file in the format expected by gdalwarp.

```
function gdal_size() {
    SIZE=$(gdalinfo $1 |\
        grep 'Size is ' |\
        cut -d\   -f3-4 |\
        sed 's/,//g')
    echo -n "$SIZE"
}
```

This can be used to easily resample one raster to the dimensions of another:

```
gdalwarp -ts $(gdal_size bigraster.tif) -r cubicspline
    smallraster.tif resampled_smallraster.tif
```

Extent Functions

These extent functions echo the extent of the given file in the order/format expected by gdal_translate -projwin. (Originally from Linfiniti).

```
function gdal_extent() {
    if [ -z "$1" ]; then
        echo "Missing arguments. Syntax:"
        echo "  gdal_extent <input_raster>"
        return
    fi
    EXTENT=$(gdalinfo $1 |\
        grep "Upper Left\|Lower Right" |\
        sed "s/Upper Left  //g;s/Lower Right //g;s/).*//g" |\
        tr "\n" " " |\
        sed 's/ *$//g' |\
        tr -d "[(,]")
    echo -n "$EXTENT"
}

function ogr_extent() {
    if [ -z "$1" ]; then
        echo "Missing arguments. Syntax:"
        echo "  ogr_extent <input_vector>"
        return
    fi
    EXTENT=$(ogrinfo -al -so $1 |\
        grep Extent |\
        sed 's/Extent: //g' |\
        sed 's/(//g' |\
        sed 's/)//g' |\
        sed 's/ - /, /g')
    EXTENT=`echo $EXTENT | awk -F ',' '{print $1 " " $4 " " $3 " " $2}'`
    echo -n "$EXTENT"
}

function ogr_layer_extent() {
    if [ -z "$2" ]; then
        echo "Missing arguments. Syntax:"
        echo "  ogr_extent <input_vector> <layer_name>"
        return
    fi
    EXTENT=$(ogrinfo -so $1 $2 |\
        grep Extent |\
        sed 's/Extent: //g' |\
        sed 's/(//g' |\
```

```
        sed 's/)//g' |\
        sed 's/ - /, /g')
    EXTENT=`echo $EXTENT | awk -F ',' '{print $1 " " $4 " " $3 " " $2}'`
    echo -n "$EXTENT"
}
```

Extents can be passed directly into a gdal_translate command like so:

```
gdal_translate -projwin $(ogr_extent boundingbox.shp) input.tif clipped_output.tif
```

or

```
gdal_translate -projwin $(gdal_extent target_crop.tif) input.tif clipped_output.tif
```

This can be a useful way to quickly crop one raster to the same extent as another. Add these to your ~/.bash_profile file for easy terminal access.

7.3 Sources

http://live.osgeo.org/en/quickstart/gdal_quickstart.html

https://github.com/nvkelso/geo-how-to/wiki/OGR-to-reproject,-modify-Shapefiles

ftp://ftp.remotesensing.org/gdal/presentations/OpenSource_Weds_Andre_CUGOS.pdf

http://developmentseed.org/blog/2009/jul/30/using-open-source-tools-make-elevation-maps-afghanistan-and-pakistan/

http://linfiniti.com/2010/12/a-workflow-for-creating-beautiful-relief-shaded-dems-using-gdal/

http://linfiniti.com/2009/09/clipping-rasters-with-gdal-using-polygons/

http://nautilus.baruch.sc.edu/twiki_dmcc/bin/view/Main/OGR_example

http://www.gdal.org/frmt_hdf4.html

http://planetflux.adamwilson.us/2010/06/modis-processing-with-r-gdal-and-nco.html

http://trac.osgeo.org/gdal/wiki/FAQRaster

http://www.mikejcorey.com/wordpress/2011/02/05/tutorial-create-beautiful-hillshade-maps-from-digital-elevation-models-with-gdal-and-mapnik/

http://dirkraffel.com/2011/07/05/best-way-to-merge-color-relief-with-shaded-relief-map/

http://gfoss.blogspot.com/2008/06/gdal-raster-data-tips-and-tricks.html

http://osgeo-org.1560.x6.nabble.com/gdal-dev-Dissolve-shapefile-using-GDAL-OGR-td5036930.html

https://www.mapbox.com/tilemill/docs/guides/terrain-data/

https://gist.github.com/ashaw/0862ec044c45b9aa3c76

https://github.com/gina-alaska/dans-gdal-scripts

CHAPTER 8

GDALWarp

本章来源:

- GDAL源码剖析（十二）之GDAL Warp API使用说明
- GDALWarpOptions Struct Reference
- GDAL_Algorithms_C_API

顺序重新调整,增加GDAL算法介绍，本章是进阶章节，阅读前请务必先熟悉 *GDAL数据模型* 与 栅格数据读写 章节。

8.1 GDALWarp简介

GDAL Warp API（在文件 `gdalwarper.h` 中定义）是一个高效的进行图像变换的接口。主要由几何变换函数（ `GDALTransformerFunc` ），多种图像重采样方式，掩码操作选项等组成。这个接口可以对很大的图像进行处理。

使用步骤如下:

- 在程序中，首先要初始化一个 `GDALWarpOptions` 结构体的对象
- 然后使用 `GDALWarpOptions` 的对象来初始化 `GDALWarpOperation` 的对象
- 最后通过调用 `GDALWarpKernel` 类里面的 `GDALWarpOperation::ChunkAndWarpImage()` 或 `GDALWarpOperation::ChunkAndWarpMulti()` 函数来完成图像的变换

下面先介绍 `GDALWarpOptions` 和 `GDALTransformerFunc` ，然后对完整流程进行解析。

8.2 GDALWarpOptions选项介绍

`GDALWarpOptions` 是对图像变换进行设置的类，成员见下:

class GDALWarpOptions
GDALWarpOptions结构体中包含了很多参数来对变换进行设置。下面对一些比较重要的进行列举说明:

char ****papszWarpOptions**
这个是一个字符串列表，用来设置图像变换过程中的一些选项，样式为 `NAME=VALUE` ，可以使用 `CSLSetNameValue()` 设置。现在支持的一些键值对设置为:

- INIT_DEST=[value] 或者 INIT_DEST=NO_DATA：这个选项用来强制设置结果图像的初始值（所有的波段），初始值为指定的value，或者NODATA值。NODATA值从参数padfDstNoDataReal或者参数padfDstNoDataImag中获取。如果这个值没有设置，那么将会使用原始图像的NODATA值来覆盖。
- WRITE_FLUSH=YES/NO：这个选项用来强制设置在处理完每一块后将数据写入磁盘中。在某些时候，这个选项可以更加安全的写入结果数据，但是同时会增加更多的磁盘操作。目前这个默认值为NO。
- SKIP_NOSOURCE=YES/NO :跳过没有相应输入的数据块的处理。这将禁止初始化要写入的数据块以及其他所有处理，要谨慎使用。常用于影像拼接、镶嵌情况下，减少额外工作。
- UNIFIED_SRC_NODATA=YES/[NO] :默认情况下，每个波段的 nodata 掩膜数据是独立的。但有些时候所有波段的 nodata 值一致情况下，可以将此参数设置为YES。

通常在计算一个数据的部分区域时，变换样区在输出区域每个边界上转换21个点到源文件中，然后在源图中计算一个足够大的窗口。这样做是为了使转换更加高效，但某些情况下，这种做法计算的窗口过小，甚至丢失大量的区域。特别是对于非线性或者是翻转的转换。例如对极地点附近的区域，从WGS84转到Polar Stereographic投影转换，或者某些根本无法进行投影转换的数据。对于这种情况，GDAL提供以下键值对设置来防止窗口计算错误：

- SAMPLE_GRID=YES/NO : 如果设置为 YES ,将强制采样样区边缘点和中心点。对于变形很大的投影或者是大部分数据变换后不在源图像上的情况非常适用。
- SAMPLE_STEPS : 调整采样密度，默认为21，增加可以增强精确度，但是会降低计算效率。
- SOURCE_EXTRA : 该数值是边界增加的像素，默认为1，保证边界不出错，设置更大的画，将增加读取数量，但是可以避免丢失源数据。
- CUTLINE：裁切线，使用wkt表示。与在 GDALWarpOptions hCutline 中设置相同，在 GDALWarpOperation::Initialize() 调用时将被设置到 hCutline 属性中。与 gdalwarp 工具不同，坐标系为图像的坐标系。
- CUTLINE_BLEND_DIST : 与 dfCutlineBlendDist 属性相同
- CUTLINE_ALL_TOUCHED : 默认为 FALSE ，但是可以设置为 TURE ，保证与裁切线相交的像素会被裁剪，而不仅仅是完全被包含的像素。
- OPTIMIZE_SIZE : 默认为 FALSE ，设置为 TRUE 时，在压缩格式中将获取更小的文件，可能导致转换变慢。
- NUM_THREADS : (GDAL >= 1.10) 设置为一个数值，或者是 ALL_CPUS 可以设置多线程，使用并行化处理图像变换。没有设置默认为单线程。

double **dfWarpMemoryLimit**

设置GDALWarpOperation在处理图像中使用的最大内存数。单位为比特，默认值比较保守（比较小），可以根据自己的内存大小来进行调整这个值。增加这个值可以帮助提高程序的运行效率，但是需要注意内存的大小。这个大小、GDAL的缓存大小，还有你的应用程序以及系统所需要的内存加起来要小于系统的内存，否则会导致错误。一般来说，比如一个内存为256MB的系统，这个值最少设置为64MB比较合理。注意，这个值不包括GDAL读取数据使用的内存。

GDALResampleAlg **eResampleAlg**

重采样的算法选择：

- GRA_NearestNeighbour： 最邻近插值
- GRA_Bilinear： 双线性插值 (2x2 kernel)
- GRA_Cubic： 立方卷积逼近 (4x4 kernel)
- GRA_CubicSpline：三次样条线逼近(4x4 kernel)
- GRA_Lanczos： Lanczos windowed sinc 插值 (6x6 kernel)

- GRA_Average： 均值 (计算样区内非空值的所有像素均值)
- GRA_Mode： 众数 (出现频数最高的值)

GDALDataType **eWorkingDataType**
计算时用的数据类型，GDT_Unknown 为程序自动判断

GDALDatasetH **hSrcDS**
源数据集

GDALDatasetH **hDstDS**
目标数据集

int **nBandCount**
所需转换的波段数，0为所有波段

int ***panSrcBands**
int数组，源图像中所需处理的波段，从1开始

int ***panDstBands**
int数组，输出的波段

int **nSrcAlphaBand**
源数据中的Alpha波段，0为禁止

int **nDstAlphaBand**
目标数据中的Alpha波段，0为禁止

double ***padfSrcNoDataReal**
源数据中 nodata 的实部，NULL表示没有

double ***padfSrcNoDataImag**
源数据中 nodata 的虚部，NULL表示没有

double ***padfDstNoDataReal**
目标数据中 nodata 的实部，NULL表示没有

double ***padfDstNoDataImag**
目标数据中 nodata 的虚部，NULL表示没有

GDALProgressFunc **pfnProgress**
进度条显示函数，NULL表示没有

void ***pProgressArg**
传给pfnProgress的回调参数

GDALTransformerFunc **pfnTransformer**
几何变换函数

void ***pTransformerArg**
几何变换函数参数

void ***hCutline**
裁切线，OGRPolygonH 类型

double **dfCutlineBlendDist**
切割线宽容度，默认为零

8.3 几何变换函数简介

GDALTransformerFunc 是一个函数签名，用来计算空间点几何变换，也是 `GDALWarp` 的核心部分，定义在GDAL算法文件中。 `GDAL` 中可用的算法有：

- GDALApproxTransform 近似变换
- GDALGCPTransform GCP投影变换
- GDALGenImgProjTransform 图像几何变换
- GDALReprojectionTransform 重投影变换
- GDALTPSTransform TPS变换
- GDALRPCTransform RPC 变换
- GDALGeoLocTransform GeoLoc 变换

与 GDALTransformerFunc 对应，默认有几个函数生成几何变换函数参数：

- GDALCreateApproxTransformer 对应生成近似变换参数
- GDALCreateGCPTransformer 对应生成GCP投影变换参数
- GDALCreateGenImgProjTransformer 对应生成图像几何变换参数
- GDALCreateGenImgProjTransformer2 对应生成图像几何变换参数
- GDALCreateGenImgProjTransformer3 对应生成图像几何变换参数
- GDALCreateReprojectionTransformer 对应生成重投影变换参数
- GDALCreateTPSTransformer 对应生成TPS变换参数
- GDALCreateRPCTransformer 对应生成 RPC 变换投影变换参数
- GDALCreateGeoLocTransformer 对应生成 GeoLoc 投影变换参数

这部分看起来比较复杂，实际使用时，比较简便。只需要在 GDALWarpOptions->pfnTransformer 中指定算法名称， GDALWarpOptions->pTransformerArg 中指定对应的算法参数函数即可，如下代码所示：

```
psWarpOptions->pfnTransformer = GDALGenImgProjTransform;
psWarpOptions->pTransformerArg =
    GDALCreateGenImgProjTransformer(pSrcDs,
                                    sFromWkt,
                                    pDstDs,
                                    sToWkt,
                                    FALSE, 0.0, 3);
```

每个Transformer的所需参数不同，请参见 GDAL Algorithms C API ，近似计算有些特殊，在使用Transformer时要传递几何变换函数，参考 性能优化 部分最后一条。

8.4 完整流程

下面介绍使用 GDALWarp 提供的接口，实现几何变换的完整流程：

8.4.1 创建输出文件

几何变换后输出文件的范围和投影仿射参数都将发生变化，输出的文件大小可能是已知，也可能是未知的。因此创建输出文件方式也有两种：

- 已知的例如重采样或者裁剪，我们知道输出文件大小，需要自己计算文件大小
- 未知例如投影，我们GDAL中提供计算函数，我们将使用 GDALSuggestedWarpOutput 函数预测输出文件大小和仿射变换，使用 GDALDriver->Create() 函数创建输出文件

重采样创建文件示例:

```
GDALAllRegister();
//打开输入文件
GDALDataset *pDSrc = (GDALDataset *)GDALOpen(InFile, GA_ReadOnly);

if(pDSrc == NULL) {
    return -1;
}
//创建Driver
GDALDriver *pDriver = GetGDALDriverManager()->GetDriverByName("GTiff");

if(pDriver == NULL) {
    GDALClose((GDALDatasetH) pDSrc);
    return -2;
}
//获取源数据中的投影/波段数/仿射变换等信息
int inBandCount = pDSrc->GetRasterCount();
const char* strWkt = pDSrc->GetProjectionRef();
GDALDataType dataType = pDSrc->GetRasterBand(1)->GetRasterDataType();
double geos[6] = {0};
pDSrc->GetGeoTransform(geos);
//获取源数据中左上角点地理坐标Img2CoordX/Img2CoordY为图片到投影坐标转换函数
double lefttopx = Img2CoordX(geos,0,0);
double lefttopy = Img2CoordY(geos,0,0);
//计算新高度和宽度,以及投影信息
int newWidth =(int) pDSrc->GetRasterXSize()*geos[1]/fResX;
int newHeight =(int) fabs(pDSrc->GetRasterYSize()*geos[5]/fResY);
geos[0] = lefttopx;
geos[1] = fResX;
geos[3] = lefttopy;
geos[5] = -fResY;
int *pSrcBand = NULL;
int *pDstBand = NULL;
int iBandSize = 0;
//创建输出文件
GDALDataset *pDDst = pDriver->Create(OutFile,
                                     newWidth,
                                     newHeight,
                                     iBandSize,
                                     dataType,
                                     NULL);
pDDst->SetProjection(strWkt);
pDDst->SetGeoTransform(geos);
```

