Работа с векторными данными при помощи OGR и Python

Обсудить в форуме Комментариев — 57

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/ogr-python.html

Руководство по использованию OGR Python API, частично основано на «OGR API tutorial»

OGR — свободная библиотека для работы с векторными данными. Утилиты командной строки, входящие в состав библиотеки широко используются для выполнения разнообразных задач (примеры).

Благодаря наличию развитого API можно работать с функциями OGR из многих языков программирования. Эта статья описывает работу с OGR API через Python. При написании статьи использовались материалы OGR API tutorial.

Содержание

Подготовка

Будем исходить из того, что все программное обеспечение установлено при помощи установщика OSGeo4W. Для использования OGR совместно с Python необходимо наличие соответствующих обёрток для этого языка (bindings). В стеке программ OSGeo4W нужный пакет называется gdal-python (для GDAL/OGR 1.5.x) или gdal16-python (для GDAL/OGR 1.6.x). Рекомендую использовать версию 1.6.x.

GDAL Python API состоит из пяти основных модулей и пяти дополнительных (существуют для совместимости со старыми версиями):

- gdal — Python интерфейс к библиотеке GDAL
- ogr — Python интерфейс к библиотеке OGR
- osr — работа с системами координат
- gdal_array — вспомогательные функции
- gdalconst — константы

Подключить их можно командами:

```
# основные
from osgeo import gdal
from osgeo import ogr
from osgeo import osr
from osgeo import gdal_array
from osgeo import gdalconst

# версии для совместимости. Будут удалены в версии 2.0
import gdal
```

```
import ogr
import osr
import gdalnumeric
import gdalconst
```

Если используется GDAL/OGR версии 1.5 и выше, рекомендуется использовать «основные» модули. А для случаев, когда необходимо использовать код написанный ранее можно сделать проверку

```
try:
  from osgeo import ogr
except ImportError:
  import ogr
```

В большинстве случаев достаточно подключить только модуль ogr.

Открытие файла

Для открытия векторного набора данных используем функцию ogr.Open(). В качестве источника данных может выступать файл, база данных, каталог с файлами и даже удаленный web-сервис, все зависит от используемого драйвера. В качестве аргументов функция ogr.Open() принимает полный путь к набору данных и необязательную константу, описывающую режим открытия. Если константа опущена, то подразумевается режим только для чтения. В случае успеха функция вернет объект OGRDataSource, в противном случае — None.

```
import osgeo.ogr as ogr
ogrData = ogr.Open( "/home/alex/test/points.shp", False )
# или так
# ogrData = ogr.Open( "/home/alex/test/points.shp" )

# проверяем все ли в порядке
if ogrData is None:
  print "ERROR: open failed"
  sys.exit( 1 )
```

Работа со слоями

Т.к. с одним набором данных может быть ассоциированно множество слоев, необходимо указать рабочий слой. Количество слоев можно узнать при помощи метода GetLayerCount() объекта OGRDataSource. А обратиться к слою можно как по индексу (из диапазона 0..GetLayerCount() - 1), так и по имени:

```
print "Number of layers", ogrData.GetLayerCount()

# обращаемся к слою по индексу
layer = ogrData.GetLayer( 0 )
# или
layer = ogrData[0]
# или по имени
layer = ogrData.GetLayerByName( "points" )
# или
layer = ogrData["points"]
```

В примере опущена проверка полученного значения, но это не значит, что ее не должно быть. В случае ошибки (например, при выходе индекса за допустимые границы) переменная layer будет равна None.

После получения слоя стоит вызвать функцию ResetReading(), чтобы быть уверенными в том, что мы находимся в начале слоя. Но если слой был только что открыт, как в нашем примере, то вызов этой функции можно опустить.

```
if layer is None:
  print "ERROR: can't access layer"
  sys.exit( 1 )
layer.ResetReading()
```

Получение информации и работа с объектами

Количество объектов в слое можно узнать при помощи GetFeatureCount(), вызов GetSpatialRef() покажет информацию о используемой проекции и системе координат. Охват слоя можно получить при помощи GetExtent(), а количество полей — GetFieldCount(). При получении охвата следует помнить, что в зависимости от используемого драйвера значение охвата может зависеть или не зависить от наличия пространственного фильтра, поэтому перед вызовом GetExtent() рекомендуется деактивировать пространственный фильтр.

```
print "Feature count", layer.GetFeatureCount()
print "Layer SRC", layer.GetSpatialRef()
print "Layer extent", layer.GetExtent()
print "Field count", layer.GetFieldCount()
```

После указания рабочего слоя можно приступать к чтению объектов. Для доступа к объектам удобно использовать цикл и функцию GetNextFeature(), которая ввозвращает следующий объект набора или None если достигнут конец набора. Перед получением объектов можно задать пространственный фильтр или фильтр по атрибутам используя SetSpatialFilter()/SetSpatialFilterRect() и SetAttributeFilter() соответственно.

```
feat = layer.GetNextFeature()
while feat is not None:
  # обрабатываем объект
```

