在接触一个新的东西的时候，我喜欢从这新东西的一些名词和结构入手。记得以前一位老师说过，一个新的东西，肯定会有自己的专业术语，然后就是体系结构等……，我们就从这个开始吧

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

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

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

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

1．GDAL

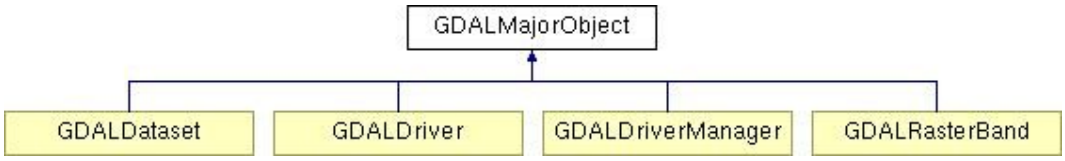
GDAL（Geospatial Data Abstraction Library）提供对多种栅格数据的支持，包括 Arc/Info ASCII Grid(asc)，GeoTiff (tiff)，Erdas Imagine Images(img)，ASCII DEM(dem) 等格式。

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

详细的结构描述请访问

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

GDAL 的核心类结构设计如图所示：



其中的类说明如下：

GDALMajorObject 类：带有元数据的对象。

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

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

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

2. OGR

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

Format Name	Code	Creation	Georeferencing
Arc/Info Binary Coverage	AVCBin	No	Yes
Arc/Info .E00 (ASCII) Coverage	AVCE00	No	Yes
Atlas BNA	BNA	Yes	No
AutoCAD DXF	DXF	Yes	No
Comma Separated Value (.csv)	CSV	Yes	No
DODS/OPeNDAP	DODS	No	Yes
ESRI Personal GeoDatabase	PGeo	No	Yes
ESRI ArcSDE	SDE	No	Yes
ESRI Shapefile	ESRI Shapefile	Yes	Yes
FMEObjects Gateway	FMEObjects Gateway	No	Yes
GeoJSON	GeoJSON	Yes	Yes
Géoconcept Export	Geoconcept	Yes	Yes
GeoRSS	GeoRSS	Yes	Yes
GML	GML	Yes	Yes
GMT	GMT	Yes	Yes
GPSBabel	GPSBabel	Yes	Yes
GPX	GPX	Yes	Yes
GRASS	GRASS	No	Yes
GPSTrackMaker (.gtm, .gtz)	GPSTrackMaker	Yes	Yes
Hydrographic Transfer Format	HTF	No	Yes
Informix DataBlade	IDB	Yes	Yes
INTERLIS	"Interlis 1" and "Interlis 2"	Yes	Yes

更详细的请访问 http://gdal.org/ogr/ogr_formats.html

1)OGR 体系结构

OGR 包括如下几部分：

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

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

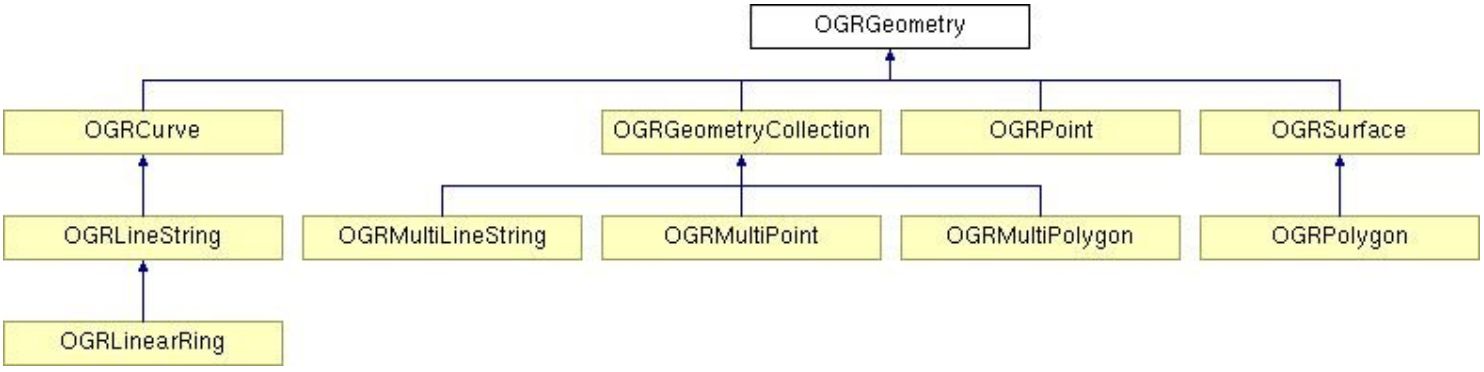
Feature：类 OGRFeature 封装了一个完整 Feature 的定义，一个完整的 Feature 包括一个 Geometry 和 Geometry 的一系列属性。

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

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

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

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



2）OGR 的 Geometry 模型

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

图-OGR 的 Geometry 模型关系图

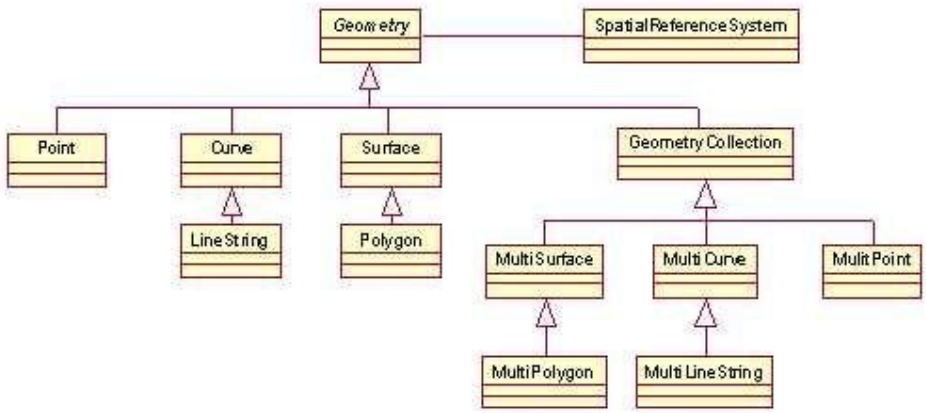
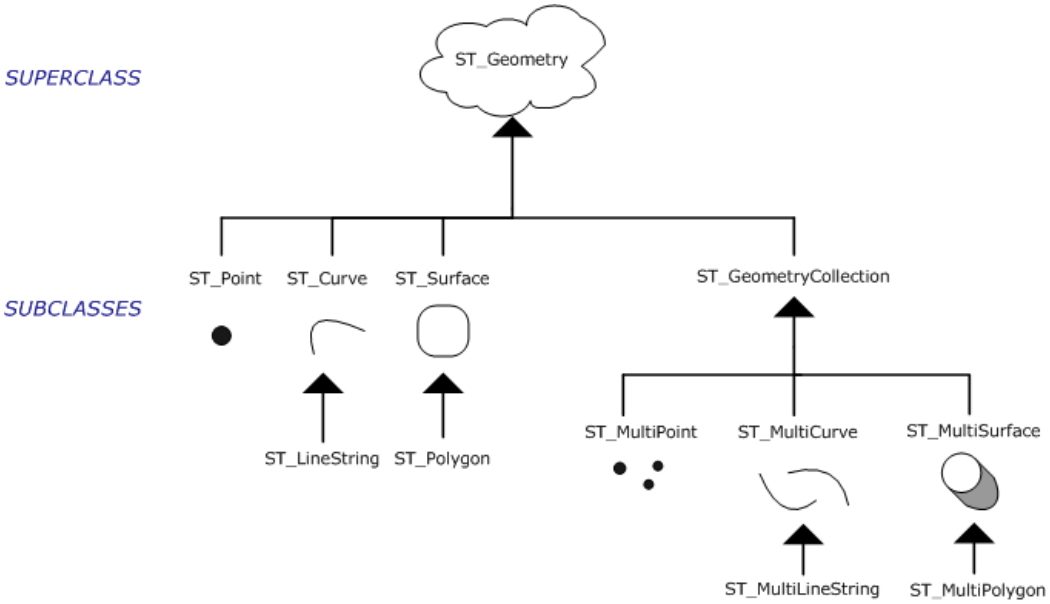