投影变换创建文件示例:

```
//打开文件
GDALAllRegister();
GDALDataset *pProjDS = (GDALDataset *)GDALOpen(ProjFile, GA_ReadOnly);

if(pProjDS == NULL) {
    return -2;    //打开坐标文件失败
}
const char *sToWkt = ...//输出的投影

//输出文件
GDALDataset *pDstDs;
//创建Driver
```

```
GDALDriver* pDriver = (GDALDriver*)GDALGetDriverByName("GTiff");

if(pDriver == NULL) {
    return -3;    //注册driver失败
}

//计算输出投影参数

const char *sFromWkt = GDALGetProjectionRef(pProjDS);
CPLAssert(sFromWkt != NULL &&strlen(sFromWkt) > 0);

//计算变换参数
void *pTransformArg;
pTransformArg = GDALCreateGenImgProjTransformer(pProjDS,
                                                 sFromWkt,
                                                 NULL,
                                                 sToWkt,
                                                 FALSE,
                                                 0,
                                                 3);
CPLAssert(pTransformArg != NULL);
double geos[6];
int nPixels=0, nLines=0;
CPLErr eErr;
eErr = GDALSuggestedWarpOutput(pProjDS,
                               GDALGenImgProjTransform,
                               pTransformArg,
                               geos,
                               &nPixels,
                               &nLines);
CPLAssert(eErr == CE_None);
GDALDestroyGenImgProjTransformer(pTransformArg);
//输出数据类型
GDALDataType eDataType= GDALGetRasterDataType(GDALGetRasterBand(pProjDS,1));
//计算创建波段数量
int nCreateBandCount = pProjDS->GetRasterCount();

//创建文件
pDstDs = (GDALDataset*)GDALCreate(pDriver,
                                  sOutFileName,
                                  nPixels,
                                  nLines,
                                  nCreateBandCount,
                                  eDataType,
                                  NULL);
GDALSetProjection(pDstDs, sToWkt);
GDALSetGeoTransform(pDstDs, geos);
```

8.4.2 设置选项

选项设置参见 *GDALWarpOptions选项介绍* ，设置一般需要设置输入输出数据集，需要操作的波段，几何变换函数和参数，进度条，优化参数这几项。示例如下：

```
//warp 选项
GDALWarpOptions *psWarpOptions = GDALCreateWarpOptions();
//源
```

```
psWarpOptions->hSrcDS = pSrcDs;
psWarpOptions->hDstDS = pDstDs;
//优化选项
psWarpOptions->dfWarpMemoryLimit = 512000000;
char** papszWarpOptions=NULL;
papszWarpOptions = CSLSetNameValue(papszWarpOptions,"NUM_THREADS","ALL_CPUS");
papszWarpOptions = CSLSetNameValue(papszWarpOptions,"WRITE_FLUSH","YES");
psWarpOptions->papszWarpOptions=papszWarpOptions;
//源和输出对应的波段
psWarpOptions->nBandCount = nCreateBandCount;
psWarpOptions->panSrcBands = (int *) CPLMalloc(nCreateBandCount*sizeof(int));
psWarpOptions->panDstBands = (int *) CPLMalloc(nCreateBandCount*sizeof(int));
int i;

for(i=0; i<nCreateBandCount; i++) {
    psWarpOptions->panSrcBands[i] = pSrcBand[i];
    psWarpOptions->panDstBands[i] = pDstBand[i];
}

delete[] pSrcBand;
delete[] pDstBand;
//进度条
psWarpOptions->pfnProgress = GDALTermProgress;
// 几何变换函数和参数
psWarpOptions->pfnTransformer = GDALGenImgProjTransform;
psWarpOptions->pTransformerArg =
    GDALCreateGenImgProjTransformer(pSrcDs,
                                    sFromWkt,
                                    pDstDs,
                                    sToWkt,
                                    FALSE, 0.0, 3);
```

8.4.3 执行变换

执行较为简单，新建 GDALWarpOperation ,调用 Initialize 函数，然后调用 ChunkAndWarpMulti 或者 ChunkAndWarpImage ，代码一般如下：

```
GDALWarpOperation opt;
if(opt.Initialize(psWarpOptions)!=CE_None) {
    GDALClose(pSrcDs);
    GDALClose(outDs);
    return -1;
}
opt.ChunkAndWarpMulti(0, 0, CutWidth, CutHeight);
```

8.4.4 清理环境

完成后，需要释放变换参数、删除数据集。代码如下：

```
GDALDestroyGenImgProjTransformer(psWo->pTransformerArg);
GDALDestroyWarpOptions(psWo);
GDALClose((GDALDatasetH) pDSrc);
GDALClose((GDALDatasetH) pDDst);
delete[] pSrcBand;
delete[] pDstBand;
```

8.4.5 完整代码

完整的重采样代码如下：

```
/**
图像坐标转地理坐标x
    @param geos    变换参数
    @param x       图像X坐标
    @param y       图像Y坐标
    @return        地理X坐标
 */
double Img2CoordX(const double *geos, double x, double y)
{
  return geos[0] + geos[1] * x + geos[2] * y;
}
/**
图像坐标转地理坐标y
    @param geos    变换参数
    @param x       图像X坐标
    @param y       图像Y坐标
    @return        地理Y坐标
 */
double Img2CoordY(const double *geos, double x, double y)
{
  return geos[3] + geos[4] * x + geos[5] * y;
}

/**
    图像重采样
    @param InFile        输入文件
    @param OutFile       输出文件
    @param fResX         x分辨率
    @param fResY         Y分辨率
    @param iResampleAlg 重采样方式,默认为2
                            GRA_NearestNeighbour=0,
                            GRA_Bilinear=1,
                            GRA_Cubic=2,
                            GRA_CubicSpline=3,
                            GRA_Lanczos=4,
                            GRA_Average=5,
                            GRA_Mode=6
    @param pBandIndex    重采样的波段，默认为NULL，全部波段重采样
    @param pnBandCount    重采样波段总数，默认为0 ，全部波段重采样
*/
int Resample(const char *InFile , const char *OutFile, float fResX ,
             float fResY, int iResampleAlg, int *pBandIndex, int pnBandCount)
{
  GDALAllRegister();
  GDALResampleAlg eResample = (GDALResampleAlg)iResampleAlg;
  //打开文件
  GDALDataset *pDSrc = (GDALDataset *)GDALOpen(InFile, GA_ReadOnly);

  if(pDSrc == NULL) {
    return -1;
  }

  GDALDriver *pDriver = GetGDALDriverManager()->GetDriverByName("GTiff");
```

```
if(pDriver == NULL) {
  GDALClose((GDALDatasetH) pDSrc);
  return -2;
}

//计算输出投影参数
int inBandCount = pDSrc->GetRasterCount();
const char *strWkt = pDSrc->GetProjectionRef();
GDALDataType dataType = pDSrc->GetRasterBand(1)->GetRasterDataType();
double geos[6] = {0};
pDSrc->GetGeoTransform(geos);
double lefttopx = Img2CoordX(geos, 0, 0);
double lefttopy = Img2CoordY(geos, 0, 0);
int newWidth = (int) pDSrc->GetRasterXSize() * geos[1] / fResX;
int newHeight = (int) fabs(pDSrc->GetRasterYSize() * geos[5] / fResY);
geos[0] = lefttopx;
geos[1] = fResX;
geos[3] = lefttopy;
geos[5] = -fResY;
//输入输出波段对应
int *pSrcBand = NULL;
int *pDstBand = NULL;
int iBandSize = 0;

if(pBandIndex != NULL && pnBandCount != NULL) {
  iBandSize = pnBandCount;
  pSrcBand = new int[iBandSize];
  pDstBand = new int[iBandSize];

  for(int i = 0; i < iBandSize; i++) {
    pSrcBand[i] = pBandIndex[i];
    pDstBand[i] = i + 1;
  }
} else {
  iBandSize = inBandCount;
  pSrcBand = new int[iBandSize];
  pDstBand = new int[iBandSize];

  for(int i = 0; i < iBandSize; i++) {
    pSrcBand[i] = i + 1;
    pDstBand[i] = i + 1;
  }
}

//创建输出文件
GDALDataset *pDDst = pDriver->Create(OutFile,
                                     newWidth,
                                     newHeight,
                                     iBandSize,
                                     dataType,
                                     NULL);
pDDst->SetProjection(strWkt);
pDDst->SetGeoTransform(geos);
//参数选项
GDALWarpOptions *psWo = GDALCreateWarpOptions();
psWo->papszWarpOptions = CSLDuplicate(NULL);
//数据源
psWo->eWorkingDataType = dataType;
```

```
  psWo->eResampleAlg = eResample;
  psWo->hSrcDS = (GDALDatasetH) pDSrc;
  psWo->hDstDS = (GDALDatasetH) pDDst;
  //波段
  psWo->nBandCount = iBandSize;
  psWo->panSrcBands = (int *) CPLMalloc(iBandSize * sizeof(int));
  psWo->panDstBands = (int *) CPLMalloc(iBandSize * sizeof(int));

  for(int i = 0; i < iBandSize; i++) {
    psWo->panSrcBands[i] = pSrcBand[i];
    psWo->panDstBands[i] = pDstBand[i];
  }

  //变换函数设置
  void *hTransformArg = NULL;
  hTransformArg =  GDALCreateGenImgProjTransformer2(
                     (GDALDatasetH) pDSrc,
                     (GDALDatasetH) pDDst,
                     NULL);
  GDALTransformerFunc pFnTransformer = GDALGenImgProjTransform;
  psWo->pfnTransformer = pFnTransformer;
  psWo->pTransformerArg = hTransformArg;
  //进度条
  psWo->pfnProgress = GDALTermProgress;
  //优化选项
  psWo->dfWarpMemoryLimit = 512000000;
  char **papszWarpOptions = NULL;
  papszWarpOptions = CSLSetNameValue(papszWarpOptions, "NUM_THREADS", "ALL_CPUS");
  papszWarpOptions = CSLSetNameValue(papszWarpOptions, "WRITE_FLUSH", "YES");
  psWo->papszWarpOptions = papszWarpOptions;
  //开始转换
  GDALWarpOperation oWo;

  if(oWo.Initialize(psWo) != CE_None) {
    GDALClose((GDALDatasetH) pDSrc);
    GDALClose((GDALDatasetH) pDDst);
    return -4;
  }

  oWo.ChunkAndWarpMulti(0, 0, newWidth, newHeight);
  //清理环境
  GDALDestroyGenImgProjTransformer(psWo->pTransformerArg);
  GDALDestroyWarpOptions(psWo);
  GDALClose((GDALDatasetH) pDSrc);
  GDALClose((GDALDatasetH) pDDst);
  delete[] pSrcBand;
  delete[] pDstBand;
  return 0;
}
```

8.5 性能优化

下面几点可以在使用变换API的时候提高处理效率。

- 增加变换API使用的内存，可以使程序执行的更快。这个参数叫GDALWarpOptions::dfWarpMemoryLimit。理论上，越大的块对于数据I/O效率越高，并且变换

的效率也会越高。然而，变换所需的内存和GDAL缓存应该小于机器的内存，最多是内存的2/3左右。

- 增加GDAL的缓存。这个尤其对于在处理非常大的输入图像很有用。如果所有的输入输出图像的数据频繁的读取会极大的降低运行效率。使用函数GDALSetCacheMax()来设置GDAL使用的最大缓存。
- 当写入一个空的输出文件，在GDALWarpOptions::papszWarpOptions 对象中使用INIT_DEST选项来进行设置。这样可以很大的减少磁盘的IO操作。
- 输入和输出文件使用分块存储的文件格式。分块文件格式可以快速的访问一块数据，相比来说，普通的大数据量的顺序存储文件格式在IO操作中要花费更多的时间。
- 在一次调用中处理所有的波段。一次处理所有的波段比每次处理一个波段要保险。
- 使用GDALWarpOperation::ChunkAndWarpMulti()方法来代替GDALWarpOperation::ChunkAndWarpImage()方法。这个使用多个独立的线程来进行IO操作和变换影像处理，能够提高CPU和IO的效率。执行这个操作，GDAL需要多线程的支持（从GDAL1.8.0开始，默认在Win32平台、Unix平台是支持的，对于之前的版本，需要在配置中进行修改才行）。
- 重采样方式，最邻近象元执行速度最快，双线性内插次之，三次立方卷次最慢。不要使用根据复杂的重采样函数。
- 避免使用复杂的掩码选项。比如，最邻近采样在处理没有掩码的8bit数据要比一般的效率高很多。
- 使用近似的变换代替精确的变换（精确的变换是每个象元都会计算一次）。下面的代码用来说明近似变换的使用方式，近似变换要求指定一个变换的误差dfErrorThreshold，这个误差单位是输出图像的象元个数

```
hTransformArg =  GDALCreateApproxTransformer(GDALGenImgProjTransform,
                                             hGenImgProjArg,
                                             dfErrorThreshold );
pfnTransformer = GDALApproxTransform;
```

CHAPTER 9

静态编译

分享一下完全的静态编译过程，栅格数据格式加入 hdf4 、 hdf5 、 netcdf 、 openJPEG 、 webp 静态编译，矢量数据格式加入编译 libexpat 和 libcurl spatialite， gdal 是最新 svn 中编译的，其他库都是最新 release 版本。

这样编译出来的 gdal 只有 gdal 本身的 dll 和 lib ，不需要添加其他的 dll ，比较方便一些，且比加入其他的 dll 要小，有问题请直接在最下评论。

动态编译请参考 GDAL源码剖析（二）之编译说明

9.1 编译环境

编译所需的软件:

- cmake 根据netcdf的要求是2.8.10以上版本，用于编译 expat hdf4 hdf5 openJPEG
- vs2010 ，一般都使用命令行编译
- nsis 2.46版， 用于生成 hdf4 hdf5 安装文件

推荐编译顺序： libexpat libcurl hdf4 hdf5 netcdf4 geos proj.4 libsqlite3 libwebp openJPEG gdal

编译顺序除了注意事项中所描述的，一般可以互换。

netcdf hdf4 hdf5 spatialite 这几个库编译比较复杂，请注意编译条件的修改。

虽然 cmake 可以使用 GUI ，推荐都使用命令行编译，命令行开头我将以 > 标识出来,本文都是在 Windows 下编译， Linux 下没有尝试。

Warning:
- libexpat库静态编译必须设置 XML_STATIC 宏
- openJPEG库静态编译后需要设置 OPJ_STATIC 宏
- 如果需要编译netcdf，libcurl库必须在netcdf前编译
- netcdf与hdf4如果都需要的话，编译hdf4时必须关闭netcdf设置
- geos库最好不用svn的版本
- 编译都假设起始目录为代码源目录
- 不要使用 nsis3.0a1 版本，无法生成安装文件
- hdf5是只读驱动,但是netcdf也算hdf5格式,netcdf是读写驱动

Attention: 我编译netcdf时没有加入hdf4设置，加入后无法通过编译，如果有人解决，请在评论中留言

9.2 libexpat

9.2.1 编译前准备

下载源码后，修改 `expat-2.1.0/lib/expat.h` 文件,在文件首加上:

```
#define XML_STATIC
```

这一步也可以在最后的生成include中修改，只要能链接就可以。

打开vs2010命令行，依次输入:

```
mkdir build
cd build
cmake -G   "Visual Studio 10" -DBUILD_shared=OFF
      -DCMAKE_INSTALL_PREFIX=E:\BUILD\expat ..
cmake --build . --config Release --target INSTALL
```

生成文件将在 `E:/BUILD/expat` 路径下

9.3 libcurl

libcurl更新比较快,1-2月更新一个版本,请选择最新的版本,可选 `openssl` 和 `zlib` 等库,最简单的编译方式如下:

```
cd winbuild
nmake /f Makefile.vc mode=static
```

生成文件将在 `builds` 文件夹下.

