cnmaps

发布 *1.0.1*

Clarmy

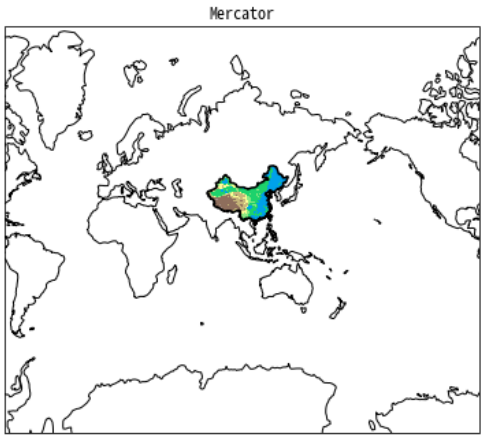
2022 年 04 月 28 日

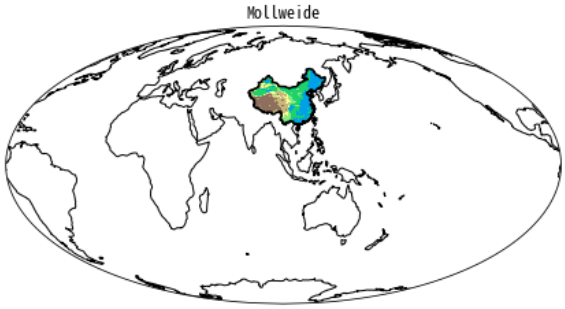
Contents

cnmaps 是一个致力于让中国地图的获取和使用更丝滑的 python 扩展包。

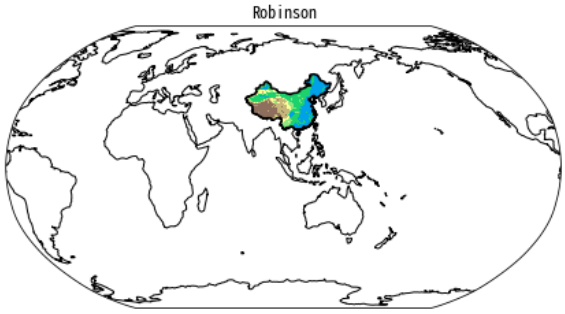
当前版本具有以下几个主要功能：

1. 自带合规地图边界，数据源来自于高德等测绘机构，让你无需再额外寻找地图边界文件。
2. 支持地图边界之间的加减、交并集等常规操作，让你可以自由地组合想要的地图形状。
3. 具有易于使用的地图裁剪功能，且裁剪效果好，平滑无锯齿。
4. 与 cartopy 集成，可以自动转换地图边界的投影。


Mercator


Mollweide


Orthographic


Robinson

CHAPTER 1

cnmaps 使用指南

1.1 安装

cnmaps 依赖于 `cartopy>=0.19.0`，因此在安装 cnmaps 之前请确保 cartopy 已安装，cartopy 的安装方法 。

在完成 cartopy 的安装以后，你可以使用 pip 来安装 cnmaps：`$ pip install cnmaps==1.0.1`

> **警告：** 由于 cnmaps 目前为探索和实验阶段，有些功能可能会随着版本的更新而发生巨大的变化，为了避免由于版本更新而导致代码不可用，请在安装时指定版本号。本文档是以 `cnmaps==1.0.1` 版本进行说明的，其他版本可能并不适用。

1.2 快速开始

我们先用几个小例子，快速入门 cnmaps 的基本功能的使用。

1.2.1 查询行政边界

你可以使用 `get_adm_maps` 轻松查询到你想要的行政边界，例如你想要查询北京市，可以使用以下方式。

```
In [1]: from cnmaps import get_adm_maps

In [2]: get_adm_maps(city='北京市')
Out[2]:
[{'国家': '中华人民共和国',
'省/直辖市': '北京市',
'市': '北京市',
'区/县': None,
'级别': '市',
'来源': '高德',
'类型': '陆地',
'geometry': <cnmaps.maps.MapPolygon at 0x7f861690c050>}]
```

查询海淀区。

```
In [1]: from cnmaps import get_adm_maps

In [2]: get_adm_maps(district='海淀区')
Out[2]:
[{'国家': '中华人民共和国',
'省/直辖市': '北京市',
'市': '北京市',
'区/县': '海淀区',
'级别': '区县',
'来源': '高德',
'类型': '陆地',
'geometry': <cnmaps.maps.MapPolygon at 0x7f861af85fd0>}]
```