Для получения списка полей объекта и их значений используется вот такая конструкция:

```
feat = layer.GetNextFeature()
featDef = layer.GetLayerDefn() # схема (таблица атрибутов) слоя
while feat is not None:
  for i in range( featDef.GetFieldCount() ): # проходим по всем полям
    fieldDef = featDef.GetFieldDefn( i ) # получаем i-тое поле
    print "Field name", fieldDef.GetNameRef() # и выводим информацию
    print "Field type", fieldDef.GetType()
    print "Field value", feat.GetFieldAsString(i)
  feat = layer.GetNextFeature() # переходим к следующему объекту
```

Поля могут иметь разный тип, для каждого типа данных есть соответсвующая функция доступа. Например, для типа OFTInteger — GetFieldAsInteger(), для OFTDateTime — GetFieldAsDateTime() и т.д. Кроме того, адекватное представление значения любого поля можно получить при помощи GetFieldAsString().

GetGeometryRef() вернет геометрию объекта:

```
geom = feat.GetGeometryRef()
if geom is None:
  print "Invalid geometry"
if geom.GetGeometryType() == ogr.wkbPoint:
  print "%.3f, %.3f" % ( geom.GetX(), geom.GetY() )
else:
  print "Non point geometry"
```

Используя OGR полученную геометрию можно экспортировать в различные форматы при помощи функций ExportToGML(), ExportToJson(), ExportToKML(), ExportToWkb(), ExportToWkt() а также выполнять различные действия над ней: ConvexHull(), Buffer(), Difference()...

После окончания работы нужно освободить ресурсы и закрыть набор данных

```
ogrData.Destroy()
```

Сохранение файла

Сохранение данных рассмотрим на примере записи в шейп-файл. Прежде всего, надо указать какой драйвер использовать для записи

```
driverName = "ESRI Shapefile"
drv = ogr.GetDriverByName( driverName )
if drv is None:
    print "%s driver not available.\n" % driverName
```

Если драйвер инициализирован успешно, переходим к созданию набора данных. Драйвер шейп-файлов позволяет создать как отдельный шейп-файл, так и каталог в котором будет находиться один или несколько файлов. Если надо создать один файл, обязательно указываем имя файла с расширением. При необходимости, вторым параметром передаются «тонкие» настройки драйвера в виде списка «имя=значение».

```
ogrData = drv.CreateDataSource( "/home/alex/point_out.shp" )
if ogrData is None:
    print "Creation of output file failed.\n"
    sys.exit( 1 )
```

Далее создаем новый слой. Обязательно нужно указать имя слоя и тип геометрии, система координат указывается по необходимости

```
layer = ogrData.CreateLayer( "point_out", None, ogr.wkbPoint )
if layer is None:
    print "Layer creation failed."
    sys.exit( 1 )
```

Слой создан, добавим атрибуты описывающие объекты слоя. Добавлять поля (атрибуты) надо до того как буду вставляться объекты. Для каждого поля нужно создать объект OGRField и задать необходимые параметры (имя, тип, точность):

```
fieldDef = ogr.FieldDefn( "FileName", ogr.OFTString ) # имя поля и его тип
fieldDef.SetWidth( 32 ) # длина поля

if layer.CreateField ( fieldDef ) != 0:
    print "Creating field failed.\n"
    sys.exit( 1 )

# для чисел с плавающей точкой можно задать не только длину,
# но и точность
fieldDef = ogr.FieldDefn( "Area", ogr.OFTReal ) # имя поля и его тип
fieldDef.SetWidth( 18 ) # длина поля
fieldDef.SetPrecision( 6 ) # точность

if layer.CreateField ( fieldDef ) != 0:
    print "Creating field failed.\n"
    sys.exit( 1 )
```

Чтобы записать объект, надо создать OGRFeature, установить атрибуты и присоединить геометрию. Объект OGRFeature должен быть связан со слоем:

```
name = "/home/alex/test.tif" # значение атрибута
feat = ogr.Feature( layer.GetLayerDefn() ) # создаем OGRFeature
feat.SetField( "FileName", name ) # устанавливаем атрибут

pt = ogr.Geometry( ogr.wkbPoint ) # создаем точку
pt.SetPoint_2D( 0, x, y )
feat.SetGeometry( pt ) # присоединяем геометрию к OGRFeature
```

И наконец, записываем все в файл

```
if layer.CreateFeature( feat ) != 0:
  print "Failed to create feature in shapefile.\n"
  sys.exit( 1 )

# освобождаем память
feat.Destroy()
ogrData.Destroy()
```