9.4 hdf4

hdf4最终版本是 `4.2.11` ,更新较为缓慢

编辑 `/config/cmake/cacheinit.cmake` 文件:

设置静态库,7行:

```
SET (BUILD_SHARED_LIBS OFF CACHE BOOL "Build Shared Libraries" FORCE)
```

关闭 `fortan` 编译,15行:

```
SET (HDF4_BUILD_FORTRAN OFF CACHE BOOL "Build FORTRAN support" FORCE)
```

关闭 `netcdf` 支持，如果在 `gdal` 中不同时使用 `netcdf` ，可以不修改，23行:

```
SET (HDF4_ENABLE_NETCDF OFF CACHE BOOL "Build HDF4 versions of NetCDF-3 APIS" FORCE)
```

编译 `zlib` 库和 `szip` 库,使用svn中的代码(需要联网)，49行:

```
SET (HDF4_ALLOW_EXTERNAL_SUPPORT "SVN" CACHE STRING  \
    "Allow External Library Building" FORCE)
```

在 `vs2010` 命令行工具中，依次输入:

```
mkdir build
cd build
cmake -G "Visual Studio 10"  -C ..\config\cmake\cacheinit.cmake ..
cmake --build . --config Release
copy /B .\bin\Release\libjpeg.lib .\bin\libjpeg.lib
copy /B .\bin\Release\libzlib.lib .\bin\libzlib.lib
copy /B .\bin\Release\libszip.lib .\bin\libszip.lib
cmake --build . --config Release
cpack -C Release CPackConfig.cmake
HDF-4.2.9-win32.exe
```

Attention:

- `cmake --build . --config Release` 运行了两次，因为hdf库的cmake写的有些问题，需要把 `libjpeg` 等库先拷贝到上一层才能完成全部的编译。
- 如果需要编译64位的话，第三行需要修改为： `cmake -G "Visual Studio 10 Win64" -C ../config/cmake/cacheinit.cmake ..`

9.5 hdf5

编辑 `/config/cmake/cacheinit.cmake` 文件:

编译静态库，7行:

```
SET (BUILD_SHARED_LIBS OFF CACHE BOOL "Build Shared Libraries" FORCE)
```

关闭 `fortran` 库编译，17行:

```
SET (HDF5_BUILD_FORTRAN OFF CACHE BOOL "Build FORTRAN support" FORCE)
```

`zlib` 和 `szip` 库支持，63行:

```
SET (HDF5_ALLOW_EXTERNAL_SUPPORT "SVN" CACHE STRING \
    "Allow External Library Building" FORCE)
```

Attention:

- 网络不好的情况下,可以在 HDF5官网 下载szip和zlib库,放在hdf5文件夹下,然后修改第63行左右为: SET (HDF5_ALLOW_EXTERNAL_SUPPORT “TGZ” CACHE STRING “Allow External Library Building” FORCE)

在 `vs2010` 命令行工具中，依次输入:

```
mkdir build
cd build
cmake -G "Visual Studio 10"  -C ../config/cmake/cacheinit.cmake ..
cmake --build . --config Release
copy /B .\bin\Release\libzlib.lib .\bin\libzlib.lib
copy /B .\bin\Release\libszip.lib .\bin\libszip.lib
cmake --build . --config Release
cpack -C Release CPackConfig.cmake
```

Attention:

- 注意 `cmake --build . --config Release` 运行了两次，因为hdf库的cmake写的有些问题，需要把 `libzlib` 等库先拷贝到上一层才能完成全部的编译。
- 如果需要编译64位的话，第三行需要修改为： `cmake -G "Visual Studio 10 Win64" -C ../config/cmake/cacheinit.cmake ..`

Warning:
- hdf5.1.8.13 版本静态编译有问题,没有特殊需求不要使用,若使用,请删除或注释/hdf5-1.8.13/src/H5.c 第841行以下的部分

9.6 netcdf

netcdf4.3.0直接按照说明文件可以编译通过,如下所述:

使用 cmake 编译，自己根据情况修改库和 include , -D 表示配置

注意 cmake 命令是一整行

```
mkdir build

cd build

cmake -G "Visual Studio 10" -DCMAKE_INSTALL_PREFIX=e:/build/netcdf
    -DENABLE_NETCDF_4=ON -D"CURL_LIBRARY=E:/BUILD/libcurl/lib/libcurl_a.lib"
    -D"CURL_INCLUDE_DIR=E:/BUILD/libcurl/include"  -D"BUILD_SHARED_LIBS=OFF"
    -D"HDF5_LIB=E:/BUILD/hdf/1.8.11/lib/libhdf5.lib"
    -D"HDF5_HL_LIB=E:/BUILD/hdf/1.8.11/lib/libhdf5_hl.lib"
    -D"HDF5_INCLUDE_DIR=E:/BUILD/hdf/1.8.11/include"
    -D"ZLIB_LIBRARY=E:/BUILD/hdf/1.8.11/lib/libzlib.lib;
    E:/BUILD/hdf/1.8.11/lib/libszip.lib"
    -D"ZLIB_INCLUDE_DIR=E:/BUILD/hdf5-1.8.11/build/ZLIB-prefix/src/ZLIB"
    -D"SZIP_INCLUDE_DIR=E:/BUILD/hdf5-1.8.11/build/SZIP-prefix/src/SZIP/src"
    -D"SZIP_DIR=E:\BUILD\hdf5-1.8.11\build\SZIP-prefix\src\SZIP-build"
    -D"USE_SZIP=ON"    ..

 cmake --build . --config Release --target INSTALL
```

完整的命令解释:

```
cmake -G "Visual Studio 10"                                          #使用vs2010编译
      -DCMAKE_INSTALL_PREFIX=e:/build/netcdf                          #安装路径
      -DENABLE_NETCDF_4=ON                                            #编译netcdf4#
      -D"CURL_LIBRARY=E:/BUILD/libcurl/lib/libcurl_a.lib"             #curl库路径
      -D"CURL_INCLUDE_DIR=E:/BUILD/libcurl/include"                   #curl库头文件路径
      -D"BUILD_SHARED_LIBS=OFF"                                       #静态库
      -D"HDF5_LIB=E:/BUILD/hdf/1.8.11/lib/libhdf5.lib"                #hdf5库
      -D"HDF5_HL_LIB=E:/BUILD/hdf/1.8.11/lib/libhdf5_hl.lib"          #hdf5库
      -D"HDF5_INCLUDE_DIR=E:/BUILD/hdf/1.8.11/include"                #hdf5库头文件
                                                                      #zlib库和szip库文件
      -D"ZLIB_LIBRARY=E:/BUILD/hdf/1.8.11/lib/libzlib.lib;
        E:/BUILD/hdf/1.8.11/lib/libszip.lib"
                                                                      #zlib库头文件
      -D"ZLIB_INCLUDE_DIR=E:/BUILD/hdf5-1.8.11/build/ZLIB-prefix/src/ZLIB"
                                                                      #szip库头文件
      -D"SZIP_INCLUDE_DIR=E:/BUILD/hdf5-1.8.11/build/SZIP-prefix/src/SZIP/src"
                                                                      #szip源文件
      -D"SZIP_DIR=E:\BUILD\hdf5-1.8.11\build\SZIP-prefix\src\SZIP-build"
      -D"USE_SZIP=ON"                                                 #使用sizp库
      ..                              #编译目录中cmakelist.ext文件，在上级目录中。
```

Attention:
- 如果需要编译64位的话，第一行需要修改为： `cmake -G "Visual Studio 10 Win64"`

Important:
- netcdf4.4.0 windows的cmake build有问题,需要修改cmake文件,详细介绍如下:

netcdf4.4.0 的 *cmake* windows版本有问题,详细信息可参见 github issue #222

先需要修改CMakeLists.txt,在498行 ELSE前面,加入‘‘INCLUDE_DIRECTORIES(${HDF5_INCLUDE_DIR})‘‘

再修改编译命令,将‘‘HDF5_HL_LIB‘‘修改为‘‘HDF5_HL_LIBRARY‘‘,‘‘HDF5_LIB‘‘改为‘‘HDF5_C_LIBRARY‘‘,具体如下:

```
mkdir build

cd build

cmake -G "Visual Studio 10" -DCMAKE_INSTALL_PREFIX=d:/GDAL/netcdf4.4.0
   -D"HDF5_DIR=E:/lib/hdf5-1.8.16/build"
   -D"ZLIB_LIBRARY=D:/GDAL/HDF5-1.8.16-win32/lib/libzlib.lib;
    D:/GDAL/HDF5-1.8.16-win32/lib/libszip.lib"
   -D"ZLIB_INCLUDE_DIR=E:/BUILD/hdf5-1.8.16/build/ZLIB-prefix/src/ZLIB"
   -D"SZIP_INCLUDE_DIR=E:/BUILD/hdf5-1.8.16/build/SZIP-prefix/src/SZIP/src"
   -DENABLE_NETCDF_4=ON
   -D"CURL_LIBRARY=D:/GDAL/libcurl-vc-x86-7.47.1/lib/libcurl_a.lib"
   -D"CURL_INCLUDE_DIR=D:/GDAL/libcurl-vc-x86-7.47.1/include"
   -D"HDF5_C_LIBRARY=D:/GDAL/HDF5-1.8.16-win32/lib/hdf5.lib"
   -D"HDF5_HL_LIBRARY=D:/GDAL/HDF5-1.8.16-win32/lib/hdf5_hl.lib"
   -D"HDF5_INCLUDE_DIR=D:/GDAL/HDF5-1.8.16-win32/include"
   -D"BUILD_SHARED_LIBS=OFF"    ..
```

9.7 geos

请直接下最新的 `release` 编译， `svn` 中部分存在问题，编译不过， `geos` 直接采用 `nmake` 可以生成静态库和动态库，在 `src` 子目录下。

```
nmake /f Makefile.vc
```

9.8 proj.4

修改 `nmake.opt` 文件中32、33行:

```
# Uncomment the first for linking exes against DLL or second for static
#EXE_PROJ =    proj_i.lib
EXE_PROJ =    proj.lib
```

以及安装目录 `INSTDIR` ,然后开始编译即可

```
nmake /f makefile.vc
nmake /f makefile.vc install-all
```

9.9 libsqlite3

- 下载 sqlite3 源码，放入 src 目录中。
- 下载 sqliteCmake ,放在 src 目录上一层
- 使用cmake编译

如下所示，设置安装路径和静态库即可

```
mkdir build
cd build
cmake -G "Visual Studio 10" ..
cmake --build . --config Release --target INSTALL
```

Attention:
- 如果需要编译64位的话，第三行需要修改为： cmake -G "Visual Studio 10 Win64" ..

9.10 libwebp

静态库，输出在 output/release-static/x86 中。

```
nmake /f Makefile.vc CFG=release-static RTLIBCFG=static OBJDIR=output
```

Attention:
- 如果需要支持WINDOWS XP 的话,libwebp版本不能超过0.4.4,从libwep 0.5开始,不支持windows xp系统

9.11 openJPEG

```
mkdir build
cd build
cmake -G   "Visual Studio 10" .. -DBUILD_SHARED_LIBS=OFF
                                  -DCMAKE_INSTALL_PREFIX=f:/gdal/openjpeg
                                  -DBUILD_THIRDPARTY=ON
cmake --build . --config Release --target INSTALL
```

完成后，添加 #define OPJ_STATIC 1 到输出的 inlcude/openjpeg-2.0/openjpeg.h 里

9.12 pcre

```
mkdir build
cd build
cmake -G   "Visual Studio 10" ..
cmake --build . --config Release --target INSTALL
```

完成后需要修改‘‘pcre.h‘‘,添加‘‘#define PCRE_STATIC‘‘

9.13 spatialite

spatialite 库可以让ogr中使用更多sql函数，方便矢量操作，但是编译比较复杂，依赖很多，需要依次编译 libiconv FreeXL libxml2 zlib sqlite3 geos PROJ.4 ，后四个库 zlib sqlite3 geos PROJ.4 已经在前面编译完成，可以直接使用，其中 zlib 在 hdf5 中编译，然后依次按顺序编译。

9.13.1 libiconv

首先需要编译 libiconv 库， libiconv 库在windows下一般是编译不过的，有两种方案，一种是使用旧版本的iconv库，1.11版本应该是可以的，另一种是改源代码然后编译。 我直接使用 iconv for windows ， 里面有 vs2010 工程文件，可以直接编译。

编译完成后，将 iconv.obj 文件拷至 iconv 的库目录中,并重命名为 lib.obj ,后续过程中需要使用

9.13.2 FreeXL

下载 FreeXL 库，修改其中的 makefile.vc 文件中 iconv 的 include 和 lib 路径,然后编译

```
nmake /f Makefile.vc
nmake /f makefile.vc install
```

9.13.3 libxml2

下载 libxml2 库，转到目录 win32/VC10 下,使用 libxml2.sln 工程编译

- 编译前,先进入 vs2010 命令行工具中，在 win32 目录下,运行 cscript configure.js help
- 按照提示,添加iconv的include目录和iconv目录，运行 cscript configure.js compiler=msvc prefix=c:\libxml2 include=c:\iconv\include lib=c:\iconv\lib ，此时在命令行中是无法编译通过的(不做这一步后面sln工程可能无法编译通过)
- 进入 win32/VC10 目录中,打开工程文件
- 修改 include``和 ``lib 路径
- 需要添加 WIN32 宏，在 配置->C/C++ ->预处理器->预处理器定义 中添加
- 修改部分文件中 XMLChar 问题,将其在函数最前面声明,后面使用即可.
- 开始编译

9.13.4 spatialite

下载 spatialite 库，编辑 nmake.opt 文件和 makefile.vc 文件中各个库的路径和头文件路径,然后在 vs2010 命令行工具中输入:

```
nmake /f Makefile.vc
nmake /f makefile.vc install
```