图-OpenGIS 的简单要素数据模型

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

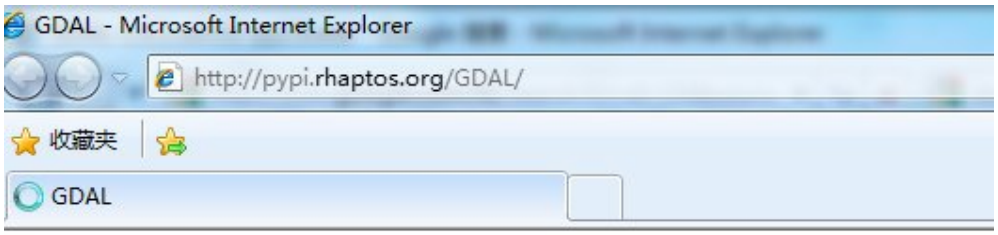


ESRI 的 ST_Geometry 数据存储模型

GDAL/OGR 是非常著名的开源 GIS 库，GDAL 是对栅格数据进行操作，而 OGR 是对矢量数据进行操作，它们相当于一个通用数据访问库。甚至 ESRI 的产品中都用了此库。

安装

ArcGIS 10 Desktop 装了 Python2.6，那么我们就不用装这个了，然后去下面的地址找到 gdal 的 Python 包 GDAL-1.6.1.win32-py2.6.exe，自动安装到 python 的安装目录 E:\Python26\ArcGIS10.0\Lib\site-packages

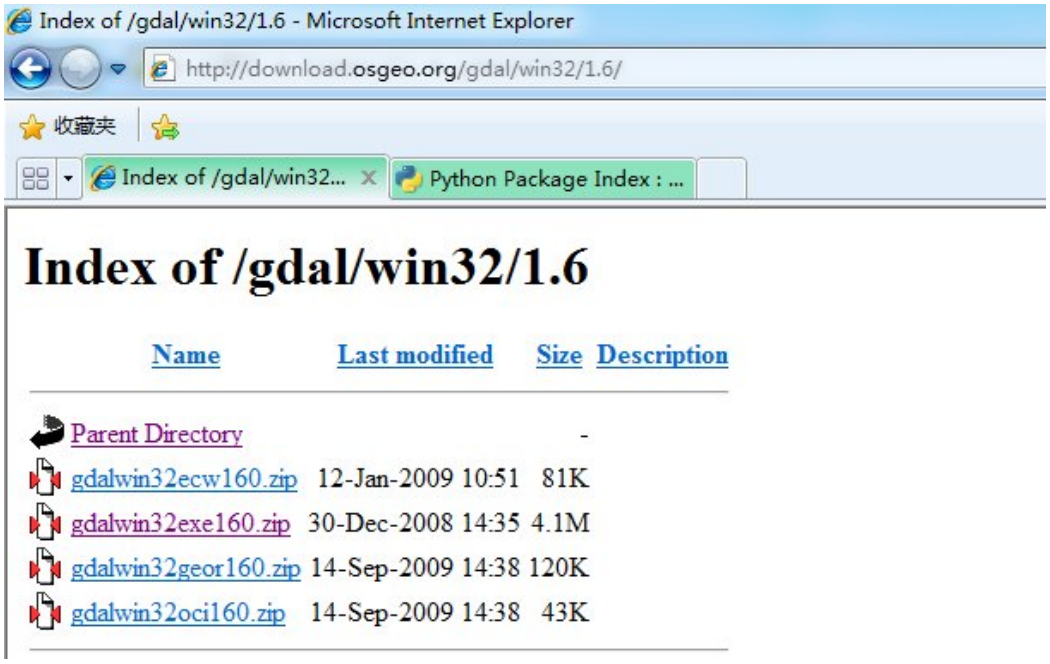


GDAL

GDAL-1.7.1.tar.gz
GDAL-1.7.0.tar.gz
GDAL-1.6.1-py2.6-win32.egg
GDAL-1.6.1-py2.5-win32.egg
GDAL-1.6.1-py2.4-win32.egg
GDAL-1.6.1.win32-py2.6.exe
GDAL-1.6.1.win32-py2.5.exe
GDAL-1.6.1.win32-py2.4.exe
GDAL-1.6.1.tar.gz
GDAL-1.6.0-py2.6-win32.egg
GDAL-1.6.0-py2.5-win32.egg
GDAL-1.6.0-py2.4-win32.egg
GDAL-1.6.0.win32-py2.6.exe
GDAL-1.6.0.win32-py2.5.exe
GDAL-1.6.0.win32-py2.4.exe
GDAL-1.6.0.tar.gz
GDAL-1.5.2-py2.5-win32.egg
GDAL-1.5.2-py2.4-win32.egg
GDAL-1.5.2-py2.5-macosx-10.5-i386.egg
GDAL-1.5.2.win32-py2.5.exe
GDAL-1.5.2.win32-py2.4.exe
GDAL-1.5.2.tar.gz
GDAL-1.5.0-py2.5-win32.egg
GDAL-1.5.0-py2.4-win32.egg
GDAL-1.5.0-py2.5-macosx-10.5-i386.egg
GDAL-1.5.0.win32-py2.5.exe
GDAL-1.5.0.win32-py2.4.exe
GDAL-1.5.0.tar.gz

下载下面的文件

(GDAL Windows binaries)



解压后按照 README_EXE.TXT 的说明添加环境变量，我的路径为 C:\gdalwin32-1.6

添加数据路径的变量

为了验证是否安装成功，可以执行下列命令:

```
>>>from osgeo import gdal
>>>from osgeo.gdalconst import *
```

```
>>>dataset=gdal.Open("C:\data\liuyu.jpg",GA_ReadOnly)
>>>dataset.GetDriver().ShortName
```

对矢量数据的操作时通过 OGR，对每种数据有一个驱动，因此在操作数据的时候，我们先要获取相应数据格式的驱动：如 driver = ogr.GetDriverByName(‘ ESRI Shapefile ’)，但是对于栅格数据略有不同；GDAL 数据驱动，与 OGR 数据驱动类似，需要先创建某一类型的数据驱动，再创建相应的栅格数据集。一次性注册所有的数据驱动，但是只能读不能写：gdal.AllRegister().单独注册某一类型的数据驱动，这样的话可以读也可以写，可以新建数据集：

```
driver = gdal.GetDriverByName('HFA')
driver.Register()
```

导入库：

```
from osgeo import ogr
```

要读取某种类型的数据，必须要先载入数据驱动，也就是初始化一个对象，让它“知道”某种数据结构。

```
from osgeo import ogr
driver = ogr.GetDriverByName( ‘ ESRI Shapefile ’ )
```