Пример. Выборка по условию и расчет площади

Вооружившись этими сведениями попробуем написать простенький скрипт на Python для выборки из

объектов полигонального шейпа по атрибуту, записи этих объектов в другой файл и вычисления площади полигонов. Скрипт принимает четыре параметра: исходный шейп, путь для сохранения результата, имя и значение атрибута по которому будет выполняться фильтрация.

```
# -*- coding: utf-8 -*-

#!/usr/bin/env python

import sys, os.path
import osgeo.ogr as ogr

def usage():
  print "Usage: query_shape.py input_shapefile output_shapefile query_field
match_value"
  sys.exit( 0 )

if __name__ == '__main__':
  args = sys.argv[ 1: ]

  if len( args ) != 4:
    usage()

  inPath = os.path.normpath( args[ 0 ] )
  outPath = os.path.normpath( args[ 1 ] )
  fieldName = args[ 2 ]
  fieldValue = args[ 3 ]

  # открываем исходный набор данных
  inputData = ogr.Open( inPath, False )
  if inputData is None:
    print "ERROR: open failed"
    sys.exit( 1 )

  # в шейп-файле есть только один слой,
  # делаем его рабочим и переходим на первую запись
  inputLayer = inputData.GetLayer( 0 )
  inputLayer.ResetReading()

  # устанавливаем фильтр по атрибутам
  # (для простоты берем равенство)
  query = fieldName + '=' + fieldValue
  inputLayer.SetAttributeFilter( query )

  # подготавливаем все для записи результата
  # выбираем драйвер
  driverName = "ESRI Shapefile"
  drv = ogr.GetDriverByName( driverName )
  if drv is None:
    print "%s driver not available." % driverName
    sys.exit( 1 )

  # создаем выходной набор данных
  outputData = drv.CreateDataSource( outPath )
  if outputData is None:
    print "Creation of output file failed."
    sys.exit( 1 )

  # и выходной слой
  layerName = os.path.basename( outPath ).strip( ".shp" )
  outputLayer = outputData.CreateLayer( layerName, None, ogr.wkbPolygon )
  if outputLayer is None:
    print "Layer creation failed."
    sys.exit( 1 )

  # начинаем просматривать объекты в исходном слое
```

```
inFeat = inputLayer.GetNextFeature()

# и сразу же формируем таблицу атрибутов результирущего слоя
inFeatDef = inputLayer.GetLayerDefn()
# делаем копию всех полей
for i in range( inFeatDef.GetFieldCount() ):
  fieldDef = inFeatDef.GetFieldDefn( i )
  if outputLayer.CreateField ( fieldDef ) != 0:
    print "Can't create field %s" % fieldDef.GetNameRef()
    sys.exit( 1 )
# добавлем поле для площади
fieldDef = ogr.FieldDefn( "area", ogr.OFTReal )
fieldDef.SetWidth( 12 )
fieldDef.SetPrecision( 4 )
if outputLayer.CreateField ( fieldDef ) != 0:
  print "Can't create field %s" % fieldDef.GetNameRef()
  sys.exit( 1 )

# собственно цикл в котором и перебираем объекты, удовлетворяющие
# условию фильтра
inFeatDef = inputLayer.GetLayerDefn()
while inFeat is not None:
  outFeat = ogr.Feature( outputLayer.GetLayerDefn() )
  # копируем атрибуты и геометрию исходного объекта
  res = outFeat.SetFrom( inFeat )
  if res != 0:
    print "Can't copy feature"
    sys.exit( 1 )
  # находим площадь полигона
  geom = inFeat.GetGeometryRef()
  area = geom.GetArea()
  # заполняем поле с геометрией
  outFeat.SetField( "area", area )

  # записываем новый объект в выходной слой
  if outputLayer.CreateFeature( outFeat ) != 0:
    print "Failed to create feature in shapefile.\n"
    sys.exit( 1 )

  # убираем мусор и переходим к следующему объекту
  outFeat.Destroy()
  inFeat = inputLayer.GetNextFeature()

# все объекты обработаны, закрываем наборы данных
inputData.Destroy()
outputData.Destroy()
```

Возьмем в качестве исходного слоя шейп-файл veg.shp из геосэмпла и попробуем с помощью этого скрипта получить новый файл с объектами у которых поле SUBTYPE_L равно 2. Строка запуска скрипта будет такой (перед запуском не забудьте выполнить gdal16):

```
python query_shape.py shape/veg.shp out.shp SUBTYPE_L 2
```

Ссылки по теме

1. Работа с растрами при помощи GDAL и Python
2. OGR API tutorial
3. OGR Architecture
4. GDAL/OGR in Python

Обсудить в форуме Комментариев — 57

Последнее обновление: 2014-05-15 00:14

Дата создания: 15.04.2010

Автор(ы): Александр Бруй