查询山西省。

```
In [1]: from cnmaps import get_adm_maps

In [2]: get_adm_maps(province='山西省')
Out[2]:
[{'国家': '中华人民共和国',
'省/直辖市': '山西省',
```

(下页继续)

(续上页)

```
'市': None,
'区/县': None,
'级别': '省',
'来源': '高德',
'类型': '陆地',
'geometry': <cnmaps.maps.MapPolygon at 0x7f8618f86790>}]
```

查询山西省下辖地级市。

```
In [1]: from cnmaps import get_adm_maps

In [2]: get_adm_maps(province='山西省', level='市')
Out[2]:
[{'国家': '中华人民共和国',
'省/直辖市': '山西省',
'市': '太原市',
'区/县': None,
'级别': '市',
'来源': '高德',
'类型': '陆地',
'geometry': <cnmaps.maps.MapPolygon at 0x7f863fd618d0>},
... # 为节省篇幅，中间部分省略
'省/直辖市': '山西省',
'市': '吕梁市',
'区/县': None,
'级别': '市',
'来源': '高德',
'类型': '陆地',
'geometry': <cnmaps.maps.MapPolygon at 0x7f863fd613d0>}]

In [3]: get_adm_maps(province='山西省', level='市', engine='geopandas')
Out[3]:
        国家  省/直辖市     市    区/县  级别   来源   类型                                ␣
↪           geometry
0   中华人民共和国   山西省   太原市  None   市   高德   陆地   MULTIPOLYGON (((113.06683␣
↪38.05646, 113.06708 ...
1   中华人民共和国   山西省   大同市  None   市   高德   陆地   MULTIPOLYGON (((113.57727␣
↪39.43812, 113.57460 ...
2   中华人民共和国   山西省   阳泉市  None   市   高德   陆地   MULTIPOLYGON (((113.99691␣
↪37.70448, 113.99567 ...
3   中华人民共和国   山西省   长治市  None   市   高德   陆地   MULTIPOLYGON (((111.99642␣
↪36.68713, 111.99480 ...
4   中华人民共和国   山西省   晋城市  None   市   高德   陆地   MULTIPOLYGON (((113.46543␣
↪35.51493, 113.46300 ...
```

(下页继续)

(续上页)

```
5   中华人民共和国   山西省   朔州市   None   市   高德   陆地   MULTIPOLYGON (((112.62431␣
↪40.23685, 112.62429 ...
6   中华人民共和国   山西省   晋中市   None   市   高德   陆地   MULTIPOLYGON (((113.06683␣
↪38.05646, 113.06903 ...
7   中华人民共和国   山西省   运城市   None   市   高德   陆地   MULTIPOLYGON (((110.90373␣
↪34.66882, 110.89349 ...
8   中华人民共和国   山西省   忻州市   None   市   高德   陆地   MULTIPOLYGON (((111.26944␣
↪39.42373, 111.27091 ...
9   中华人民共和国   山西省   临汾市   None   市   高德   陆地   MULTIPOLYGON (((110.41054␣
↪36.89947, 110.41487 ...
10  中华人民共和国   山西省   吕梁市   None   市   高德   陆地   MULTIPOLYGON (((111.41469␣
↪36.80403, 111.41071 ...
```

备注： 当你向 get_adm_maps 传递行政区域的名称时，应传入行政区的正式全称，简称无法识别，如果不知道全称可以通过 get_adm_names 查询。

假如我们不知道省一级行政区的正式名称，可以执行：

```
In [1]: from cnmaps import get_adm_names

In [2]: get_adm_names(level='省')
Out[2]:
['北京市',
'天津市',
'河北省',
... # 为节省篇幅，中间部分省略
'台湾省',
'香港特别行政区',
'澳门特别行政区']
```

当我们已经知道了省的名称以后，可以继续下探到市，以四川省为例：

```
In [1]: from cnmaps import get_adm_names

In [2]: get_adm_names(province='四川省', level='市')
Out[2]:
['成都市',
'自贡市',
'攀枝花市',
... # 为节省篇幅，中间部分省略
'阿坝藏族羌族自治州',
'甘孜藏族自治州',
```

(下页继续)