数据驱动 driver 的 open()方法返回一个数据源对象

```
open(<filename>, <update>)
```

其中 update 为 0 是只读，为 1 是可写

例如：

```
>>>from osgeo import ogr
>>>driver = ogr.GetDriverByName('ESRI Shapefile')
>>>inputs="C:\Export_Output.shp"
```

```
>>>datasource = driver.Open(inputs,0)
>>>if datasource is None:
    print "could not open"
```

读取数据层

```
layer = datasource.GetLayer()
```

一般 ESRI 的 shapefile 都是填 0 的，如果不填的话默认也是 0.

```
 >>> # 读取数据
>>>layer = datasource.GetLayer()
>>> n=layer.GetFeatureCount()
>>>n
255
>>>feature=layer.GetNextFeature()
>>> #注意里面没有 FeatureClass 这个说法
FID=feature.GetFID()
>>> FID
0
```

读出上下左右边界

```
extent = layer.GetExtent()
print 'extent:', extent
print 'ul:', extent[0], extent[3]
```

```
print 'lr:', extent[1], extent[2]
```

读取字段的个数

```
>>>fieldscount=feature.GetFieldCount()
>>>fieldscount
23
```

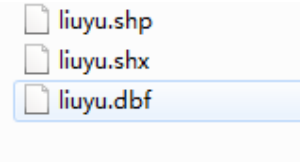
释放内存

```
feature.Destroy()
```

关闭数据源，相当于文件系统操作中的关闭文件

```
datasource.Destroy()
```

```
>>> #创建文件
>>>datasource=driver.CreateDataSource("C:\liuyu.shp")
>>>shplayer=datasource.CreateLayer("liuyu",geom_type=ogr.wkbPoint)
```




```
>>>shplayer=datasource.CreateLayer("liuyu",geom_type=ogr.wkbPoint)
```

shplayer 相当于我们的 FeatureClass，因为我们将要素写入文件是通过 shplayer 的，还记得 C#和 Engine 是如何操作数据的

要添加一个新字段，只能在 layer 里面加，而且还不能有数据

添加的字段如果是字符串，还要设定宽度

```
fieldDefn = ogr.FieldDefn('id', ogr.OFTString)
```

```
fieldDefn.SetWidth(4)

shplayer.CreateField(fieldDefn)
```

添加一个新的 feature，首先得完成上一步，把字段 field 都添加齐了

然后从 shpfile 中读取相应的 feature 类型，并创建 feature

```
fieldDefn = ogr.FieldDefn('id', ogr.OFTString)

>>>fieldDefn.SetWidth(4)

>>>shplayer.CreateField(fieldDefn)

0

>>>featureDefn = shplayer.FeatureDefn()

>>>feature = ogr.Feature(featureDefn)

)
```

将 feature 写入 shplayer

建立点对象

```
point = ogr.Geometry(ogr.wkbPoint)

>>>point.AddPoint(10,10)

>>>feature.SetGeometry(point)

0

>>>point.GetX()

10.0
```

创建要素

```
>>>shpfile.CreateFeature(feature)

0

>>> n=shplayer.GetFeatureCount()

>>>n

1

>>>field=feature.SetField("id","liu")
```

刚才说过 Engine 中操作数据是通过 FeatureClass 的，而通过上面的 Python 可以看出 Python 中操作数据是通过这个 shplayer 的。

```
//IFeatureClass CreateFeature Example

    public void IFeatureClass_CreateFeature_Example(IFeatureClass featureClass)
    {
        //Function is designed to work with polyline data
        if (featureClass.ShapeType !=
ESRI.ArcGIS.Geometry.esriGeometryType.esriGeometryPolyline) { return; }

        //create a geometry for the features shape
        ESRI.ArcGIS.Geometry.IPolyline polyline = new
ESRI.ArcGIS.Geometry.PolylineClass();
        ESRI.ArcGIS.Geometry.IPoint point = new ESRI.ArcGIS.Geometry.PointClass();
        point.X = 0; point.Y = 0;
        polyline.FromPoint = point;

        point = new ESRI.ArcGIS.Geometry.PointClass();
        point.X = 10; point.Y = 10;
        polyline.ToPoint = point;

        IFeature feature = featureClass.CreateFeature();
```

```
        //Apply the constructed shape to the new features shape
        feature.Shape = polyline;


        ISubtypes subtypes = (ISubtypes)featureClass;
        IRowSubtypes rowSubtypes = (IRowSubtypes)feature;
        if (subtypes.HasSubtype)// does the feature class have subtypes?
        {
            rowSubtypes.SubtypeCode = 1; //in this example 1 represents the Primary
Pipeline subtype
        }


        // initalize any default values that the feature has
        rowSubtypes.InitDefaultValues();


        //Commit the default values in the feature to the database
        feature.Store();


        //update the value on a string field that indicates who installed the feature.
        feature.set_Value(feature.Fields.FindField("InstalledBy"), "K Johnston");


        //Commit the updated values in the feature to the database
        feature.Store();
    }
```

如果用我们 ArcGIS 装了之后的那个 Python 环境敲写代码，一两句的话，还能忍受，但是如果大量的话，那可是要死人的，所以我们用 Eclipse 作为环境

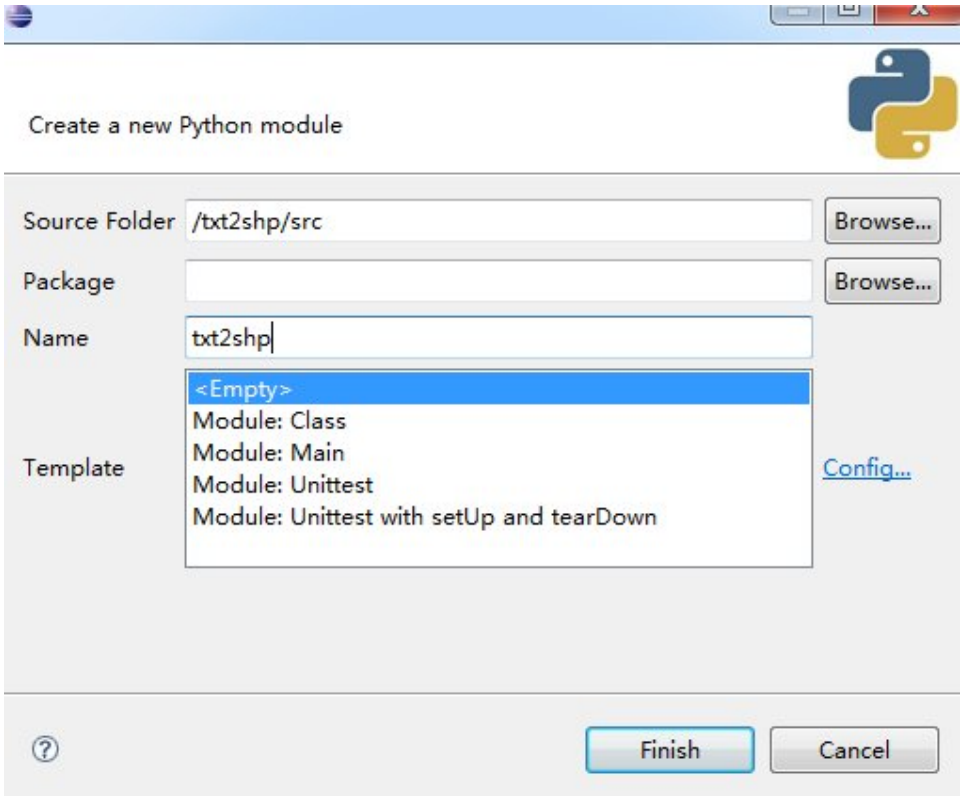
通过选择'help'（帮助）'-software updates'（软件更新）>>Find and Insatll>>Search for new Feature to Install 下一中添加一个 New Remote Site 为 http://pydev.sf.net/updates/ ,然后就可以点击下一步自动升级了,这个是直接从网站上下载安装的。这个和 Eclipse 安装其他插件一样，或者将这个插件单独下载，然后将插件的文件复制到 Eclipse 的相关目录中