编译完成后，将 sqlite 的头文件拷贝至 include/spatialite 文件夹中，后续编译将会使用。

9.14 gdal

修改nmake.opt文件,注意按照自己实际情况修改

```
//303行
# Uncomment for Expat support (required for KML, GPX and GeoRSS read support).
#EXPAT_DIR = "C:\Program Files\Expat 2.0.1"
#EXPAT_INCLUDE = -I$(EXPAT_DIR)/source/lib
#EXPAT_LIB = $(EXPAT_DIR)/bin/libexpat.lib
//改为
# Uncomment for Expat support (required for KML, GPX and GeoRSS read support).
EXPAT_DIR =E:\BUILD\lib\expat
EXPAT_INCLUDE = -I$(EXPAT_DIR)/include
EXPAT_LIB = E:\BUILD\lib\expat\lib\expat.lib

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

//331行
# Uncomment the following 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
//改为
# Uncomment the following and update to enable NCSA HDF Release 4 support.
HDF4_PLUGIN = NO
HDF4_DIR =    E:\BUILD\lib\4.2.9
HDF4_LIB =    $(HDF4_DIR)\lib\libhdf.lib $(HDF4_DIR)\lib\libmfhdf.lib  \
      $(HDF4_DIR)\lib\libxdr.lib $(HDF4_DIR)\lib\libjpeg.lib  Ws2_32.lib

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

//336行
# Uncomment the following 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
//改为
# Uncomment the following and update to enable NCSA HDF Release 5 support.
HDF5_PLUGIN = NO
HDF5_DIR =    E:\BUILD\lib\1.8.11
HDF5_LIB =    $(HDF5_DIR)\lib\libhdf5.lib $(HDF5_DIR)\lib\libhdf5_hl.lib \
     $(HDF5_DIR)\lib\libszip.lib $(HDF5_DIR)\lib\libzlib.lib

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

//387行
# SQLite Libraries
#SQLITE_INC=-IN:\pkg\sqlite-win32
#SQLITE_LIB=N:\pkg\sqlite-win32\sqlite3_i.lib
//改为
# SQLite Libraries
SQLITE_INC=-If:\GDAL\libsqlite3\include
SQLITE_LIB=f:\GDAL\libsqlite3\lib\sqlite3.lib

//如果是使用spatialite,那么需要修改上面的注释
# SQLite Libraries
#SQLITE_INC=-IN:\pkg\sqlite-win32
```

```
#SQLITE_LIB=N:\pkg\sqlite-win32\sqlite3_i.lib
# For spatialite support, try this instead (assuming you grab the \
    libspatialite-amalgamation-2.3.1 and installed it in osgeo4w):
# The -DSPATIALITE_AMALGAMATION, which cause "spatialite/sqlite3.h" \
    to be included instead of "sqlite3.h" might not be necessary
# depending on the layout of the include directories. In case of compilation errors,\
    remove it.
#SQLITE_INC=-IC:\osgeo4w\include -DHAVE_SPATIALITE -DSPATIALITE_AMALGAMATION
#SQLITE_LIB=C:\osgeo4w\lib\spatialite_i.lib
# Uncomment following line if libsqlite3 has been compiled with \
    SQLITE_HAS_COLUMN_METADATA=yes
#SQLITE_HAS_COLUMN_METADATA=yes
# Uncomment following line if spatialite is 4.1.2 or later
#SPATIALITE_412_OR_LATER=yes
//改为
# SQLite Libraries
#SQLITE_INC=-ID:\GDAL\sqlite3.13\include
#SQLITE_LIB=D:\GDAL\sqlite3.13\lib\sqlite3-static.lib
# For spatialite support, try this instead
# (assuming you grab the libspatialite-amalgamation-2.3.1 and installed it in osgeo4w):
# The -DSPATIALITE_AMALGAMATION, which cause "spatialite/sqlite3.h"  \
    to be included instead of "sqlite3.h" might not be necessary
# depending on the layout of the include directories. In case of compilation errors,\
    remove it.
SQLITE_INC=-IC:\OSGeo4w\libspatialite\include \
           -DHAVE_SPATIALITE -DSPATIALITE_AMALGAMATION
SQLITE_LIB=C:\OSGeo4w\libspatialite\lib\spatialite.lib \
    "C:\OSGeo4w\libspatialite\lib\sqlite3-static.lib"  \
    "C:\OSGeo4w\libspatialite\lib\libxml2.lib" \
    "C:\OSGeo4w\libspatialite\lib\iconv.lib"
# Uncomment following line if libsqlite3 has been compiled \
    with SQLITE_HAS_COLUMN_METADATA=yes
#SQLITE_HAS_COLUMN_METADATA=yes
# Uncomment following line if spatialite is 4.1.2 or later
SPATIALITE_412_OR_LATER=yes

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

//401行
# PCRE Library (REGEXP support for SQLite) for example from \
    http://sourceforge.net/projects/gnuwin32/files/pcre/7.0/pcre-7.0.exe/download
#PCRE_INC=-I"C:\Program Files\GNUWin32\include" -DHAVE_PCRE
#PCRE_LIB="C:\Program Files\GNUWin32\lib\pcre.lib"
//改为
# PCRE Library (REGEXP support for SQLite) for example from \
    http://sourceforge.net/projects/gnuwin32/files/pcre/7.0/pcre-7.0.exe/download
PCRE_INC=-I"e:\BUILD\lib\PCRE\include" -DHAVE_PCRE
PCRE_LIB="e:\BUILD\lib\PCRE\lib\pcre.lib"

--------------------------------------------------------------------------------
//420行
# Uncomment the following to enable NetCDF format.
#NETCDF_PLUGIN = NO
#NETCDF_SETTING=yes
#NETCDF_LIB=C:\Software\netcdf\lib\netcdf.lib
#NETCDF_INC_DIR=C:\Software\netcdf\include
```

```
# Uncomment the following to add NC4 and HDF4 support
#NETCDF_HAS_NC4 = yes
#NETCDF_HAS_HDF4 = yes

# PROJ.4 stuff
# Uncomment the following lines to link PROJ.4 library statically. Otherwise
# it will be linked dynamically during runtime.
#PROJ_FLAGS = -DPROJ_STATIC
#PROJ_INCLUDE = -Id:\projects\proj.4\src
#PROJ_LIBRARY = d:\projects\proj.4\src\proj_i.lib
//改为
# Uncomment the following to enable NetCDF format.
NETCDF_PLUGIN = NO
NETCDF_SETTING=yes
NETCDF_LIB=E:\BUILD\netcdf\lib\netcdf.lib
NETCDF_INC_DIR=E:\BUILD\netcdf\include

# Uncomment the following to add NC4 and HDF4 support
NETCDF_HAS_NC4 = yes
#NETCDF_HAS_HDF4 = yes

# PROJ.4 stuff
# Uncomment the following lines to link PROJ.4 library statically. Otherwise
# it will be linked dynamically during runtime.
PROJ_FLAGS = -DPROJ_STATIC
PROJ_INCLUDE = -IC:\PROJ\include
PROJ_LIBRARY = c:\PROJ\lib\proj.lib

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

//479行
# Uncomment to use libcurl (DLL by default)
# The cURL library is used for WCS, WMS, GeoJSON, SRS call importFromUrl(),\
    WFS, GFT, CouchDB, /vsicurl/ etc.
#CURL_DIR=C:\curl-7.15.0
#CURL_INC = -I$(CURL_DIR)/include
# Uncoment following line to use libcurl as dynamic library
#CURL_LIB = $(CURL_DIR)/libcurl_imp.lib wsock32.lib wldap32.lib winmm.lib
# Uncoment following two lines to use libcurl as static library
#CURL_LIB = $(CURL_DIR)/libcurl.lib wsock32.lib wldap32.lib winmm.lib
#CURL_CFLAGS = -DCURL_STATICLIB
//改为
# Uncomment to use libcurl (DLL by default)
# The cURL library is used for WCS, WMS, GeoJSON, SRS call importFromUrl(),\
    WFS, GFT, CouchDB, /vsicurl/ etc.
CURL_DIR=E:\BUILD\lib\libcurl
CURL_INC = -I$(CURL_DIR)/include
# Uncoment following line to use libcurl as dynamic library
#CURL_LIB = $(CURL_DIR)/libcurl_imp.lib wsock32.lib wldap32.lib winmm.lib
# Uncoment following two lines to use libcurl as static library
CURL_LIB = $(CURL_DIR)/lib/libcurl_a.lib wsock32.lib wldap32.lib winmm.lib
CURL_CFLAGS = -DCURL_STATICLIB

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

//495行
# Uncomment for GEOS support (GEOS >= 3.1.0 required)
#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

//改为
# Uncomment for GEOS support (GEOS >= 3.1.0 required)
GEOS_DIR=e:\BUILD\geos-3.4.2
GEOS_CFLAGS = -I$(GEOS_DIR)/capi -I$(GEOS_DIR)/include -DHAVE_GEOS
GEOS_LIB      = $(GEOS_DIR)/src/geos.lib

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

//505行
# Uncomment for OpenJpeg (release v2.0.0) support
#OPENJPEG_ENABLED = YES
#OPENJPEG_CFLAGS = -IC:\openjpeg\include
#OPENJPEG_LIB = C:\openjpeg\lib\openjpeg.lib

//改为
# Uncomment for OpenJpeg (release v2.0.0) support
OPENJPEG_ENABLED = YES
OPENJPEG_CFLAGS = -If:\GDAL\openjpeg\include
OPENJPEG_LIB = f:\GDAL\openjpeg\lib\openjp2.lib

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

//530行
# Uncomment for WEBP support
#WEBP_ENABLED = YES
#WEBP_CFLAGS = -IE:/libwebp-0.1-windows/dev/Include
#WEBP_LIBS = e:/libwebp-0.1-windows/dev/lib/libwebp_a.lib

//改为
# Uncomment for WEBP support
WEBP_ENABLED = YES
WEBP_CFLAGS = -IF:\GDAL\libwebp-0.3.1\src
WEBP_LIBS = f:\GDAL\libwebp\lib\libwebp.lib

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

//591行
LINKER_FLAGS = $(EXTRA_LINKER_FLAGS) $(MSVC_VLD_LIB) $(LDEBUG)

//改为,防止openjepg库链接出错
LINKER_FLAGS = $(EXTRA_LINKER_FLAGS) $(MSVC_VLD_LIB) $(LDEBUG) /NODEFAULTLIB:LIBCMT
```

需要中文路径支持，请参看 关于GDAL180中文路径不能打开的问题分析与解决

我选择的是方案2，修改 GDAL_HOME\frmts\gdalallregister.cpp 文件73行左右，GDALAllRegister() 函数，以及 GDAL_HOME\ogr\ogrsf_frmts\generic\ogrregisterall.cpp 38行左右，OGRRegisterAll() 函数，在函数最前面添加

```
CPLSetConfigOption("GDAL_FILENAME_IS_UTF8","NO");
```

然后使用nmake即可，需要debug的话，加上参数 debug=1

```
nmake /f makefile.vc
nmake /f makefile.vc devinstall
```

如果需要64位，请修改153行左右， #WIN64=YES 为 WIN64=YES

GDAL 2.0``的版本中,某些gtiff的文件投影默认读不出来,需要添加``GDAL_DATA``环境变量,也可以删除frmtsgtiffgt_wkt_srs.cpp‘‘中 716-733行,‘‘GDAL dev‘‘中已经修复, Ticket 6210 :

```
if( psDefn->Model == ModelTypeProjected &&
    psDefn->PCS != KvUserDefined &&
    GDALGTIFKeyGetSHORT(hGTIF, ProjectionGeoKey, &tmp, 0, 1  ) == 0 &&
    GDALGTIFKeyGetSHORT(hGTIF, ProjCoordTransGeoKey, &tmp, 0, 1  ) == 0 &&
    GDALGTIFKeyGetSHORT(hGTIF, GeographicTypeGeoKey, &tmp, 0, 1  ) == 0 &&
    GDALGTIFKeyGetSHORT(hGTIF, GeogGeodeticDatumGeoKey, &tmp, 0, 1  ) == 0 &&
    GDALGTIFKeyGetSHORT(hGTIF, GeogEllipsoidGeoKey, &tmp, 0, 1  ) == 0 &&
    CSLTestBoolean(CPLGetConfigOption("GTIFF_IMPORT_FROM_EPSG", "YES")) )
{
    // Save error state as importFromEPSGA() will call CPLReset()
    int errNo = CPLGetLastErrorNo();
    CPLErr eErr = CPLGetLastErrorType();
    const char* pszTmp = CPLGetLastErrorMsg();
    char* pszLastErrorMsg = CPLStrdup(pszTmp ? pszTmp : "");
    CPLPushErrorHandler(CPLQuietErrorHandler);
    OGRErr eImportErr = oSRS.importFromEPSG(psDefn->PCS);
    CPLPopErrorHandler();
    // Restore error state
    CPLErrorSetState( eErr, errNo, pszLastErrorMsg);
    CPLFree(pszLastErrorMsg);
    bGotFromEPSG = (eImportErr == OGRERR_NONE);
}
```

9.15 编译结果下载

放在百度网盘里，有hdf4、hdf5、curl、expat、gdal等，会更新，需要者自取。

Note: 编译完成的库

CHAPTER 10

OGR矢量结构

本章主要介绍OGR中矢量类型的架构,与 *GDAL数据模型* 一节类似,主要讲解OGR中矢量的整体结构，和其中涉及到的类，OGR设计是参照OpenGIS中的简单要素模型，因此本章中会简单介绍一些OpenGIS中的内容。本章节主要是对应 GDAL1.11 以下版本， GDAL2.0 以上版本中，将 驱动 修改为 GDALDriver ，将 数据源 修改为 GDALDataset ，其他没有明显变化和修改。本章主要参考 OGR Architecture ， OpenGIS的一些基本概念 。

10.1 OpenGIS中矢量要素简介

OpenGIS定义了一组基于数据的服务，而数据的基础是要素 Feature 。所谓要素，简单地说就是一个独立的对象，在地图中可能表现为一个多边形建筑物，在数据库中即一个独立的条目。要素具有两个必要的组成部分，几何信息和属性信息。

OpenGIS将几何信息分为点、边缘、面和几何集合四种：其中我们熟悉的线 Linestring 属于边缘的一个子类，而多边形 Polygon 是面的一个子类。也就是说OpenGIS定义的几何类型并不仅仅是我们常见的点、线、多边形三种，它提供了更复杂更详细的定义，增强了未来的可扩展性。另外，几何类型的设计中采用了组合模式 Composite ，将几何集合 GeometryCollection 也定义为一种几何类型，类似地，要素集合 FeatureCollection 也是一种要素。属性信息没有做太大的限制，可以在实际应用中结合具体的实现进行设置。

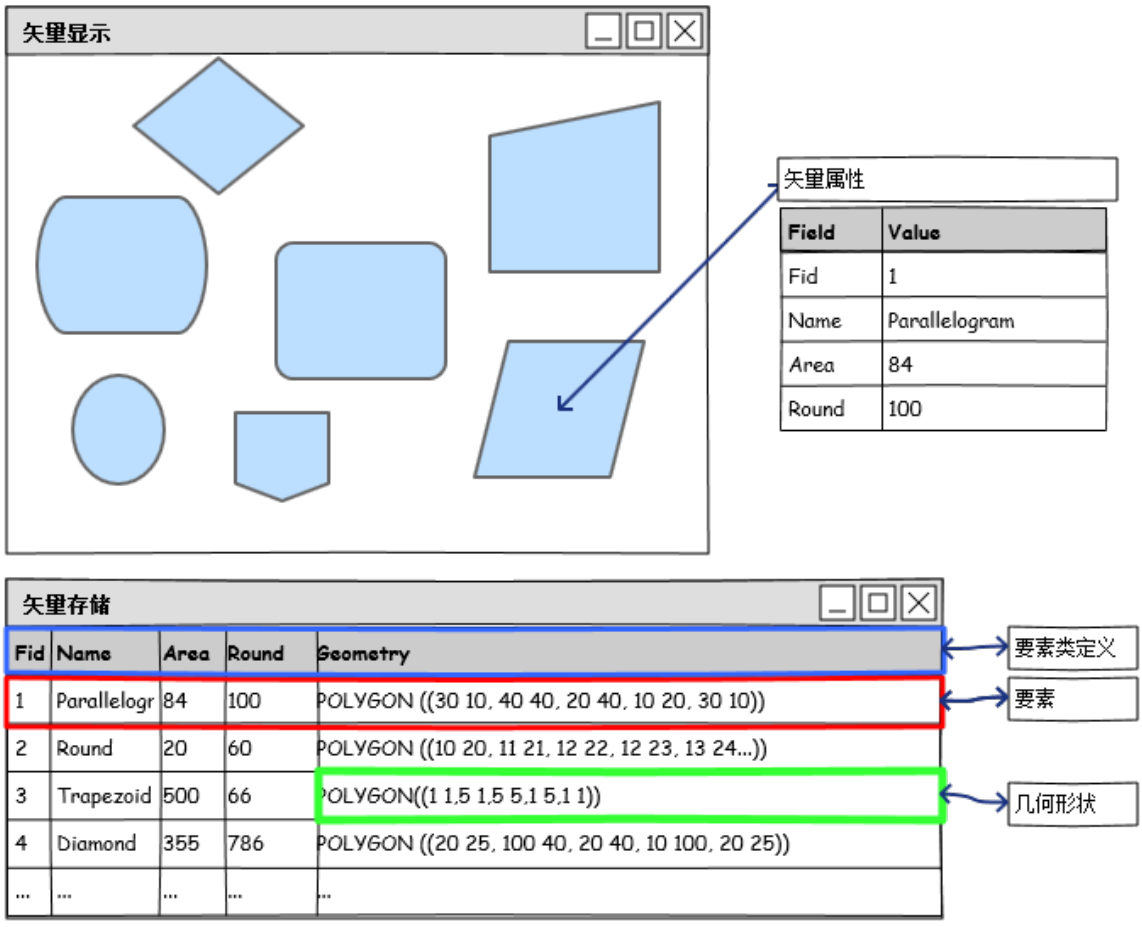
相同的几何类型、属性类型的组合成为要素类型 FeatureType ，要素类型相同的要素可以被存放在一个数据源中。而一个数据源只能拥有一个要素类型。因此，可以用要素类型来描述一组属性相似的要素。在面向对象的模型中，完全可以把要素类型理解为一个类，而要素则是类的实例。