(续上页)

```
'凉山彝族自治州']
```

知道了市的名称以后，可以继续下探到区县，以成都市为例：

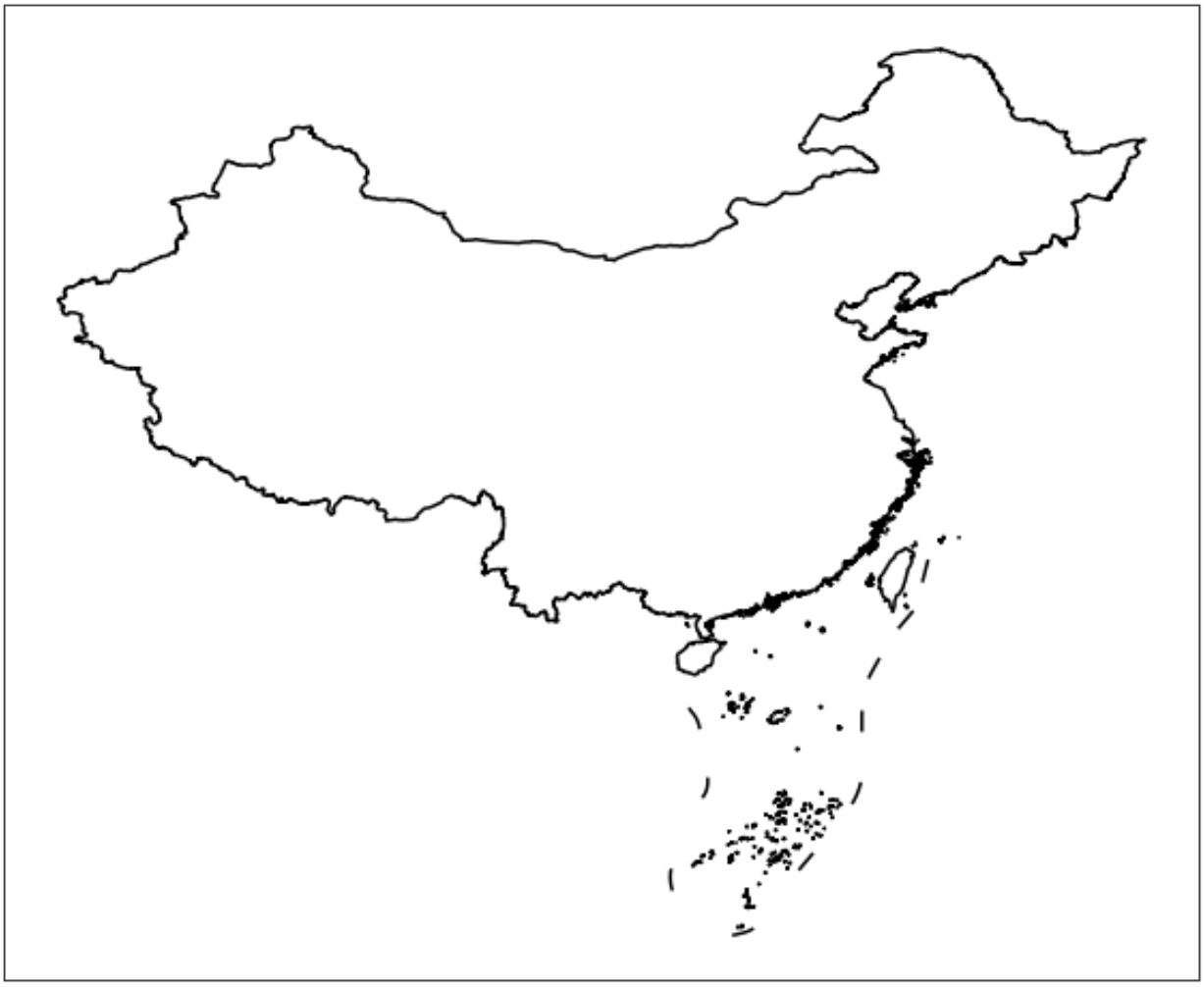
```
In [1]: from cnmaps import get_adm_names

In [2]: get_adm_names(province='四川省', city='成都市', level='区县')
Out[2]:
['锦江区',
'青羊区',
'金牛区',
... # 为节省篇幅，中间部分省略
'邛崃市',
'崇州市',
'简阳市']
```