当我们用这个 Eclipse 调试的时候,调试速度太慢了,看着半天没反应,真难受，看到下面的红字

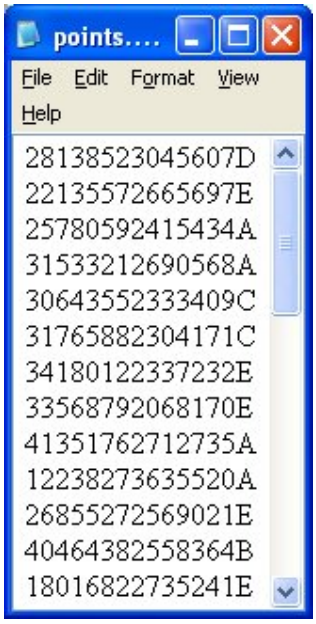
```
pydev debugger: warning: psyco not available for speedups (the debugger
```

```
will still work correctly, but a bit slower)

pydev debugger: starting
```

于是在网上找到 psyco（我就这个为调试加速器），下载一个适合自己平台的，安装就可以了，我没有装，我的代码不是很多最多就 200 行左右，我相信我的机器，哈哈



下面这个数据是很简单的，前面六位（1-6）表示 X 坐标，而紧接着 X 的六位（7-12）表示 Y 坐标，我们就将这个文件利用 OGR 弄成 shp 数据，下面我们看下这个数据：



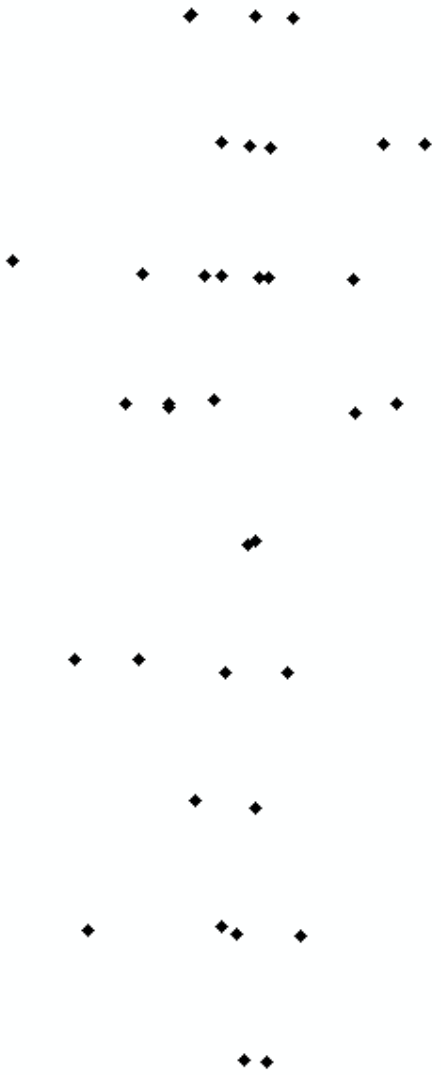

```
import sys
from osgeo import ogr
ogr.RegisterAll()
driver=ogr.GetDriverByName('ESRI Shapefile')
# "C:\txt2shp.shp" this is wrong
ds=driver.CreateDataSource("C:\liuyu.shp")
layer=ds.CreateLayer("liuyu", geom_type=ogr.wkbPoint)

# Create fields
field1=ogr.FieldDefn("Num",ogr.OFTInteger)
field2=ogr.FieldDefn("Char",ogr.OFTString)
field2.SetWidth(4)
layer.CreateField(field1)
layer.CreateField(field2)
featuredefn=layer.GetLayerDefn()
feature=ogr.Feature(featuredefn)
# open txt
txtfile =open("C:\points.csp","r")
line = txtfile.readline()
while line:
cleanline = line.replace('\n','')
point = ogr.Geometry(ogr.wkbPoint)
```

```
point.AddPoint(float(line[0:6]),float(line[6:12]))
feature.SetGeometry(point)
feature.SetField(0,int(line[12:14]))
feature.SetField(1,line[14:-1])
layer.CreateFeature(feature)
line = txtfile.readline()
```

下面是生成后的的数据表

FID	Shape	Num	Char
0	Point	7	D
1	Point	97	E
2	Point	34	A
3	Point	68	A
4	Point	9	C
5	Point	71	C
6	Point	32	E
7	Point	70	E
8	Point	35	A
9	Point	20	A
10	Point	21	E
11	Point	64	B
12	Point	41	E
13	Point	80	C
14	Point	6	C
15	Point	76	E
16	Point	94	A
17	Point	0	E
18	Point	17	B
19	Point	96	E
20	Point	99	E
21	Point	10	B
22	Point	96	B
23	Point	95	E
24	Point	83	A
25	Point	11	D
26	Point	29	E
27	Point	48	E
28	Point	1	D
29	Point	7	E
30	Point	70	C
31	Point	98	C
32	Point	45	E
33	Point	75	C
34	Point	77	E
35	Point	34	

下面的是图形



建立新的几何形状

建立空的 geometry 对象：ogr.Geometry

定义各种不同的 geometry 使用的方法是不一样的(point, line, polygon, etc)

新建点，如我们上面的例子

```
 point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(float(line[0:6]),float(line[6:12]))
```

新建 line

使用 AddPoint(<x>, <y>, [<z>])添加点

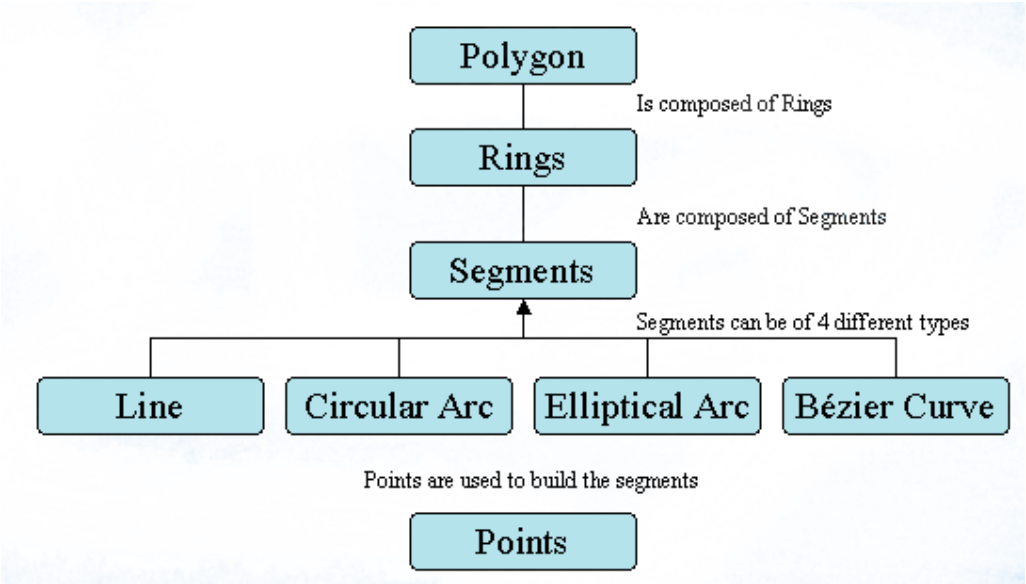
使用 SetPoint(<index>, <x>, <y>, [<z>])更改点的坐标