实际理解中，可以将数据源理解为一个数据库，有多个图层，每个图层相当于一张数据表，每个数据表中有一个字段存储几何数据(OGR 1.11以后可以有多个几何数据字段)，其他字段中存储该数据的属性，如下图所示:

10.2 OGR类总览

总体结构如下所示:

- Geometry 几何形状： OGRGeometry 类以及其子类封装了OpenGIS模型中矢量数据，提供了一些几何操作，以及转换为WKB、WKT的操作，每个几何形状中包含有空间参考系统（投影）。
- Spatial Reference 空间参考：OGRSpatialReference 类封装了投影和水准面。
- Feature 要素： OGRFeature 类封装一个要素，包括其几何形状和其他属性信息


矢量显示
矢量属性
Field
Value
Fid
1
Name
Parallelogram
Area
84
Round
100
矢量存储
Fid
Name
Area
Round
Geometry
1
Parallelogr
84
100
POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
2
Round
20
60
POLYGON ((10 20, 11 21, 12 22, 12 23, 13 24...))
3
Trapezoid
500
66
POLYGON((1 1,5 1,5 5,1 5,1 1))
4
Diamond
355
786
POLYGON ((20 25, 100 40, 20 40, 10 100, 20 25))
...
...
...
...
...
要素类定义
要素
几何形状

- Feature Class Definition 要素类定义: OGRFeatureDefn 类通常存储整个图层的属性的元数据
- Layer 图层： OGRLayer 表示一个图层，同类型要素的集合
- Data Source 数据源： OGRDataSource 表示存储矢量数据的文件或数据库对象
- Drivers 驱动： OGRSFDriver 每个驱动可以可以打开对应类型的数据，转换为数据源，通过 OGRSFDriverRegistrar 管理。

10.3 几何形状

几何形状类用于表示各种几何矢量。它的派生类包括了各种形状，有如下类型： OGRPoint , OGRLineString , OGRPolygon , OGRGeometryCollection , OGRMultiPolygon , OGRMultiPoint , OGRMultiLineString ，其派生关系如下：

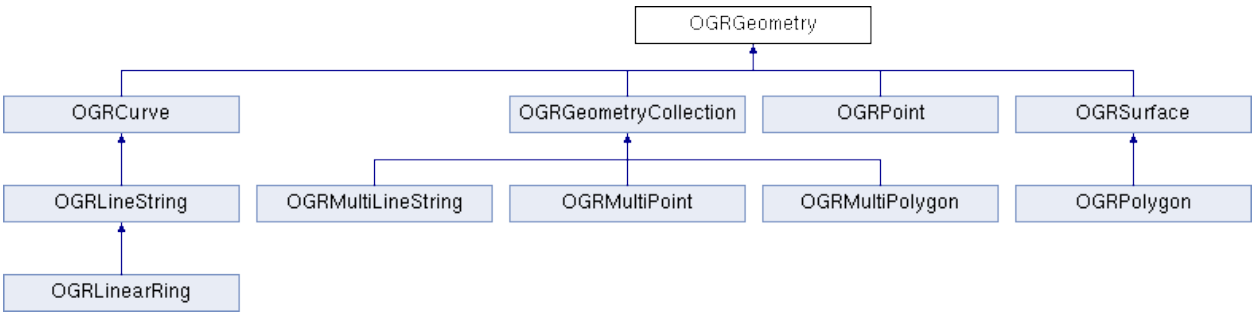


Fig. 10.1: **OGRGeometry 类**

OGR矢量结构 中提到，OGR设计参照的是OpenGIS中的简单要素模型，下面就是OpenGIS规范中的设计：

OGRCurve 、 OGRSurface 两个中间抽象基类抽象出一些公用的功能，当前在简单要素模型中的一些中间基类没有出现在OGR中，以后可能会改变。

OGRGeometryFactory 类用于将 WKT 或 WKB 串转化为几何形状。

OGRGeometry 包含 OGRSpatialReference 的引用，用于定义几何形状的空间参考。

很多分析功能暂时没有在 OGRGeometry 中实现。

虽然理论上可以从现有的 OGRGeometry 类中派生出更多的其他几何形状，但是目前没有成熟方案。而且，可以使用 OGRGeometryFactory 创建特殊几何形状而不用修改它。

10.4 空间参考

OGRSpatialReference 类用于实现OpenGIS空间参考系统定义。目前已经支持本地坐标系、地理坐标系、投影坐标系、垂直坐标系、地心坐标系、复合坐标系。

空间坐标系统是从OpenGIS定义的WKT投影格式中继承的。其中简单模式在“Simple Features specifications“ 中定义，复杂模式在 Coordinate Transformation specification 中定义，OGRSpatialReference 类实现是建立在 Coordinate Transformation specification 模式上，但是兼容早期的简单模式。

OGRCoordinateTransformation 类使用 PROJ.4 库实现坐标转换。请参照坐标转换相关内容。

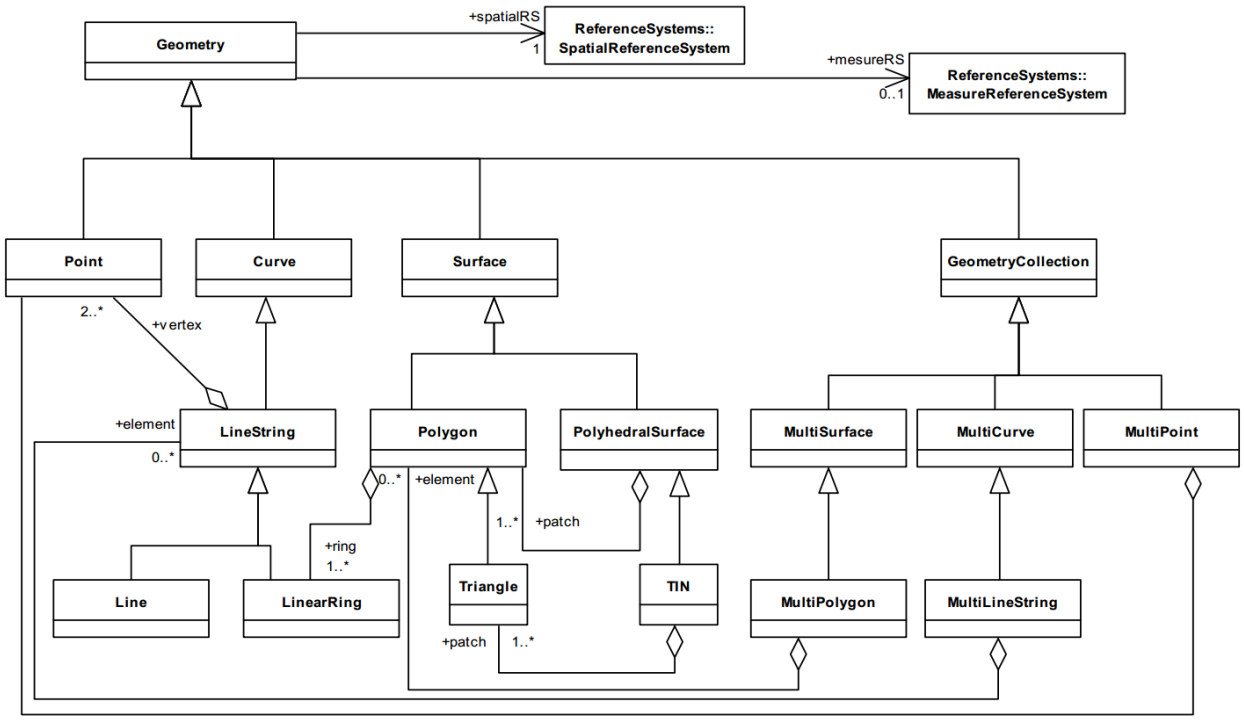


Fig. 10.2: **OpenGIS Geometry 描述**

10.5 要素/要素类定义

`OGRFeature` 保存几何形状和其属性、ID、要素类标识符。从 OGR1.11开始，要素中可以有多个几何形状字段。

`OGRFeatureDefn` 用于表示要素属性的类型、名称等元数据信息。一个 `OGRFeatureDefn` 通常用于表示一个图层的要素属性信息。具有相同要素类定义的要素通过引用计数的方式共享同一个要素类定义。

要素ID（FID）在图层中是唯一的，用于标识。单独的要素或者还未写入图层的要素可能有空FID，在OGR中FID用长整型表示，而其他模型中并不一定，例如GML中用字符串表示，而Oracle中行ID比长整型要大。

要素类中包含几何类型（使用 `OGRFeatureDefn::GetGeomType()` 获取 `OGRwkbGeometryType`），如果是wkbUnknown类型，那么该要素中可以添加任意类型的几何形状。这意味着一个图层中可以有不同的几何形状。

从 `OGR 1.11` 开始，要素允许有多个几何字段，每个几何字段可以有不同的类型值和不同的空间参考系统。

`OGRFeatureDefn` 包含要素类名称，通常作为图层名称。

10.6 图层

`OGRLayer` 表示数据源中的一层要素，同一图层的要素有相同的属性，`OGRLayer` 中也包含从数据源中读写要素的方法。图层主要用于在数据源中读写数据，通常代表一个文件，或者是空间表。

`OGRLayer` 有顺序和随机读写方法，`GetNextFeature` 方法可以顺序读取所有要素，可以使用 `SetSpatialFilter` 等滤波器限定该方法获取的要素范围。

当前 OGR 有一个缺陷：一个图层只能设置一个空间滤波器，以后将引入 OGRLayerView 或其他方式来改进。

OGRLayer 和 OGRFeatureDefn 虽然是一一对应的关系，但是设计为两个类的原因如下：

1. OGRFeatureDefn 和 OGRFeature 并不依赖于 OGRLayer ，因此两者可以独立存在于内存中。
2. SF CORBA模型与SFCOM 和 SFSQL 不同，并没有图层与固定属性列的概念，在OGR中实现SFCORBA需要考虑这些。

OGRLayer 是一个抽象类，每个驱动将单独实现， OGRLayers 通常都由 OGRDataSource 直接创建，而不能直接初始化或者删除。

10.7 数据源

OGRDataSource 表示 OGRLayer 对象集合。通常代表单个文件、一系列文件、数据库或者网关。OGRDataSource 通常拥有 OGRLayer 列表，可以返回它们的引用。

OGRDataSource 是抽象基类，将有每个格式的驱动实现， OGRDataSource 对象通常由对应的 OGRSFDriver 实例化，删除 OGRDataSource 对象仅是关闭其对数据源的访问，而非物理删除数据源。

OGRSFDriver 可以通过 OGRDataSource 的名称(通常为文件名)重新打开数据源。

OGRDataSource 可以通过 ExecuteSQL 方法执行对数据源的特殊命令，例如用SQL进行查询，虽然一些数据源底层支持空间SQL操作，OGR也实现了对任意数据源的查询操作（SQL的子集）。

10.8 驱动

OGRSFDriver 用于将对应格式的文件对象实例化， OGRSFDriverRegistrar 用于注册各种驱动。每个格式有对应的派生类来实例化 OGRDataSource 和 OGRLayer 。

在应用程序启动时需要调用所需文件格式的驱动，这些函数实例化对应的驱动对象，通过 OGRSFDriverRegistrar 注册。当数据源打开时，OGRSFDriverRegistrar 将遍历使用所有的驱动，直到成功返回 OGRDataSource 对象。

CHAPTER 11

OGR投影简介

本章来源： OGR Projections Tutorial

11.1 简介

OGRSpatialReference，和 OGRCoordinateTransformation 类提供的表达坐标系统（投影和基准面）和它们之间的变换的接口。这些服务是仿照OpenGIS规范的坐标变换，并使用WKT格式描述的坐标系统。

Simple Features for COM 中可以找到 OpenGIS 坐标系统和服务的介绍， Spatial Reference Systems Abstract Model 的介绍可以在 OGC 网站上找到。 GeoTIFF Projections Transform List 中也可以找到WKT的一些说明。 EPSG Geodesy 网站上的资源也非常有用.

