

3.3 二进制数据读写

Markdown by 张涛涛

二进制数据可以快速地与机器交互、节省空间。广义地讲，NetCDF、Grib、HDF数据均属于二进制数据。

1) 普通二进制数据

普通二进制文件直接将数据存储为二进制，读写该数据须对数据进行编解码，可用Python内置struct模块的pack()和unpack()函数

将wave.nc写成普通二进制格式文件wave.grd

注意：有的Xarray不支持中文路径，包括所在文件夹为中文

```
In [1]: import xarray as xr
from struct import pack
if __name__ == '__main__':
    da = xr.open_dataset('wave.nc')
    dat = da['wave']
    dat = dat.to_numpy()
    f = open('wave.grd', 'wb')
    for x in dat:
        for y in x:
            for z in y:
                z = pack('f', z)
                f.write(z)
    f.close()
```

将写的数据读出来

```
In [2]: from struct import unpack
import numpy as np
if __name__ == '__main__':
    with open('wave.grd', 'rb') as f:
        dat = f.read()
        dat = unpack(str(12*181*360)+'f', dat)
        dat = np.reshape(dat, (12, 181, 360))
        print(dat.shape)
```

```
(12, 181, 360)
```

pack()和unpack()函数中的数据格式的设置取决于具体需求，一般使用little-endian的（表示二进制的位是正序的）；在个别模式中数据会存储为big-endian的格式（二进制的位是反序的），如：unpack('>f', x)；除浮点型外，二进制还可以存储整型或64位的浮点型等。若对二进制数据进行再编码，就可以存成netCDF、HDF、Grib等数据类型。

2) NetCDF和HDF

NetCDF是支持多平台创建、读写和分享的数组形式的数据，气象数据常存储成netCDF格式，比如各种再分析资料。

NetCDF和HDF可用xarray.open_dataset()打开，也可用netCDF4和h5py

利用netCDF4库读取nc文件

```
In [3]: from netCDF4 import Dataset, num2date
        if __name__ == '__main__':
            f = Dataset('wave.nc')
            print(f)
```

```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF4 data model, file format HDF5):
  dimensions(sizes): time(12), lat(181), lon(360)
  variables(dimensions): int64 time(time), int64 lat(lat), int64 lon(lon), float64 wave(time, lat, lon)
  groups:
```

```
In [4]: dat = f.variables['wave']
        print(dat)
```

```
<class 'netCDF4._netCDF4.Variable'>
float64 wave(time, lat, lon)
  _FillValue: nan
  units: unitless
  creator: Zhou Y.
  description: A sin wave
  unlimited dimensions:
  current shape = (12, 181, 360)
  filling on
```

```
In [5]: dat = dat[:]
        print(dat.shape) #print(dat)
```

```
(12, 181, 360)
```

netCDF4库中的num2date函数可将整数序列转换成时间

```
In [6]: time = f.variables['time']
        print(time[:])
        time = num2date(time[:], time.units, calendar='standard')
        print(time)
```

```
[ 0  31  59  90 120 151 181 212 243 273 304 334]
[cftime.DatetimeGregorian(2100, 1, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 2, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 3, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 4, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 5, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 6, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 7, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 8, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 9, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 10, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 11, 1, 0, 0, 0, 0, has_year_zero=False)
 cftime.DatetimeGregorian(2100, 12, 1, 0, 0, 0, 0, has_year_zero=False)]
```

将netCDF格式数据写成HDF格式

```
In [7]: from netCDF4 import Dataset
        import h5py
        if __name__ == '__main__':
            f = Dataset('wave.nc')
            dat = f.variables['wave']
            time = f.variables['time']
            lon = f.variables['lon'][:]
```

```

lat = f.variables['lat'][:]
with h5py.File('wave.hdf5', 'w') as fh:
    fh['wave'] = dat
    fh['time'] = time[:]
    fh['lat'] = lat
    fh['lon'] = lon
    fh['time'].attrs['units'] = time.units
    fh['time'].make_scale('time name')
    fh['lon'].make_scale('lon name')
    fh['lat'].make_scale('lat name')

```

利用h5py库读取HDF5数据

```

In [8]: import h5py
if __name__ == '__main__':
    with h5py.File('wave.hdf5') as f:
        print(f)
        dat = f['wave']
        print(dat)
        dat = dat[:]
        # print(dat)

```

<HDF5 file "wave.hdf5" (mode r)>

<HDF5 dataset "wave": shape (12, 181, 360), type "<f8">

3) Grib数据

Grib是WMO用于存储和交换格点数据的一种数据格式，目前有Grib1和Grib2两个版本，一般涉及到读，很少进行写的操作。在Python中可以用ecCodes、pygrib、xarray+cfgrib等模块进行读的操作。

创建一个grib数据

```

In [9]: import numpy as np
from eccodes import *
def crdat():
    lon = np.arange(0, 360, 2.5)
    lat = np.arange(90, -91, -2.5)
    x, y = np.meshgrid(lon, lat)
    m, n = (1, 2)
    A = 100 * np.sin(n * y / 180 * np.pi) ** 2
    dat = []
    for i in range(12):
        tmp = A * np.sin(m * x / 180 * np.pi - i * np.pi / 12)
        dat.append(tmp)
    time = [''.join(['2100', '%02i' % (i+1), '01']) for i in range(12)]
    time = [int(x) for x in time]
    return time, lat, lon, dat
if __name__ == '__main__':
    time, lat, lon, dat = crdat()
    sample_id = codes_grib_new_from_samples("regular_ll_pl_grib2")
    keys = {'dataDate': time[0],
            'startStep': 0,
            'endStep': 0,
            'hour': 0,
            'gridType': 'regular_ll',
            'stepType': 'instant',
            'tablesVersion': 28,