读取 0 号点的 x 坐标和 y 坐标

```
print line.GetX(0)

print line.GetY(0)
```

新建面

对于面的构建，如果用过 AE 的话，我们都知道，面是有一组 ring 构成册，而 ring 又是由 segment 构成，segment 又是由 point 构成，我大学的时候老师说过这样一句话矢量数据的本质是 point，而栅格数据的本质是 pixel。我们看一下 AE 中的这个图：



那么 OGR 中的 polygon 呢？它的结构是什么样子？我们从下面的例子是否可以从例子中看到？

```
#coding=gbk
'''
Created on 2010-11-9

@author: 刘宇
'''
```

```
from osgeo import ogr

ogr.RegisterAll()
driver=ogr.GetDriverByName('ESRI Shapefile')
#创建一个文件
ds=driver.CreateDataSource("C:\\test.shp")
shpfile=ds.CreateLayer("test",geom_type=ogr.wkbPolygon)

#创建内环
out=ogr.Geometry(ogr.wkbLinearRing)
out.AddPoint(0,0)
out.AddPoint(1000,0)
out.AddPoint(1000,1000)

out.AddPoint(0,1000)
out.CloseRings()
#创建外环
inring=ogr.Geometry(ogr.wkbLinearRing)
inring.AddPoint(250,250)

inring.AddPoint(750,250)

inring.AddPoint(750,750)

inring.AddPoint(250,750)
inring.AddPoint(250,250)
polygon=ogr.Geometry(ogr.wkbPolygon)
#面是由环构成的
polygon.AddGeometry(out)
polygon.AddGeometry(inring)

featuredefn=shpfile.GetLayerDefn()
feature=ogr.Feature(featuredefn)
feature.SetGeometry(polygon)
print polygon.GetArea()
```

下面这句话可以帮我们数数数 polygon 能有几个 ring

```
print polygon.GetGeometryCount()
```

从 polygon 中读取 ring，index 的顺序和创建 polygon 时添加 ring 的顺序相同

```
outring = polygon.GetGeometryRef(0)
```

```
inring = polygon.GetGeometryRef(1)
```

创建复合几何形状 multi geometry

例如 MultiPoint，MultiLineString，MultiPolygon

用 AddGeometry 把普通的几何形状加到复合几何形状中，例如：

```
multipoint = ogr.Geometry(ogr.wkbMultiPoint)
point = ogr.Geometry(ogr.wkbPoint)point = ogr.Geometry(ogr.wkbPoint)
point.AddPoint(10,10)
multipoint.AddGeometry(point)
point.AddPoint(20,20)
multipoint.AddGeometry(point)
```

读取 MultiGeometry 中的 Geometry，方法和从 Polygon 中读取 ring 是一样的，可以说 Polygon 是一种内置的 MultiGeometry。

关于投影 Projections

空间数据有一个东西非常重要，那就是投影，而 OGR 支持

多种多样的 Projections，GDAL 支持 WKT，PROJ.4，ESPG，USGS，ESRI.prj

，使用 SpatialReference 对象

可以从 layer 和 Geometry 中读取 Projections，例如：

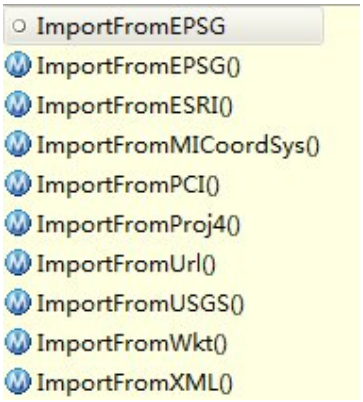
```
spatialRef = layer.GetSpatialRef()
spatialRef = geom.GetSpatialReference()
```

投影信息一般存储在.prj 文件中，如果没有这个文件，上述函数返回 None

建立一个新的 Projection：

首先导入 osr 库，之后使用 osr.SpatialReference()创建 SpatialReference 对象

之后用下列语句向 SpatialReference 对象导入投影信息



导出 Projection，使用下面的语句可以导出为字符串

我们对刚才的的例子加深下，因为我们上面的没有投影，现在我们给它指定一个投影

```
#coding=gbk
'''
Created on 2010-11-9

@author: 刘宇
'''
from osgeo import ogr
```

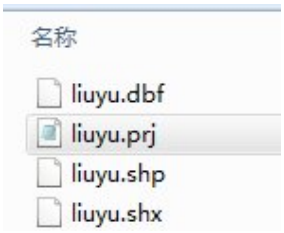
```
from osgeo import osr

ogr.RegisterAll()
driver=ogr.GetDriverByName('ESRI Shapefile')
#创建一个文件
ds=driver.CreateDataSource("C:\\test.shp")
shpfile=ds.CreateLayer("test",geom_type=ogr.wkbPolygon)

#创建内环
out=ogr.Geometry(ogr.wkbLinearRing)
out.AddPoint(0,0)
out.AddPoint(1000,0)
out.AddPoint(1000,1000)

out.AddPoint(0,1000)
out.CloseRings()
#创建外环
inring=ogr.Geometry(ogr.wkbLinearRing)
inring.AddPoint(250,250)

inring.AddPoint(750,250)

inring.AddPoint(750,750)

inring.AddPoint(250,750)
inring.AddPoint(250,250)
polygon=ogr.Geometry(ogr.wkbPolygon)
#面是由环构成的
polygon.AddGeometry(out)
polygon.AddGeometry(inring)

featuredefn=shpfile.GetLayerDefn()
feature=ogr.Feature(featuredefn)
feature.SetGeometry(polygon)
print polygon.GetArea()
print polygon.GetGeometryCount()
if(polygon.GetSpatialReference()==None):
print"OK"
sr=osr.SpatialReference()
sr.ImportFromEPSG(32612)
polygon.AssignSpatialReference(sr)   //指定一个SpatialReference
```

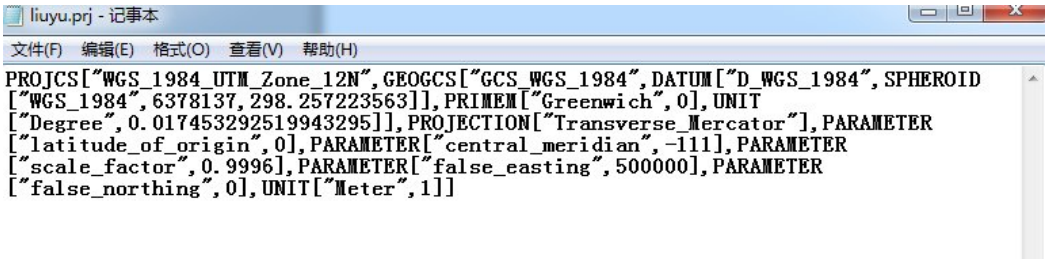
有什么问题？如果我的 geometry 很多，那我是不是要遍历每一个？想一下 ESRI 是怎么搞的？没错 PRJ 文件

```
#coding=gbk
'''
Created on 2010-11-9

@author: 刘宇
'''
from osgeo import ogr
from osgeo import osr

ogr.RegisterAll()
driver=ogr.GetDriverByName('ESRI Shapefile')
#创建一个文件
ds=driver.CreateDataSource("C:\\liuyu.shp")
shpfile=ds.CreateLayer("liuyu",geom_type=ogr.wkbPolygon)

#创建内环
out=ogr.Geometry(ogr.wkbLinearRing)
out.AddPoint(0,0)
out.AddPoint(1000,0)
out.AddPoint(1000,1000)
out.AddPoint(0,1000)
out.CloseRings()
#创建外环
inring=ogr.Geometry(ogr.wkbLinearRing)
inring.AddPoint(250,250)

inring.AddPoint(750,250)

inring.AddPoint(750,750)

inring.AddPoint(250,750)
inring.AddPoint(250,250)
polygon=ogr.Geometry(ogr.wkbPolygon)
#面是由环构成的
polygon.AddGeometry(out)
polygon.AddGeometry(inring)

featuredefn=shpfile.GetLayerDefn()
feature=ogr.Feature(featuredefn)
feature.SetGeometry(polygon)
print polygon.GetArea()
```

```
print polygon.GetGeometryCount()
if(polygon.GetSpatialReference()==None):
print"OK"
sr=osr.SpatialReference()
sr.ImportFromEPSG(32612)
#如果将geometry的投影都改变了，那要一个个的循环，数据量太多的话，很麻烦，可以像ESRI一样提供一个文件，写进去
prjfile=open("C:\liuyu.prj",'w')
#转成字符串
sr.MorphToESRI()
prjfile.write(sr.ExportToWkt())
prjfile.close()
print"OK"
```