11.2 定义地理坐标系

坐标系统被封装在 OGRSpatialReference 类中。有许多方法将 OGRSpatialReference 对象初始化为一个有效的坐标系统。有两种主要的坐标系统，地理坐标系（经纬度投影）和投影坐标系（如UTM，单位为米或英尺）。

地理坐标系统包含的基准面信息（半长轴，扁率），本初子午线（通常是格林尼治），和单位类型（通常是度）。下面的代码是地理坐标系统实例:

```
OGRSpatialReference oSRS;
oSRS.SetGeogCS( "My geographic coordinate system",
                "WGS_1984",
                "My WGS84 Spheroid",
                SRS_WGS84_SEMIMAJOR, SRS_WGS84_INVFLATTENING,
                "Greenwich", 0.0,
                "degree", SRS_UA_DEGREE_CONV );
```

上面的代码中，“My geographic coordinate system”、“My WGS84 Spheroid”、“Greenwich”和“degree”不是关键，主要用于向用户显示投影信息。然而，“WGS_1984” 作为确定基准面的关键，是需要遵循一定规则的。

OGRSpatialReference中已有很多的坐标系统的支持，其中包括“NAD27”、“NAD83”、“wgs72”和“WGS84”等，可以直接调用SetWellKnownGeogCS()获取

```
oSRS.SetWellKnownGeogCS( "WGS84" );
```

此外，可以使用EPSG数据库中的GCS码数表示投影

```
oSRS.SetWellKnownGeogCS( "EPSG:4326" );
```

OGRSpatialReference 可以调用函数将投影序列化,转为WKT串或者其他形式输出。

```
char     *pszWKT = NULL;
oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.exportToWkt( &pszWKT );
printf( "%s\n", pszWKT );
```

输出为:

```
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG",7030]],
        TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG",6326]],
    PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],
    UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG",9108]],
    AXIS["Lat",NORTH],
    AXIS["Long",EAST],
    AUTHORITY["EPSG",4326]]
```

OGRSpatialReference::importFromWkt() 方法可以用WKT坐标系统定义。

11.3 投影坐标系

投影坐标系统（如UTM、Lambert Conformal Conic等）默认包含了一个地理坐标系统，以及一个平面坐标与地理坐标相互转换的描述。下面的代码定义了一个UTM 17号带投影，默认为WGS84地理坐标系统。

```
OGRSpatialReference     oSRS;
oSRS.SetProjCS( "UTM 17 (WGS84) in northern hemisphere." );
oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.SetUTM( 17, TRUE );
```

调用 SetProjCS() 函数设置投影名称。SetWellKnownGeogCS() ``设置地理坐标系统，``SetUTM() 设置详细的投影变换参数。在现有版本中，上述步骤顺序不可颠倒，否则无法生成有效投影。

上述定义将WKT版本如下。请注意，UTM 17 在描述参数中被定义:

```
PROJCS["UTM 17 (WGS84) in northern hemisphere.",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG",7030]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG",6326]],
        PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],
        UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG",9108]],
        AXIS["Lat",NORTH],
        AXIS["Long",EAST],
        AUTHORITY["EPSG",4326]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-81],
    PARAMETER["scale_factor",0.9996],
```

```
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0]]
```

有许多投影方法包括settm()方法（横轴），setlcc()（Lambert Conformal Conic），和setmercator()。

11.4 查询坐标系统

一旦一个 OGRSpatialReference 已经建立，它的信息可以被查询。

- OGRSpatialReference::IsProjected() OGRSpatialReference::IsGeographic() 函数确定投影类型
- OGRSpatialReference::GetSemiMajor() ， OGRSpatialReference::GetSemiMinor() OGRSpatialReference::GetInvFlattening() 方法可用于获取有关球体的信息。
- OGRSpatialReference::GetAttrValue() 方法可以用来获得projcs，geogcs，基准，球体，和投影名称的字符串
- OGRSpatialReference::GetProjParm() 方法可用于获取投影参数。
- OGRSpatialReference::GetLinearUnits() 方法可用于获取投影的单位,可以转化到米。

下面的代码演示了如何获取信息:

```
const char *pszProjection = poSRS->GetAttrValue("PROJECTION");
if( pszProjection == NULL )
{
    if( poSRS->IsGeographic() )
        sprintf( szProj4+strlen(szProj4), "+proj=longlat " );
    else
        sprintf( szProj4+strlen(szProj4), "unknown " );
}
else if( EQUAL(pszProjection,SRS_PT_CYLINDRICAL_EQUAL_AREA) )
{
    sprintf( szProj4+strlen(szProj4),
       "+proj=cea +lon_0=%.9f +lat_ts=%.9f +x_0=%.3f +y_0=%.3f ",
             poSRS->GetProjParm(SRS_PP_CENTRAL_MERIDIAN,0.0),
             poSRS->GetProjParm(SRS_PP_STANDARD_PARALLEL_1,0.0),
             poSRS->GetProjParm(SRS_PP_FALSE_EASTING,0.0),
             poSRS->GetProjParm(SRS_PP_FALSE_NORTHING,0.0) );
}
...
```

11.5 坐标转换

OGRCoordinateTransformation 类用于不同坐标系之间的转换。使用 OGRCreateCoordinateTransformation() 创建转换对象，然后调用 OGRCoordinateTransformation::Transform() 方法转换坐标。

```
OGRSpatialReference oSourceSRS, oTargetSRS;
OGRCoordinateTransformation *poCT;
double x, y;

oSourceSRS.importFromEPSG( atoi( papszArgv[i + 1] ) );
oTargetSRS.importFromEPSG( atoi( papszArgv[i + 2] ) );
```

```
poCT = OGRCreateCoordinateTransformation( &oSourceSRS,
       &oTargetSRS );
x = atof( papszArgv[i + 3] );
y = atof( papszArgv[i + 4] );

if( poCT == NULL || !poCT->Transform( 1, &x, &y ) )
{
  printf( "Transformation failed.\n" );
} else {
  printf( "(%f,%f) -> (%f,%f)\n",
          atof( papszArgv[i + 3] ),
          atof( papszArgv[i + 4] ),
          x, y );
}
```

投影转换失败可能原因如下:

- `OGRCreateCoordinateTransformation()` 可能会失败，一般是因为系统之间可以建立没有变换。这可能是由于PROJ.4库不支持。如果 `OGRCreateCoordinateTransformation()` 失败，它将返回一个null。
- `OGRCoordinateTransformation::Transform()` 方法本身也会失败。

上述代码中没有演示三维点转换，但是该函数是可以用于三维点转换的，如果没有传入Z值，假设点在大地水准面上。

下面的示例演示如何方便地创建一个纬度/经度坐标系统使用相同的地理坐标系统和投影坐标系，并利用这些投影坐标和经纬度之间的转换。

```
OGRSpatialReference    oUTM, *poLatLong;
OGRCoordinateTransformation *poTransform;
oUTM.SetProjCS("UTM 17 / WGS84");
oUTM.SetWellKnownGeogCS( "WGS84" );
oUTM.SetUTM( 17 );
poLatLong = oUTM.CloneGeogCS();
poTransform = OGRCreateCoordinateTransformation( &oUTM, poLatLong );
if( poTransform == NULL )
{
    ...
}
...
if( !poTransform->Transform( nPoints, x, y, z ) )
```

11.6 其他语言接口

C接口定义的坐标系统服务 `ogr_srs_api.h` ，和Python绑定可以通过 `osr.py` 模块。其他语言用法类似c++，但是缺少一些方法。

```
typedef void *OGRSpatialReferenceH;
typedef void *OGRCoordinateTransformationH;
OGRSpatialReferenceH OSRNewSpatialReference( const char * );
void    OSRDestroySpatialReference( OGRSpatialReferenceH );
int     OSRReference( OGRSpatialReferenceH );
int     OSRDereference( OGRSpatialReferenceH );
OGRErr  OSRImportFromEPSG( OGRSpatialReferenceH, int );
OGRErr  OSRImportFromWkt( OGRSpatialReferenceH, char ** );
OGRErr  OSRExportToWkt( OGRSpatialReferenceH, char ** );
```

```
OGRErr  OSRSetAttrValue( OGRSpatialReferenceH hSRS, const char * pszNodePath,
                         const char * pszNewNodeValue );
const char *OSRGetAttrValue( OGRSpatialReferenceH hSRS,
                             const char * pszName, int iChild);
OGRErr  OSRSetLinearUnits( OGRSpatialReferenceH, const char *, double );
double  OSRGetLinearUnits( OGRSpatialReferenceH, char ** );
int     OSRIsGeographic( OGRSpatialReferenceH );
int     OSRIsProjected( OGRSpatialReferenceH );
int     OSRIsSameGeogCS( OGRSpatialReferenceH, OGRSpatialReferenceH );
int     OSRIsSame( OGRSpatialReferenceH, OGRSpatialReferenceH );
OGRErr  OSRSetProjCS( OGRSpatialReferenceH hSRS, const char * pszName );
OGRErr  OSRSetWellKnownGeogCS( OGRSpatialReferenceH hSRS,
                               const char * pszName );
OGRErr  OSRSetGeogCS( OGRSpatialReferenceH hSRS,
                      const char * pszGeogName,
                      const char * pszDatumName,
                      const char * pszEllipsoidName,
                      double dfSemiMajor, double dfInvFlattening,
                      const char * pszPMName ,
                      double dfPMOffset ,
                      const char * pszUnits,
                      double dfConvertToRadians );
double  OSRGetSemiMajor( OGRSpatialReferenceH, OGRErr * );
double  OSRGetSemiMinor( OGRSpatialReferenceH, OGRErr * );
double  OSRGetInvFlattening( OGRSpatialReferenceH, OGRErr * );
OGRErr  OSRSetAuthority( OGRSpatialReferenceH hSRS,
                         const char * pszTargetKey,
                         const char * pszAuthority,
                         int nCode );
OGRErr  OSRSetProjParm( OGRSpatialReferenceH, const char *, double );
double  OSRGetProjParm( OGRSpatialReferenceH hSRS,
                        const char * pszParmName,
                        double dfDefault,
                        OGRErr * );
OGRErr  OSRSetUTM( OGRSpatialReferenceH hSRS, int nZone, int bNorth );
int     OSRGetUTMZone( OGRSpatialReferenceH hSRS, int *pbNorth );
OGRCoordinateTransformationH
OCTNewCoordinateTransformation( OGRSpatialReferenceH hSourceSRS,
                                OGRSpatialReferenceH hTargetSRS );
void OCTDestroyCoordinateTransformation( OGRCoordinateTransformationH );
int OCTTransform( OGRCoordinateTransformationH hCT,
                  int nCount, double *x, double *y, double *z );
```

Python 接口:

```
class osr.SpatialReference
    def __init__(self,obj=None):
    def ImportFromWkt( self, wkt ):
    def ExportToWkt(self):
    def ImportFromEPSG(self,code):
    def IsGeographic(self):
    def IsProjected(self):
    def GetAttrValue(self, name, child = 0):
    def SetAttrValue(self, name, value):
    def SetWellKnownGeogCS(self, name):
    def SetProjCS(self, name = "unnamed" ):
    def IsSameGeogCS(self, other):
    def IsSame(self, other):
```

```
    def SetLinearUnits(self, units_name, to_meters ):
    def SetUTM(self, zone, is_north = 1):
class CoordinateTransformation:
    def __init__(self,source,target):
    def TransformPoint(self, x, y, z = 0):
    def TransformPoints(self, points):
```

11.7 内部实现

内部实现基于 PROJ.4 库,接口为 OGRCoordinateTransformation

CHAPTER 12

矢量数据读写

12.1 数据读取

与栅格的 数据读取 流程类似，但矢量数据定位明显，属性隐含，其几何形状和属性需要分开读取，GDAL有两个版本，读取接口不同，但是流程基本相同，以下是获取流程：

- 注册驱动
- 打开数据集
- 打开图层
- 获取要素类定义
- 遍历要素
- 获取几何形状以及属性信息

1.x 版本完整代码示例：

```
#include "ogrsf_frmts.h"
int main()
{
    //注册驱动
    OGRRegisterAll();
    //打开数据集
    OGRDataSource       *poDS;
    poDS = OGRSFDriverRegistrar::Open( "point.shp", FALSE );
    if( poDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }
    //打开图层
    OGRLayer  *poLayer;
    poLayer = poDS->GetLayerByName( "point" );
    OGRFeature *poFeature;
    //遍历要素
    poLayer->ResetReading();
    while( (poFeature = poLayer->GetNextFeature()) != NULL )
    {
        //获取要素类定义
        OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
        int iField;
        //获取属性信息
```

```
        for( iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
        {
            OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );
            if( poFieldDefn->GetType() == OFTInteger )
                printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
            else if( poFieldDefn->GetType() == OFTReal )
                printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
            else if( poFieldDefn->GetType() == OFTString )
                printf( "%s,", poFeature->GetFieldAsString(iField) );
            else
                printf( "%s,", poFeature->GetFieldAsString(iField) );
        }
        //获取几何形状
        OGRGeometry *poGeometry;
        poGeometry = poFeature->GetGeometryRef();
        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" );
        }
        OGRFeature::DestroyFeature( poFeature );
    }
    OGRDataSource::DestroyDataSource( poDS );
}
```

2.x 版本中最主要修改为 GDALDataset 代替‘‘OGRDataSource‘‘ 其余接口大致不变.打开方式略有区别

```
/**
* @param pszFilename 文件名
* @param nOpenFlags  GDAL_OF_RASTER/GDAL_OF_VECTOR  数据类型
                     GDAL_OF_READONLY/GDAL_OF_UPDATE 打开方式
                     GDAL_OF_SHARED 共享模式
                     GDAL_OF_VERBOSE_ERROR 返回详细错误
* @param papszAllowedDrivers NULL或以NULL结尾的列表,指定驱动
* @param papszOpenOptions NULL或以NULL结尾的字符串列表,根据驱动设置
* @param papszSiblingFiles NULL或以NULL结尾的文件列表
*/
GDALDatasetH GDALOpenEx     (       const char *    pszFilename,
unsigned int        nOpenFlags,
const char *const *         papszAllowedDrivers,
const char *const *         papszOpenOptions,
const char *const *         papszSiblingFiles
)
```

2.x 版本完整代码示例:

```
#include "ogrsf_frmts.h"
int main()
{
    //主要修改
    GDALAllRegister();
    GDALDataset       *poDS;
    poDS = (GDALDataset*) GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
    if( poDS == NULL )
```

```
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }
    //后面与1.x版本一致
    OGRLayer  *poLayer;
    poLayer = poDS->GetLayerByName( "point" );
    OGRFeature *poFeature;
    poLayer->ResetReading();
    while( (poFeature = poLayer->GetNextFeature()) != NULL )
    {
        OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
        int iField;
        for( iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
        {
            OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );
            if( poFieldDefn->GetType() == OFTInteger )
                printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
            else if( poFieldDefn->GetType() == OFTInteger64 )
                printf( CPL_FRMT_GIB ",", poFeature->GetFieldAsInteger64( iField ) );
            else if( poFieldDefn->GetType() == OFTReal )
                printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
            else if( poFieldDefn->GetType() == OFTString )
                printf( "%s,", poFeature->GetFieldAsString(iField) );
            else
                printf( "%s,", poFeature->GetFieldAsString(iField) );
        }
        OGRGeometry *poGeometry;
        poGeometry = poFeature->GetGeometryRef();
        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" );
        }
        OGRFeature::DestroyFeature( poFeature );
    }
    GDALClose( poDS );
}
```

Attention:
- 有很多矢量驱动是只读的，无法像栅格那样直接修改和创建文件。
- 如果需要确定，可以使用 `ogrinfo --formats` 确定驱动的权限。

12.2 数据写入

数据写入与读取类似，流程如下：

- 注册驱动
- 打开或创建数据集

- 创建图层
- 创建要素类定义
- 新建几何形状
- 新建要素
- 关闭数据集

1.x完整代码示例（C++）：

```
#include "ogrsf_frmts.h"
int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    OGRSFDriver *poDriver;
    OGRRegisterAll();
    poDriver = OGRSFDriverRegistrar::GetRegistrar()->GetDriverByName(
                pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
    OGRDataSource *poDS;
    poDS = poDriver->CreateDataSource( "point_out.shp", NULL );
    if( poDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }
    OGRLayer *poLayer;
    poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
    if( poLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }
    OGRFieldDefn oField( "Name", OFTString );
    oField.SetWidth(32);
    if( poLayer->CreateField( &oField ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }
    double x, y;
    char szName[33];
    while( !feof(stdin)
           && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {
        OGRFeature *poFeature;
        poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
        poFeature->SetField( "Name", szName );
        OGRPoint pt;

        pt.setX( x );
        pt.setY( y );

        poFeature->SetGeometry( &pt );
        if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
```

```
        {
            printf( "Failed to create feature in shapefile.\n" );
            exit( 1 );
        }
         OGRFeature::DestroyFeature( poFeature );
    }
    OGRDataSource::DestroyDataSource( poDS );
}
```

2.x 完整代码示例（C++）：

```
#include "ogrsf_frmts.h"
int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriver *poDriver;
    GDALAllRegister();
    poDriver = GetGDALDriverManager()->GetDriverByName(pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
    GDALDataset *poDS;
    poDS = poDriver->Create( "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
    if( poDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }
    OGRLayer *poLayer;
    poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
    if( poLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }
    OGRFieldDefn oField( "Name", OFTString );
    oField.SetWidth(32);
    if( poLayer->CreateField( &oField ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }
    double x, y;
    char szName[33];
    while( !feof(stdin)
           && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {
        OGRFeature *poFeature;
        poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
        poFeature->SetField( "Name", szName );
        OGRPoint pt;

        pt.setX( x );
        pt.setY( y );

        poFeature->SetGeometry( &pt );
        if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
```

```
        {
            printf( "Failed to create feature in shapefile.\n" );
            exit( 1 );
        }
        OGRFeature::DestroyFeature( poFeature );
    }
    GDALClose( poDS );
}
```

12.3 数据修改

GDAL可以添加、删除、修改属性信息和要素，下面简单介绍下如何添加/编辑属性字段(2.x):

```
GDALDataset* dst = (GDALDataset*)GDALOpenEx(shpName, GDAL_OF_UPDATE, NULL, NULL, NULL);
OGRLayer  *poLayer = dst->GetLayer(0);
OGRFieldDefn oField("imgNo", OFTString);  // Add fields
oField.SetWidth(128);
if (poLayer->CreateField(&oField, TRUE) != OGRERR_NONE)
{
printf("Creating Name field failed.\n");
}
OGRFeature *poFeature;

poLayer->ResetReading();
while ((poFeature = poLayer->GetNextFeature()) != NULL)
{
    poFeature->SetField("imgNo", "123456");  // Set fields
    poLayer->SetFeature(poFeature);
    OGRFeature::DestroyFeature(poFeature);
}
GDALClose(dst);
```

CHAPTER 13

OGR SQL

本章来源:

- GDAL官网: OGR SQL
- GDAL WIKI: RFC 28: OGR SQL Generalized Expressions

13.1 简介

`GDALDataset` 支持使用 `GDALDataset::ExecuteSQL()` 进行SQL操作,理论上该方法可以支持任何命令,但实际上主要是SQL SELECT 的子集.本篇主要讨论基本SQL和OGR实现的SQL,以及特定驱动的SQL支持.