```

```

        'discipline':0,
        'parameterCategory':3,
        'parameterNumber':5,
        'decimalPrecision':3,
        'level':500,
        'Nx':len(lon),
        'Ny':len(lat),
        'latitudeOfFirstGridPointInDegrees':lat[0],
        'longitudeOfFirstGridPointInDegrees':lon[0],
        'latitudeOfLastGridPointInDegrees':lat[-1],
        'longitudeOfLastGridPointInDegrees':lon[-1],
        'jDirectionIncrementInDegrees':2.5,
        'iDirectionIncrementInDegrees':2.5,
        'centre':'babj',
    }
    clone_id = codes_clone(sample_id)
    with open('wave.grib2','wb') as f:
        for t,x in zip(time,dat):
            keys['dataDate'] = t
            for key in keys:
                codes_set(clone_id,key,keys[key])
            codes_set_values(clone_id,x.flatten())
            codes_write(clone_id,f)

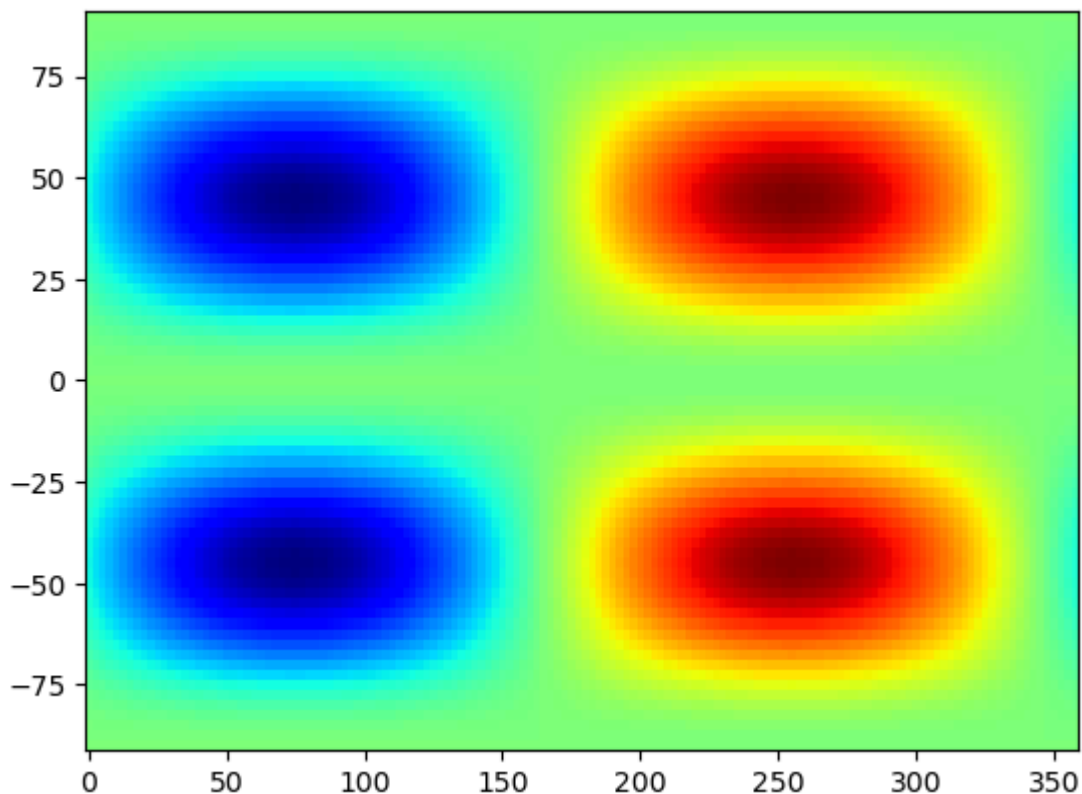
```

读取grib数据：方法一：使用pygrib

```

In [10]: import pygrib
if __name__=='__main__':
    with pygrib.open('wave.grib2') as grbs:
        for x in grbs:
            #print(x.keys())
            dat = x['values']
            lon = x['longitudes'].reshape(73,144)
            lat = x['latitudes'].reshape(73,144)
    import matplotlib.pyplot as plt
    plt.pcolormesh(lon,lat,dat,cmap='jet')
    plt.show()

```



读取grib数据：方法二：使用Xarray+cfgrib,注意设置engine='cfgrib'

```
In [11]: import xarray as xr
if __name__ == '__main__':
    da = xr.open_dataset('wave.grib2', engine='cfgrib')
    # print(da)
```

Ignoring index file 'wave.grib2.5b7b6.idx' older than GRIB file

使用xarray读取hdf, grib文件获得的是DataArray结构的数据，此时可使用DataArray.to_netcdf()把数据存储为nc文件

```
In [12]: da.to_netcdf('wave.grib2.nc')
```

4) Pickle模块

虽然鲜有气象数据存储为Pickle的格式，但Pickle是使用Python时重要的存储方式，它可以存储Python的任意对象。

创建一个Pickle数据，将上面的数据存储为字典

```
In [16]: import pickle as pk
with open('wave.pkl', 'wb') as f:
    pk.dump({'time': da['time'].values,
            'lat': da['latitude'].values,
            'lon': da['longitude'].values,
            'gh': da['gh'].values}, f)
```

读取Pickle数据

```
In [19]: with open('wave.pkl', 'rb') as f:
    dat = pk.load(f)
```

```
print(type(dat), dat.keys())
```

```
<class 'dict'> dict_keys(['time', 'lat', 'lon', 'gh'])
```

3.4 本章小结

数据读写是气象应用的基本功，掌握不同格式的读写，举一反三。
不要被数据的后缀欺骗。