上面的代码执行后，我们就可以看到下面的文件





怎么做投影变换？用这个方法TransformTo()

简单的地理数据处理 geoprocessing 可以对比 ArcGIS 提供的

多边形的：

交：poly3.Intersection(poly2)

并：poly3.Union(poly2)

差：poly3.Difference(poly2)

补：poly3.SymmetricDifference(poly2)

geometry 的：

<geom>.Buffer(<distance>) 给 geometry 加 buffer,就是把点线变成多边形，变粗了

<geom1>.Equal(<geom2>) 两个 geometry 相等吗？

<geom1>.Distance(<geom2>) 返回两个 geometry 之间的最短距离

<geom>.GetEnvelope() 信封,有意思,其实就是用一个方框框住这个几何形状，返回四个角的坐标(minx, maxx, miny, maxy)

python 的函数 function,异常 exception 和模块 module 可以从任何一本 python 教材上找到，在此不多赘述。

栅格篇

GDAL 原生支持超过 100 种栅格数据类型，涵盖所有主流 GIS 与 RS 数据格式，如下图：

Arc/Info ASCII Grid	AAIGrid	Yes	Yes	2GB
ADRG/ARC Digitilized Raster Graphics (.gen/.thf)	ADRG	Yes	Yes	--
Arc/Info Binary Grid (.adf)	AIG	No	Yes	--
AIRSAR Polarimetric	AIRSAR	No	No	--
Magellan BLX Topo (.blx, .xlb)	BLX	Yes	Yes	--
Bathymetry Attributed Grid (.bag)	BAG	No	Yes	2GiB
Microsoft Windows Device Independent Bitmap (.bmp)	BMP	Yes	Yes	4GiB
BSB Nautical Chart Format (.kap)	BSB	No	Yes	--
VTP Binary Terrain Format (.bt)	BT	Yes	Yes	--
CEOS (Spot for instance)	CEOS	No	No	--
DRDC COASP SAR Processor Raster	COASP	No	No	--
TerraSAR-X Complex SAR Data Product	COSAR	No	No	--
Convair PolGASP data	CPG	No	Yes	--
Spot DIMAP (metadata.dim)	DIMAP	No	Yes	--
ELAS DIPEx	DIPEx	No	Yes	--
DODS / OPeNDAP	DODS	No	Yes	--
First Generation USGS DOQ (.doq)	DOQ1	No	Yes	--
New Labelled USGS DOQ (.doq)	DOQ2	No	Yes	--
Military Elevation Data (.dt0, .dt1, .dt2)	DTED	Yes	Yes	--
ERMapper Compressed Wavelets (.ecw)	ECW	Yes	Yes	
ESRI .hdr Labelled	EHdr	Yes	Yes	No limits
Erdas Imagine Raw	EIR	No	Yes	--
NASA ELAS	ELAS	Yes	Yes	--
ENVI .hdr Labelled Raster	ENVI	Yes	Yes	No limits
Epsilon - Wavelet compressed images	EPSILON	Yes	No	--
ERMapper (.ers)	ERS	Yes	Yes	
Envisat Image Product (.n1)	ESAT	No	No	--
FOSAT FAST Format	FAST	No	Yes	--

完整的支持列表可以参考 http://www.gdal.org/formats_list.html

我们知道对于栅格数据，从数学的角度来考虑的话就是一个矩阵，从我们在学习栅格数据和矢量数据的时候，我们知道栅格数据便于建立模型，这是因为他的结构的原因，我们看一下几个相关的模块。

相关模块

1. Numeric：高速的数组处理，对栅格数据尤其重要
2. NumPy：下一代的 Numeric
3. 更强大的 gis 库 http://www.gispython.org/

导入 GDAL 支持库

```
from osgeo import gdal, gdalconst
```

GDAL 数据驱动，与 OGR 数据驱动类似，需要先创建某一类型的数据驱动，再创建响应的栅格数据集。一次性注册所有的数据驱动，但是只能读不能写：gdal.AllRegister()单独注册某一类型的数据驱动，这样的话可以读也可以写，可以新建数据集：

```
driver = gdal.GetDriverByName('GTiff ')
driver.Register()
```

```
>>>from osgeo import gdal
>>>from osgeo.gdalconst import *
>>>driver=gdal.GetDriverByName('JPEG')
>>>driver.Register()
15
>>>ds=gdal.Open("C:\data\liuyu.jpg",GA_ReadOnly)
>>> band=ds.GetRasterBand(1)//获取第一个波段
>>> rown=band.XSize //没注意看这个到底是行还是列
>>>rown
3888
>>>cown=band.YSize
>>>cown
```

```
2592
>>> a=band.ReadAsArray() //将波段的数据用数组表示
>>>a
array([[ 5,  4,  4, ..., 53, 55, 57],
[ 5,  5,  5, ..., 48, 49, 51],
[ 6,  6,  7, ..., 45, 45, 46],
       ...,
       [81, 81, 82, ..., 61, 62, 64],
       [81, 82, 85, ..., 67, 66, 66],
       [87, 87, 87, ..., 72, 68, 65]], dtype=uint8)
>>>
```

Engine 中是怎么读取栅格数据的？

```
IRasterProps rasterProps = (IRasterProps)clipRaster;
int dHeight = rasterProps.Height;//当前栅格数据集的行数
int dWidth = rasterProps.Width; //当前栅格数据集的列数
double dX = rasterProps.MeanCellSize().X; //栅格的宽度
double dY = rasterProps.MeanCellSize().Y; //栅格的高度
IEnvelope extent=rasterProps.Extent; //当前栅格数据集的范围
rstPixelType pixelType=rasterProps.PixelType; //当前栅格像素类型
IPnt pntSize = new PntClass();
pntSize.SetCoords(dX, dY);
IPixelBlock pixelBlock = clipRaster.CreatePixelBlock(pntSize);//会在后面说这个块，ESRI 将栅格数据是按照块来存储的，有什么好处呢？
IPnt pnt = new PntClass();
for (int i = 0; i < dHeight; i++)
for (int j = 0; j < dWidth; j++)
{
      pnt.SetCoords(i, j);
      clipRaster.Read(pnt, pixelBlock);
     if (pixelBlock != null)
     {
     object obj = pixelBlock.GetVal(0, 0, 0);
```

```
    MessageBox.Show( Convert.ToUInt32(obj).ToString());
    }
}
```