自GDAL/OGR 1.10 起,可以使用SQLITE语法代替OGRSQL语法,SQLITE语法将在下文中详细介绍

`OGRLayer::SetAttributeFilter()` 方法中也支持OGRSQL过滤,用法与SELECT语句中WHERE从句用法一致

> **Attention:** OGR SQL在 版本1.8.0后有效,最好升级到新的版本中

13.2 SELECT

语法如下:

```
SELECT <field-list> FROM <table_def>
     [LEFT JOIN <table_def>
      ON [<table_ref>.]<key_field> = [<table_ref>.].<key_field>]*
     [WHERE <where-expr>]
     [ORDER BY <sort specification list>]

<field-list> ::= <column-spec> [ { , <column-spec> }... ]

<column-spec> ::= <field-spec> [ <as clause> ]
                | CAST ( <field-spec> AS <data type> ) [ <as clause> ]

<field-spec> ::= [DISTINCT] <field_ref>
                | <cumm-field-func> ( [DISTINCT] <field-ref> )
                | <field-expr>
                | Count(*)
```

```
<field-expr> ::= <field_ref>
                 | <constant-value>
                 | <field-expr> <field-operator> <field-expr>
                 | <field-func> ( <field-expr-list> )
                 | ( <field-expr> )

<field-expr-list> ::= field-expr
                  |  field-expr , field-expr-list
                  |  <empty>

<as clause> ::= [ AS ] <column_name>

<data type> ::= character [ ( field_length ) ]
                | float [ ( field_length ) ]
                | numeric [ ( field_length [, field_precision ] ) ]
                | integer [ ( field_length ) ]
                | date [ ( field_length ) ]
                | time [ ( field_length ) ]
                | timestamp [ ( field_length ) ]

<cumm-field-func> ::= AVG | MAX | MIN | SUM | COUNT

<field-operator> ::= '+' | '-' | '/' | '*' | '||'

<field-func> ::= CONCAT | SUBSTR

<field_ref>  ::= [<table_ref>.]field_name
```

13.3 SPECIAL FIELDS

OGR SQL中有一些特殊属性可以使用.比如几何属性等.

13.3.1 FID

一般FID不作为要素的属性,只用于标识,但某些情况下能方便的查询,*不包括FID,所以必须单独写出来

```
SELECT FID, * FROM nation;
```

13.3.2 OGR_GEOMETRY

MapInfo等数据类型中可以在同一层使用多种几何类型,所以可以使用此字段进行选择:

```
SELECT * FROM nation WHERE OGR_GEOMETRY='POINT' OR OGR_GEOMETRY='POLYGON'
```

13.3.3 OGR_GEOM_WKT

OGR_GEOM_WKT 可以显示要素的WKT串作为输出,也可以在WHERE从句中指定WKT类型等进行过滤:

```
SELECT OGR_GEOM_WKT, * FROM nation;
SELECT OGR_GEOM_WKT, * FROM nation WHERE OGR_GEOM_WKT \
        LIKE 'POINT%' OR OGR_GEOM_WKT LIKE 'POLYGON%';
```

13.3.4 OGR_GEOM_AREA

该字段使用OGRSurface::get_Area() 方法获取几何要素的面积,如果不是面状要素,则返回0

```
SELECT * FROM nation WHERE OGR_GEOM_AREA > 10000000
```

13.3.5 OGR_STYLE

该字段代表使用OGRFeature::GetStyleString()获取的字符串,一般用于where从句过滤

```
SELECT * FROM nation WHERE OGR_STYLE LIKE 'LABEL%'
```

13.4 ALTER TABLE

用于增删改字段,主要用法如下:

```
/*添加字段*/
ALTER TABLE tablename ADD [COLUMN] columnname columntype
/*重命名字段*/
ALTER TABLE tablename RENAME [COLUMN] oldcolumnname TO newcolumnname
/*修改字段类型*/
ALTER TABLE tablename ALTER [COLUMN] columnname TYPE columntype
/*删除字段*/
ALTER TABLE tablename DROP [COLUMN] columnname
```

支持的数据类型如下:

```
boolean     /*GDAL>2.0*/
character   /*需要指定field_length,默认为1*/
float       /*field_length*/
numeric     /*field_length, field_precision*/
smallint    /*field_length 16bit signed integer GDAL>2.0*/
integer     /*field_length*/
bigint      /*field_length 64bit integer GDAL>2.0*/
date        /*field_length*/
time        /*field_length*/
timestamp   /*field_length*/
geometry,geometry/*geometry_type*/,geometry/*geometry_type,type_code*/
```

例如,可以使用ogrinfo 命令,修改文件字段

```
ogrinfo -sql "alter table out add COLUMN myfield integer" out.shp
```

13.5 ExecuteSQL()

只有GDALDataset才能执行ExecuteSQL函数,该函数不针对Layer处理

```
/**
* @param pszSQLCommand     SQL命令
* @param poSpatialFilter   空间范围设定
* @param pszDialect        针对不同驱动,有不同语法,一般置为NULL
*/
OGRLayer * GDALDataset::ExecuteSQL( const char *pszSQLCommand,
```

```
                                        OGRGeometry *poSpatialFilter,
                                        const char *pszDialect );
```

13.6 Non-OGR SQL

MySQL, PostgreSQL PostGIS (PG), Oracle (OCI), SQLite, ODBC, ESRI Personal Geodatabase (PGeo) MS SQL Spatial (MSSQLSpatial) 这些数据库系统都自己重写了 GDALDataset::ExecuteSQL() 方法,有些语法上有细微的差别,另外其他语句也可以在这些驱动中运行.但是只有 SQL WHERE 语句才能返回 Layer.

CHAPTER 14

SQLite SQL

本章来源:

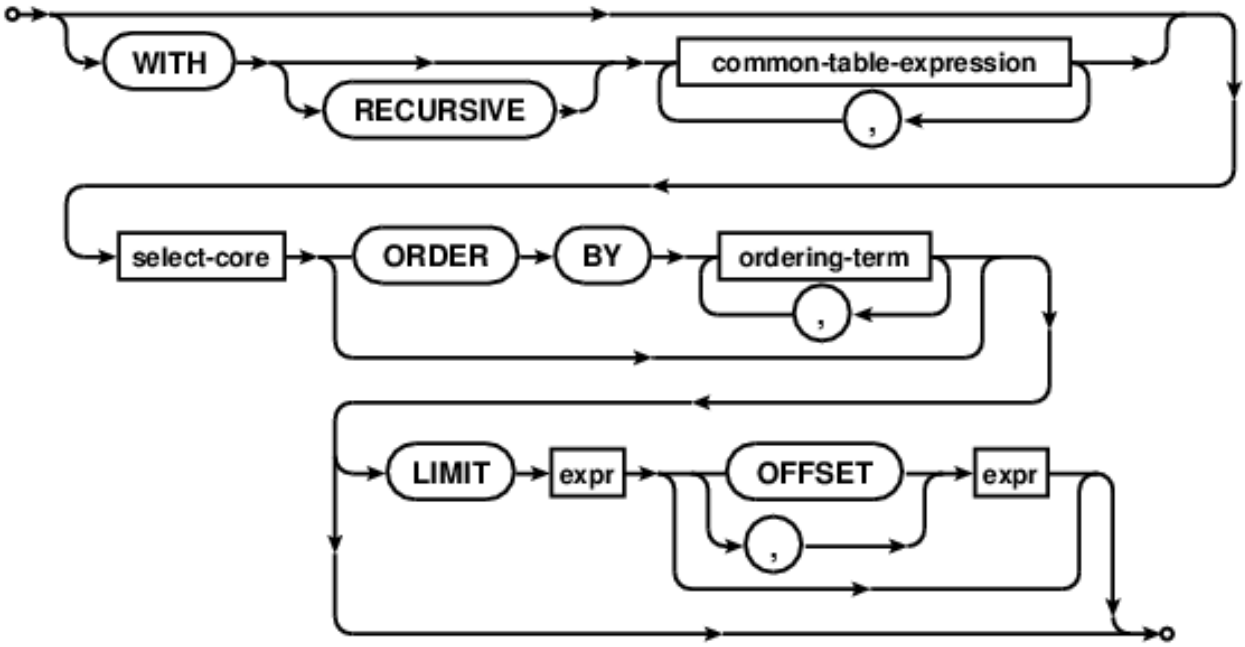
- SQLite: SQLite SQL dialect
- SpatiaLite: SpatiaLite 4.3.0 SQL functions reference list

14.1 简介

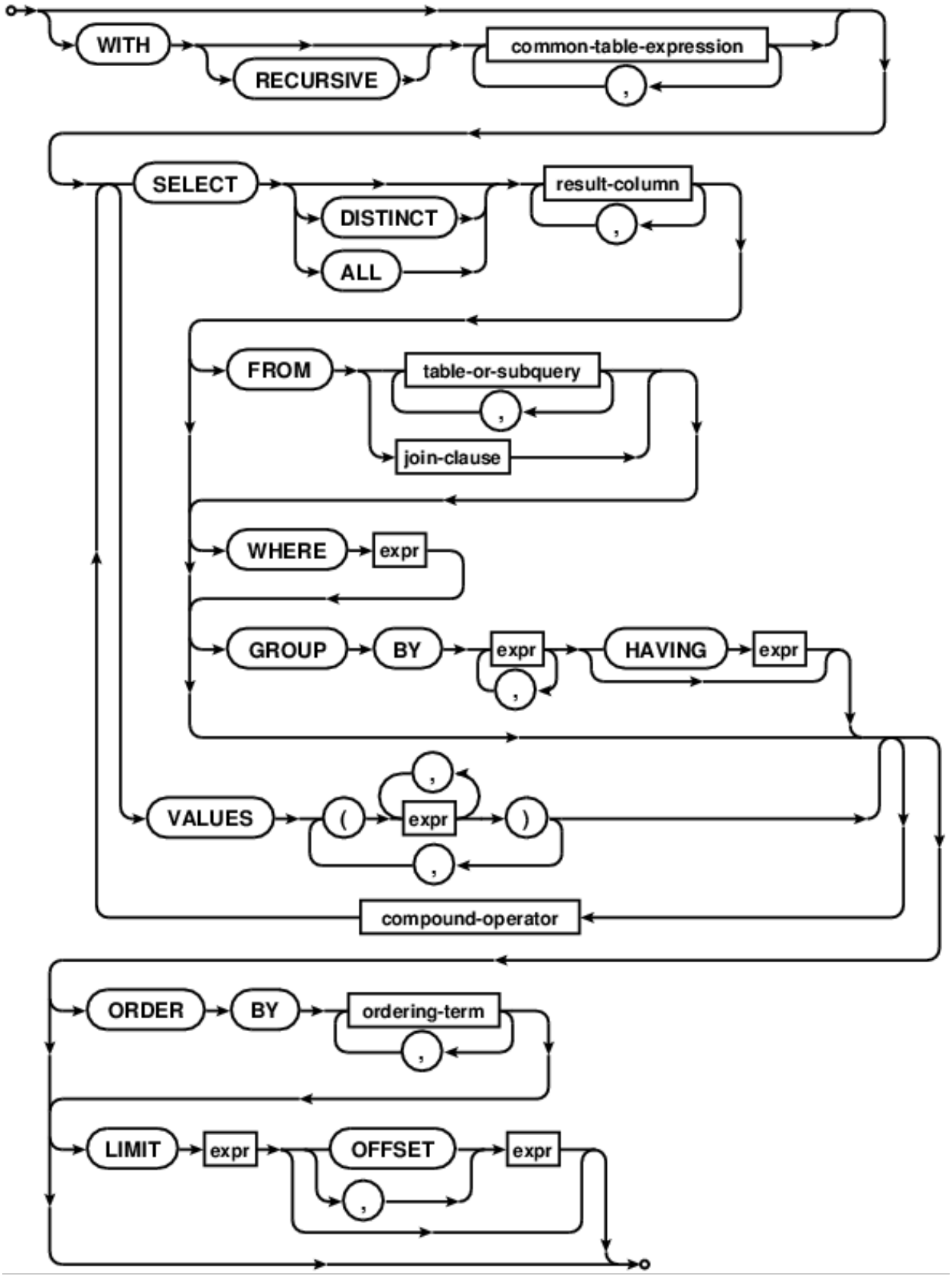
GDAL 1.10 版本后,可以使用 Sqlite 语法代替 ogrsql 语法.大部分语法支持都需要Spatialite库. 执行方式同样调用 GDALDataset::ExecuteSQL() ,但是需要指定 pszDialect 参数为 “SQLITE”.