对于栅格数据我不想做过的的介绍，刚才说了栅格数据从书序的角度来看就是一个矩阵，那么关于矩阵的操作，在学习编程的时候都会接触到，而不同的是空间数据将这些栅格数据用自己的方式存储，所以栅格数据大多数是关于数学计算，我想这可能也是栅格数据易于建模的原因之一吧，我弄这个材料不是将每一个细节都写出来，再说了，我对这些也不是很懂，如果能将方法和自己的体会告诉别人，这也未必不是一件快乐的事情。我们应该感谢那些将自己的东西乐于和别人分享的人，其实很多资料通过 google 都可以找到，但是那些人将自己搜集到的资料整理下来，并写成博文等与我们分享，让我们免遭奔波，其实怎么样快速的获取自己需要的材料并及时的解决自己的问题，这个才是王道。下来我们用一个例子结束这个栅格数据的描述：

```
# import modules
import os, numpy, sys, time
from osgeo import gdal, ogr
from osgeo.gdalconst import *

startTime = time.time()

# set the working directory
os.chdir(r'C:\Data\Conferences\UGIC2009\data')

# register all of the GDAL drivers
gdal.AllRegister()

# open the image
```

```
inDs = gdal.Open('aster.img', GA_ReadOnly)
if inDs is None:
print 'Could not open aster.img'
sys.exit(1)

# get image size
rows = inDs.RasterYSize
cols = inDs.RasterXSize
bands = inDs.RasterCount

# get the bands and block sizes
inBand2 = inDs.GetRasterBand(2)
inBand3 = inDs.GetRasterBand(3)
blockSizes = inBand2.GetBlockSize()
xBlockSize = blockSizes[0]
yBlockSize = blockSizes[1]

# create the output image
driver = gdal.GetDriverByName('HFA')
outDs = driver.Create('ndvi.img', cols, rows, 1, GDT_Float32)
if outDs is None:
print 'Could not create ndvi.img'
sys.exit(1)
outBand = outDs.GetRasterBand(1)
```

我们知道栅格数据的容量一般很大，怎么样调高效率呢，如果按照数组的方式一个一个像素的读取，将整个栅格数据集都塞进二维数组也不是不可以，但是这样占的内存很多。ArcGIS 用数据库管理栅格数据的时候是按照 block，我也不知道

这个具体怎么翻译，说块是字面意思，瓦片好像也能说得清，就叫做块吧。用块存储数据，只把要用的那一块放进内存，这样就节省了资源。有的格式，例如 GeoTiff 没有平铺，一行是一个 block。Erdas imagine 格式则按 64x64 像素平铺。如果一行是一个 block，那么按行读取是比较节省资源的。

```
# loop through the rows
for i in range(0, rows, yBlockSize):
if i + yBlockSize < rows:
numRows = yBlockSize
else:
numRows = rows - i

  # loop through the columns
for j in range(0, cols, xBlockSize):
if j + xBlockSize < cols:
numCols = xBlockSize
else:
numCols = cols - j

    # read the data in
    data2 = inBand2.ReadAsArray(j, i, numCols,
numRows).astype(numpy.float)
    data3 = inBand3.ReadAsArray(j, i, numCols,
numRows).astype(numpy.float)

    # do the calculations
mask = numpy.greater(data2 + data3, 0)
ndvi = numpy.choose(mask, (-99, (data3 - data2) / (data3 + data2 +
0.00000000001)))
```

```
    # write the data
outBand.WriteArray(ndvi, j, i)

# flush data to disk, set the NoData value and calculate stats
outBand.FlushCache()
outBand.SetNoDataValue(-99)
stats = outBand.GetStatistics(0, 1)

# georeference the image and set the projection
outDs.SetGeoTransform(inDs.GetGeoTransform())
outDs.SetProjection(inDs.GetProjection())

# build pyramids
gdal.SetConfigOption('HFA_USE_RRD', 'YES')
outDs.BuildOverviews(overviewlist=[2,4,8,16,32,64,128])

inDs = None
outDs = None

print 'script took', time.time() - startTime, 'seconds to run'
```

这个代码来自 http://www.gis.usu.edu/~chrisg/pytho 并且有

相关资料，包括讲义、源码、数据样例，感谢这个作者

从沙总给我这个任务开始，其实在这之前用的 Python 也就是 ArcGIS 提供的那些，而对它的语法了解甚少，这次的这个机会然我从头到尾看了遍 Python，Python 的代码块是按照缩行来控制，这个我有点受不了，还是习惯 donet 的那个玩意儿

，也愿意将自己遇到的一些问题和解决方法与大家分享。

沙总给我的任务是让我自己用 Python 写 IDW 不用 ArcGIS 提供的的 IDW 工具，IDW 的算法不是很难,但是怎么生成一个 Raster 呢？前面说过 Raster 从某种角度来看就是数组，于是我决定将计算的结果写到数组中然后存成 txt 文件，我们知道 ArcGIS 提供了一个文本到栅格的工具，但是这个文本有自己的格式，如下哦：

```
NCOLS 480
NROWS 450
XLLCORNER 378922
YLLCORNER 4072345
CELLSIZE 30
NODATA_VALUE -32768
43 2 45 7 3 56 2 5 23 65 34 6 32 54 57 34
35 45 65 34 2 6 78 4 2 6 89 3 2 7 45 23 5 ...
```

我暂且将前面六行叫做头文件，这个很容易想到就是用“ 字典 “ 来存储，但是，我在 Python 中用“ 字典 ” 将这几个按顺序存进去，然后有写入到文件的时候，根本就不是这个顺序，我纳闷了一会儿，然后又将它输出的顺序和我写进去的顺序比对后重新组织这个字典，结果还是然我失望，最后才发现 Python 中的“ 字典 “ 是没有顺序的。于是我放弃了字典这个结构，采用元组，OK 很完美

如下：

```
dictionary=(("NCOLS",ncols),("NROWS",nrows),("XLLCORNER",xleft),("YLL
CORNER",ytop),("CELLSIZE",string.atof(cellSize)),("NODATA_VALUE",0))
```

Python中没有数组这个结构，这我很郁闷，如果有数组的话，其余的就是操作这个数组然后，将这个数组写进去，但是Python中可以构造我们在其他语言中遇到的数组。如下：

```
#Arr=[[0]*ncols]*nrows 这个是复制这个玩意儿害死我了，让我查了整整一天
```

注意我后面的注释，我构造了这个数组，但是但我将Raster生成的时候，我发现我的栅格的每一列是同一个数值，我开始怀疑我是在写入文件的时候，将数组的第一行写了N次，下面是我的代码：

```
#  arr2txt 将数组写成txt文件，其中dictionary为头文件内容
defarr2txt(filename,dictionary,arr):
filehandle=open(filename,'w')
for i in range(0,len(dictionary)):
strs=dictionary[i][0]+" "+str(dictionary[i][1])
filehandle.write(strs)
filehandle.write("\r\n")
for row in arr:
      st=""
for col in row:
         stspace=str(col)+" "
st=st+stspace
sr=st[0:len(st)-1]
filehandle.write(sr)
filehandle.write("\r\n")
filehandle.close()
```

我在 IDE 中自己搞了一个小的数组，然后将文件生成，发现没有问题，于是我又怀疑，是不是我在计算的时候将第一行计算了 N 次？

```
#  calculate the values
   j=0;
while j<nrows:
      i=0;
      y1=ytop-(j+1)* (string.atof(cellSize)) +0.5*
(string.atof(cellSize))
while i<ncols:
         a1=0
         a2=0
x1=xleft+(i+1)* (string.atof(cellSize))-0.5*(string.atof(cellSize))
         n=0
while n< PointCount:
            x=Xlist[n]
            y=Ylist[n]
            z=Zlist[n]
n+=1;
#  利用平面距离的算法
            d= idw_pdistance(x1,y1,x,y,string.atof(Power))
if d<Radius:
               a2=a2+1/d
               a1=a1+z/d

if a2<>0:
```

```
        Arr[j][i]=a1/a2
    i+=1;
       j+=1;
```

我用 matlab 搞了几个简单的数据模拟了下，然后结合笔算，笔算的结果和 matlab 下的结果相同，我就奇怪了？我在上面的代码中加了几句，将栅格的数据全部打印出来

```
cnn=1;
while j<nrows:
       i=0;
       y1=ytop-(j+1)* (string.atof(cellSize)) +0.5*
(string.atof(cellSize))
while i<ncols:
          a1=0
          a2=0
x1=xleft+(i+1)*
(string.atof(cellSize))-0.5*(string.atof(cellSize))
          n=0
while n< PointCount:
             x=Xlist[n]
             y=Ylist[n]
             z=Zlist[n]
n+=1;
             #  利用平面距离的算法
             d= idw_pdistance(x1,y1,x,y,string.atof(Power))
if d<Radius:
                a2=a2+1/d
                a1=a1+z/d

if a2<>0:
```

```
                Arr[j][i]=a1/a2
                # 为什么这里打印出来的值在 Arr 中找不到
print Arr[j][i]
                cnn+=1
            i+=1;
        j+=1;
    # 打印计算的个数
print cnn //
print "OK"
```

我没发现相同的？害死我了，我将打印的结果复制在 word 中，因为个字大小设置的有点小，害得我粘贴了 35 页，⊙﹏⊙b 汗。到底怎么回事呢？我又添加了几句

```
cnn=1;
while j<nrows:
        i=0;
        y1=ytop-(j+1)* (string.atof(cellSize)) +0.5*
(string.atof(cellSize))
while i<ncols:
            a1=0
            a2=0
x1=xleft+(i+1)*
(string.atof(cellSize))-0.5*(string.atof(cellSize))
            n=0
while n< PointCount:
                x=Xlist[n]
                y=Ylist[n]
                z=Zlist[n]
n+=1;
                #  利用平面距离的算法
                d= idw_pdistance(x1,y1,x,y,string.atof(Power))
```

```
if d<Radius:
                a2=a2+1/d
                a1=a1+z/d


if a2<>0:
             Arr[j][i]=a1/a2
             # 为什么这里打印出来的值在 Arr 中找不到
print Arr[j][i]
             cnn+=1
          i+=1;


       j+=1;
   # 打印计算的个数
print cnn //
print "OK"

# 打印数组

    for j in range(0,nrows):
       for i in range(0,ncols):
print Arr[j][i]
```

怪异的现象发生了，这里的数值和我刚才打印的不一样，而且存在重复。问题定位到这里，是这个数组的问题。

经过一天的折腾，才发现，如果按照我的那种方式构造数组，其实是将第一行复制了 N 次。证据如下：

```
    >>> Arr=[[0]*10]*10

    >>> Arr

    [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

    >>>Arr[0][0]=1

    >>> Arr

    [[1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0,
0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1,
0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0,
0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

看来这样构造的数组用不成，我想骂人了，借助强大的 google 我找到了另外一种构造数组的方法：

```
Arr = [[0    column    range(ncols)]     row    range(nrows)]

    >>> A=[[0 for column in range(10)] for row in range(10)]

    >>>A[0][0]=1

    >>> A

    [[1, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

这个结果才是我想要的，哈哈，总算没有白折腾，Matlab 都用上了，代码如下：

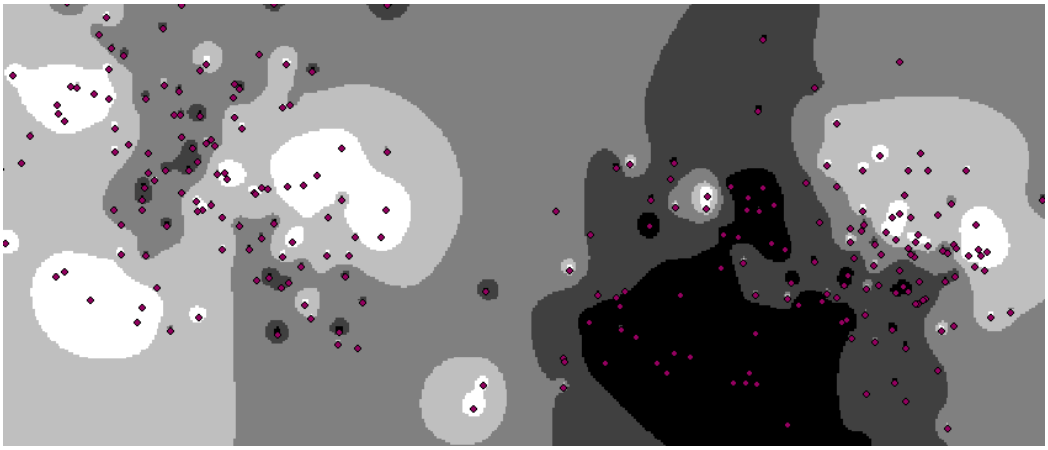
```
Arr = [[0for column inrange(ncols)] for row in range(nrows)]

#   calculate the values
```

```
    j=0;
while j<nrows:
       i=0;
       y1=ytop-(j+1)* (string.atof(cellSize)) +0.5*
(string.atof(cellSize))
while i<ncols:
          a1=0
          a2=0
x1=xleft+(i+1)* (string.atof(cellSize))-0.5*(string.atof(cellSize))
          n=0
while n< PointCount:
              x=Xlist[n]
              y=Ylist[n]
              z=Zlist[n]
n+=1;
#  利用平面距离的算法
              d= idw_pdistance(x1,y1,x,y,string.atof(Power))
if d<Radius:
                 a2=a2+1/d
                 a1=a1+z/d

if a2<>0:
              Arr[j][i]=a1/a2

          i+=1;
       j+=1;
```

当这个 txt 生成后，然后利用 ArcGIS 提供的工具

```
rasterType = "FLOAT"

# Execute ASCIIToRaster
gp.outputCoordinateSystem = SRinputPts
    gp.ASCIIToRaster_conversion(outTxt, outRaster, rasterType)
```

结果如下：



IDW 这个是第一步，还有其他的功能有待完善，稍后等我将这个完善了，继续和大家分享.只是在插值的时候，这个距离，说白了我也不知道到底给多少？上面这幅图，是我不停的设置参数，才搞出来的，突然间觉得这些插值到底有精读没？想我第一次插值，我给的距离阈值是 1250，结果我的 txt 文件中都是 nodata 也就是 0（我用 0 表示 nodata），但是跟踪我的距离，很大，于是我将距离求了平方根，才得出上面这个图，于是我就用 ArcMap 中的默认参数做了一个 IDW，那结果真扯淡，我都不想再这里截图了……，插值算法和思路上面说的清楚了，关于以后的工作，等我做完了在和大家分享。

感谢 Google,在我搞这个的时候,利用这个搜索,然我找到很多有用的资料，也同时感谢那些愿意将自己的心得和大家分享的人，这个人包不包括我，那我就不知道了……