14.2 SELECT statement

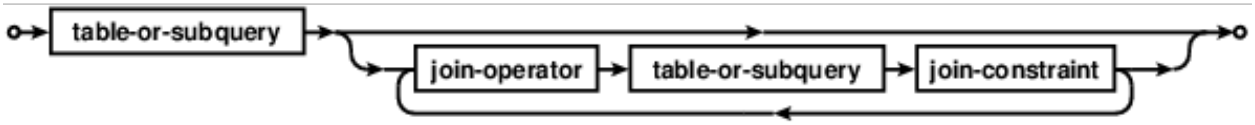
简单 select 语句如下:



select 语法图如下:


WITH
RECURSIVE
common-table-expression
,
SELECT
DISTINCT
ALL
result-column
,
FROM
table-or-subquery
,
join-clause
WHERE
expr
GROUP
BY
expr
,
HAVING
expr
VALUES
(
expr
)
,
compound-operator
ORDER
BY
ordering-term
,
LIMIT
expr
OFFSET
expr
,

join 语法如下:



14.3 Spatialite SQL

可以参考网站 SpatiaLite 4.3.0 SQL functions reference list 上的说明,下面列出来几个常用的,需要GEOS库支持

SQL functions that implement spatial operators

Function	Syntax	Summary
Intersection	Intersection(geom1 Geometry , geom2 Geometry) :Geometry	两物求交
	ST_Intersection(geom1 Geometry , geom2 Geometry) :Geometry	
Difference	Difference(geom1 Geometry , geom2 Geometry) :Geometry	两物求差
	ST_Difference(geom1 Geometry , geom2 Geometry) :Geometry	
GUnion	GUnion(geom1 Geometry , geom2 Geometry) : Geometry	两物求并
	ST_Union(geom1 Geometry , geom2 Geometry) : Geometry	
GUnion	GUnion(geom1 Geometry) : Geometry	整体求并
	ST_Union(geom1 Geometry) : Geometry	
Collect	Collect(geom1 Geometry , geom2 Geometry) : Geometry	合并两者
	ST_Collect(geom1 Geometry , geom2 Geometry):Geometry	
Collect	Collect(geom Geometry) : Geometry	合并集合
	ST_Collect(geom Geometry) : Geometry	

Function	Syntax	Summary
SymDifference	SymDifference(geom1 Geometry , geom2 Geometry):Geometry	两物逻辑或
	ST_SymDifference(geom1 Geometry ,geom2 Geometry):Geometry	
Buffer	Buffer(geom Geometry , dist Double precision[, quadrantsegments int]):Geometry	缓冲区
	ST_Buffer(geom Geometry , dist Double precision[, quadrantsegments int]):Geometry	
HausdorffDistance	HausdorffDistance(geom1 Geometry , geom2 Geometry) : Double precision	豪斯多夫距离
	ST_HausdorffDistance(geom1 Geometry, geom2 Geometry) :Double precision	
OffestCurve	OffsetCurve(geom Curve , radius Double precision) :Curve	偏移曲线正左负右
	ST_OffsetCurve(geom Curve , radius Double precision): Curve	
Snap	Snap(geom1 Geometry , geom2 Geometry , tolerance Double precision) : Geometry	咬合1到2
	ST_Snap(geom1 Geometry , geom2 Geometry , tolerance Double precision) : Geometry	

Function	Syntax	Summary
DissolveSegments	DissolveSegments(geom Geometry) : Geometry ST_DissolveSegments(geom Geometry) : Geometry	溶合线段点不会动环被拆分
DelaunayTriangulation	DelaunayTriangulation(geom Geometry [, edges_only Boolean [, tolerance Double precision]]) : Geometry ST_DelaunayTriangulation(geom Geometry [, edges_only Boolean [, tolerance Double precision]]): Geometry	德拉尼三角网
VoronojDiagram	VoronojDiagram(geom Geometry [, edges_only Boolean [, frame_extra_size Double precision [, tolerance Double precision]]]) : Geometry ST_VoronojDiagram(geom Geometry[, edges_only Boolean [, frame_extra_size Double precision [, tolerance Double precision]]]) : Geometry	Voronoj图
ConvexHull	ConvexHull(geom Geometry) : Geometry ST_ConvexHull(geom Geometry) : Geometry	凸包
ConcaveHull	ConcaveHull(geom Geometry [, factor Double precision [, allow_holes Boolean [, tolerance Double precision]]]) : Geometry ST_ConcaveHull(geom Geometry [, factor Double precision [, allow_holes Boolean [, tolerance Double precision]]]) : Geometry	凹包

CHAPTER 15

OGR工具集

主要来自 OGR Utility Programs

15.1 通用选项

所有ogr工具通用选项

--version
返回GDAL版本.

--formats
返回所有支持的格式.

--format format
某种格式的详细信息.

--optfile file
将一个文件中的内容作为工具参数列表.

--config key value
设置系统参数.

--debug value
设置debug级别.

--help-general
显示通用选项.

15.2 ogrinfo

ogrinfo 主要用来显示矢量数据的所有信息,也可以用sql操作矢量, ogrinfo 命令打开文件默认为读写方式打开,部分只读驱动会发出警告,用法如下:

```
ogrinfo [--help-general] [-ro] [-q] [-where restricted_where|\@filename]
        [-spat xmin ymin xmax ymax] [-geomfield field] [-fid fid]
        [-sql statement|\@filename] [-dialect dialect] [-al] [-so] [-fields={YES/NO}]
        [-geom={YES/NO/SUMMARY/WKT/ISO_WKT}] [-formats] [[-oo NAME=VALUE] ...]
        [-nomd] [-listmdd] [-mdd domain|`all`]*
        [-nocount] [-noextent]
        datasource_name [layer [layer ...]]
```

参数说明:

-ro:
以只读方式打开数据集.

-al:
列出所有图层的所有要素 (used instead of having to give layer names as arguments).

-so:
仅列出摘要:只列出部分要素,只显示投影/要素数量和边界范围等基本信息

-q:
隐藏提示信息.

-where `restricted_where`**:**
用 sql where 语句限定范围,仅输出查询部分的要素信息,GDAL2.1后可以使用 `\filename` 语法

-sql `statement`**:**
执行SQL语句.GDAL2.1以后,可以使用 `@filename` 语法指定针对文件查询

-dialect `dialect`**:**
特定SQL语法类型 OGRSQL 或 SQLITE .

-spat `xmin ymin xmax ymax`**:**
感兴趣区域范围,仅显示指定范围内容

-geomfield `field`**:**
指定 `geometry` 字段名字,用于空间条件过滤查询.

-fid `fid`**:**
指定查询的 `feature id`

-fields=`{YES/NO}`**:**
如果设置为NO,则不显示feature的字段值.

-geom=`{YES/NO/SUMMARY/WKT/ISO_WKT}`**:**
如何显示geometry,默认为YES,显示为WKT字符串

-oo `NAME=VALUE`**:**
数据集的打开选项(GDAL 2.0)

-nomd
(GDAL 2.0) 不显示元数据

-listmdd
(GDAL 2.0)显示所有元数据.

-mdd `domain`
(GDAL 2.0) 显示指定元数据信息

-nocount
(GDAL 2.0) 不显示要素数量.

-noextent
(GDAL 2.0) 不显示图层范围.

datasource_name:
数据集名称

layer:
一个或多个图层名

15.3 ogr2ogr

ogr2ogr 主要用于矢量格式转换,属性设置,重投影等,用法如下:

```
Usage: ogr2ogr [--help-general] [-skipfailures] [-append] [-update]
               [-select field_list] [-where restricted_where|\@filename]
               [-progress] [-sql <sql statement>|\@filename] [-dialect dialect]
               [-preserve_fid] [-fid FID]
               [-spat xmin ymin xmax ymax] [-spat_srs srs_def] [-geomfield field]
               [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def]
               [-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
               dst_datasource_name src_datasource_name
               [-lco NAME=VALUE] [-nln name]
               [-nlt type|PROMOTE_TO_MULTI|CONVERT_TO_LINEAR|CONVERT_TO_CURVE]
               [-dim XY|XYZ|XYM|XYZM|2|3|layer_dim] [layer [layer ...]]

Advanced options :
               [-gt n]
               [[-oo NAME=VALUE] ...] [[-doo NAME=VALUE] ...]
               [-clipsrc [xmin ymin xmax ymax]|WKT|datasource|spat_extent]
               [-clipsrcsql sql_statement] [-clipsrclayer layer]
               [-clipsrcwhere expression]
               [-clipdst [xmin ymin xmax ymax]|WKT|datasource]
               [-clipdstsql sql_statement] [-clipdstlayer layer]
               [-clipdstwhere expression]
               [-wrapdateline] [-datelineoffset val]
               [[-simplify tolerance] | [-segmentize max_dist]]
               [-addfields] [-unsetFid]
               [-relaxedFieldNameMatch] [-forceNullable] [-unsetDefault]
               [-fieldTypeToString All|(type1[,type2]*)] [-unsetFieldWidth]
               [-mapFieldType type1|All=type2[,type3=type4]*]
               [-fieldmap identity | index1[,index2]*]
               [-splitlistfields] [-maxsubfields val]
               [-explodecollections] [-zfield field_name]
               [-gcp pixel line easting northing [elevation]]* [-order n | -tps]
               [-nomd] [-mo "META-TAG=VALUE"]* [-noNativeData]
```

-f format_name:
　　默认为 -f "ESRI Shapefile",设置输出格式

-append:
　　添加到已存在的图层中,而非新建图层

-update:
　　更新已存在的数据集

-select field_list:
　　将字段复制到新图层中,使用逗号隔开字段名,默认全部复制, GDAL1.11开始可以复制geometry字段

-progress:
　　显示进度条,仅当输入图层有"fast feature count"属性

-sql sql_statement:
　　执行SQL语句,结果将输出到文件中.GDAL2.1以后,可以使用 @filename 语法指定针对文件查询.

-dialect dialect:
　　特定SQL语法类型 OGRSQL 或 SQLITE .

-where restricted_where:
　　属性查询 (like SQL WHERE). GDAL2.1以后,可以使用 @filename 语法指定针对文件查询.

-skipfailures:
跳过错误继续执行

-spat xmin ymin xmax ymax:
空间范围指定,空间范围外的要素将被剔除,范围内几何要素默认不被裁剪,除非设置 -clipsrc

-spat_srs srs_def:
(OGR >= 2.0) 覆盖空间过滤查询的 SRS.

-geomfield field:
(OGR >= 1.11) 空间过滤条件的geometry 字段名.

-dsco NAME=VALUE:
数据集创建选项 (format specific)

-lco NAME=VALUE:
图层创建选项(format specific)

-nln name:
指定创建图层的名称

-nlt type:
指定创建图层的几何类型 NONE, GEOMETRY, POINT, LINESTRING, POLYGON, GEOMETRYCOLLECTION, MULTIPOINT, MULTIPOLYGON or MULTILINESTRING. And CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE and MULTISURFACE (GDAL 2.0). 在类型后添加 "Z", "M", 或 "ZM" 可以设置高程/测量信息.GDAL 1.10后, PROMOTE_TO_MULTI 可以自动将polygon multipolygons混合图层转为multipolygons图层,linestrings 和 multilinestrings混合图层转为multilinestrings图层.GDAL 2.0 后, 可以使用CONVERT_TO_LINEAR将其他图层转为线层, 使用CONVERT_TO_CURVE 将非线性图层转为广义曲线类型(POLYGON to CURVEPOLYGON, MULTIPOLYGON to MULTISURFACE, LINESTRING to COMPOUNDCURVE, MULTILINESTRING to MULTICURVE). 2.1 后类型可以添加25D,与Z相同

-dim val:
(>GDAL 1.10) 强制坐标维数 (XY, XYZ, XYM, XYZM -).

-a_srs srs_def:
设置输出 SRS

-t_srs srs_def:
重投影/变换到此 SRS

-s_srs srs_def:
重指定输入 SRS

-preserve_fid:
指定与源数据相同的FID,而非默认生成FID. GDAL 2.0 以后此为默认选项,可以使用 -unsetFid 禁止此行为.

-fid fid:
处理指定fid要素,与 '-where "fid in (1,3,5)"' 效果类似.

高级选项 :

-oo NAME=VALUE:
(>= GDAL 2.0) 输入数据集打开选项(format specific)

-doo NAME=VALUE:
(>=GDAL 2.0) 输出数据集打开选项, (format specific), 仅在 -update 模式中使用

-gt n:
每次提交事务中多少要素进行转换(OGR 1.11 默认20000 , 之前版本为200),越大越快. GDAL 2.0之后可以使用unlimited一次事务中全部处理.

-ds_transaction:
(>=GDAL 2.0) 强制使用事务

-clipsrc [xmin ymin xmax ymax]|WKT|datasource|spat_extent:
(>=GDAL 1.7.0) 使用包络框裁切几何要素,如果指定datasource,还需使用 -clipsrcsql -clipsrclayer -clipsrcwhere 指定裁切要素

-clipsrcsql sql_statement:
使用sql查询选择所需裁切几何要素

-clipsrclayer layername:
选择裁切图层

-clipsrcwhere expression:
使用属性查询重新限定裁切要素

-clipdst [xmin ymin xmax ymax]|WKT|datasource|spat_extent:
(>=GDAL 1.7.0) 重投影后将要素裁切到指定范围内,如果指定数据源,还需使用-clipdstlayer, -clipdstwhere -clipdstsql指定带裁切的数据

-clipdstsql sql_statement:
使用sql查询选择所需待裁切几何要素

-clipdstlayer layername:
选择待裁切图层

-clipdstwhere expression:
使用属性查询重新限定待裁切要素

-wrapdateline:
(starting with GDAL 1.7.0) 分割穿过经线的几何要素 (long. = +/- 180deg)

-datelineoffset:
(starting with GDAL 1.10) 偏移分割经线 (default long. = +/- 10deg, geometries within 170deg to -170deg will be split)

-simplify tolerance:
(starting with GDAL 1.9.0) 化简图形,化简是保持每个图形的拓扑,不保证整个层的拓扑.

-segmentize max_dist:
(starting with GDAL 1.6.0) 两节点间最大距离.用于创建分割节点

-fieldTypeToString type1,...:
(starting with GDAL 1.7.0)将指定字段类型转为String类型到输出图层中,可以指定类型为: Integer, Integer64, Real, String, Date, Time, DateTime, Binary, IntegerList, Integer64List, RealList, StringList.

-mapFieldType srctype|All=dsttype,...:
(starting with GDAL 2.0) 将指定字段类型转换为其他类型,可指定类型为: Integer, Integer64, Real, String, Date, Time, DateTime, Binary, IntegerList, Integer64List, RealList, StringList.可以使用括号指定同时被转换的子类型, 例如:Integer(Boolean), Real(Float32)

-unsetFieldWidth:
(starting with GDAL 1.11)重置字段宽度和精度为0.

-splitlistfields:
(starting with GDAL 1.8.0) 将列表字段分割为多个字段,例如 StringList, RealList ,IntegerList转为多个String, Real or Integer.

-maxsubfields val:
与 -splitlistfields 结合使用,限制分割个数

-explodecollections:
(starting with GDAL 1.8.0) 为每个geometry生成一个feature

-zfield field_name:
(starting with GDAL 1.8.0)用指定字段填充Z值

-gcp ungeoref_x ungeoref_y georef_x georef_y elevation:
(starting with GDAL 1.10.0)指定地面控制点 ,可添加多个.

-order n:
(starting with GDAL 1.10.0)指定转换时多项式次数,默认根据控制点选择多项式阶数.

-tps:
(starting with GDAL 1.10.0) 强制使用tps转换

-fieldmap:
(starting with GDAL 1.10.0)与 -append一起使用,指定需要从源图层复制的字段索引列表,索引从0开始计算

-addfields:
(starting with GDAL 1.11)这是特定的-append版本.只能添加字段到指定输出图层中,一般用于合并字段不完全相同的文件

-relaxedFieldNameMatch:
(starting with GDAL 1.11) 放宽字段名称匹配条件 [-relaxedFieldNameMatch] [-forceNullable]

-forceNullable:
(starting with GDAL 2.0)去掉源图层禁止非空值的约束

-unsetDefault:
(starting with GDAL 2.0)去掉源图层的默认值

-unsetFid:
(starting with GDAL 2.0) 重置所有fid

-nomd:
(starting with GDAL 2.0) 禁止拷贝源图层的元数据

-mo "META-TAG=VALUE":
(starting with GDAL 2.0)指定元数据

-noNativeData:
(starting with GDAL 2.1) To disable copying of native data, i.e. details of source format not captured by OGR abstraction, that are otherwise preserved by some drivers (like GeoJSON) when converting to same format.

CHAPTER 16

Indices and tables

- genindex

Symbols

D

E

G

H

I

L

M

N

O

P

S

X

Y