1.2.2 绘制行政边界

前面使用 get_adm_maps 获取的行政边界地图列表，可以直接传入 draw_maps 函数进行绘图。

我们先来用最简单直接的方式，来绘制你的第一张中国国界地图。

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_maps

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())

draw_maps(get_adm_maps(level='国'))

plt.show()
```

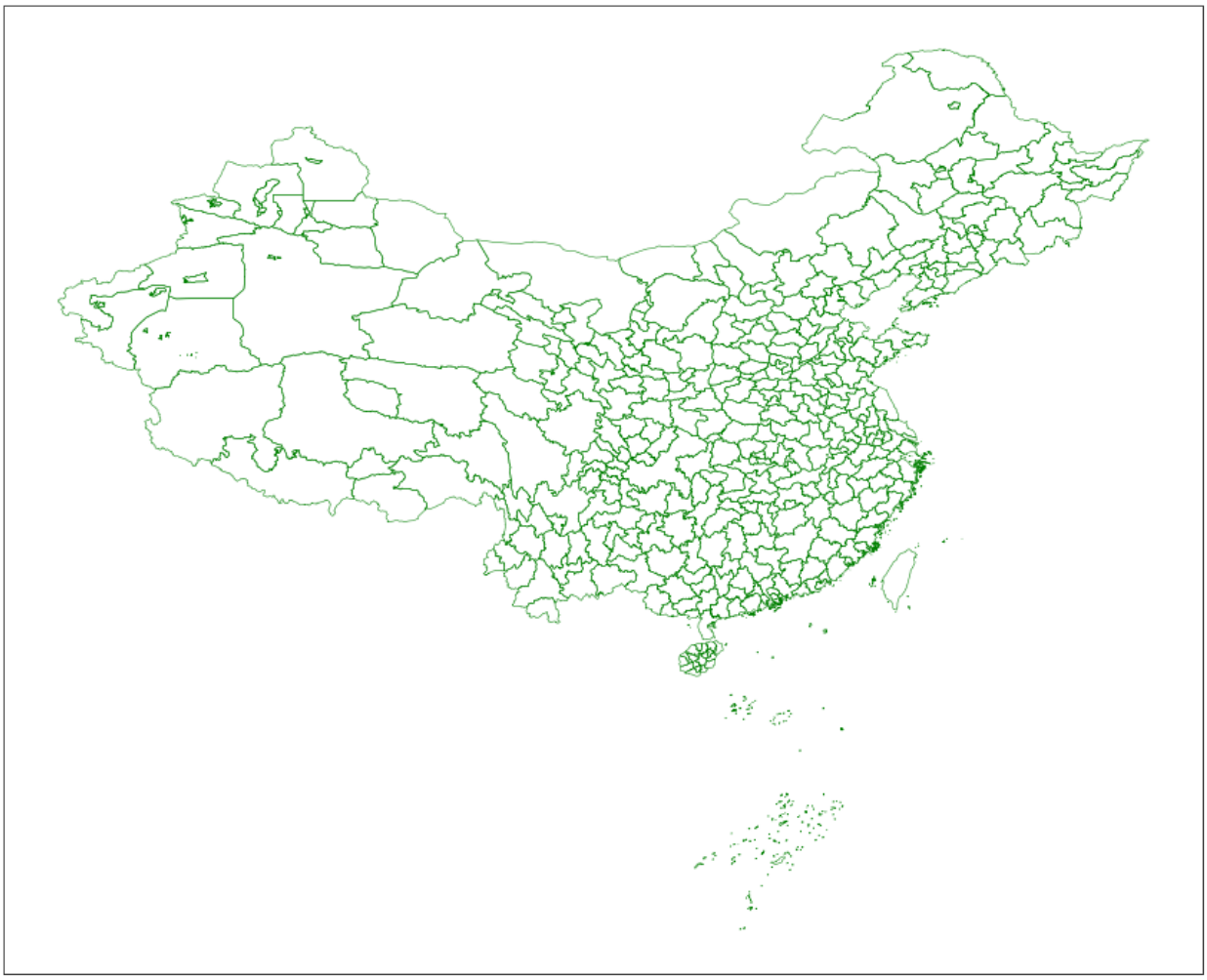


我们再来绘制一张各省的行政边界地图。

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_maps

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())

draw_maps(get_adm_maps(level='省'), linewidth=0.8, color='r')

plt.show()
```

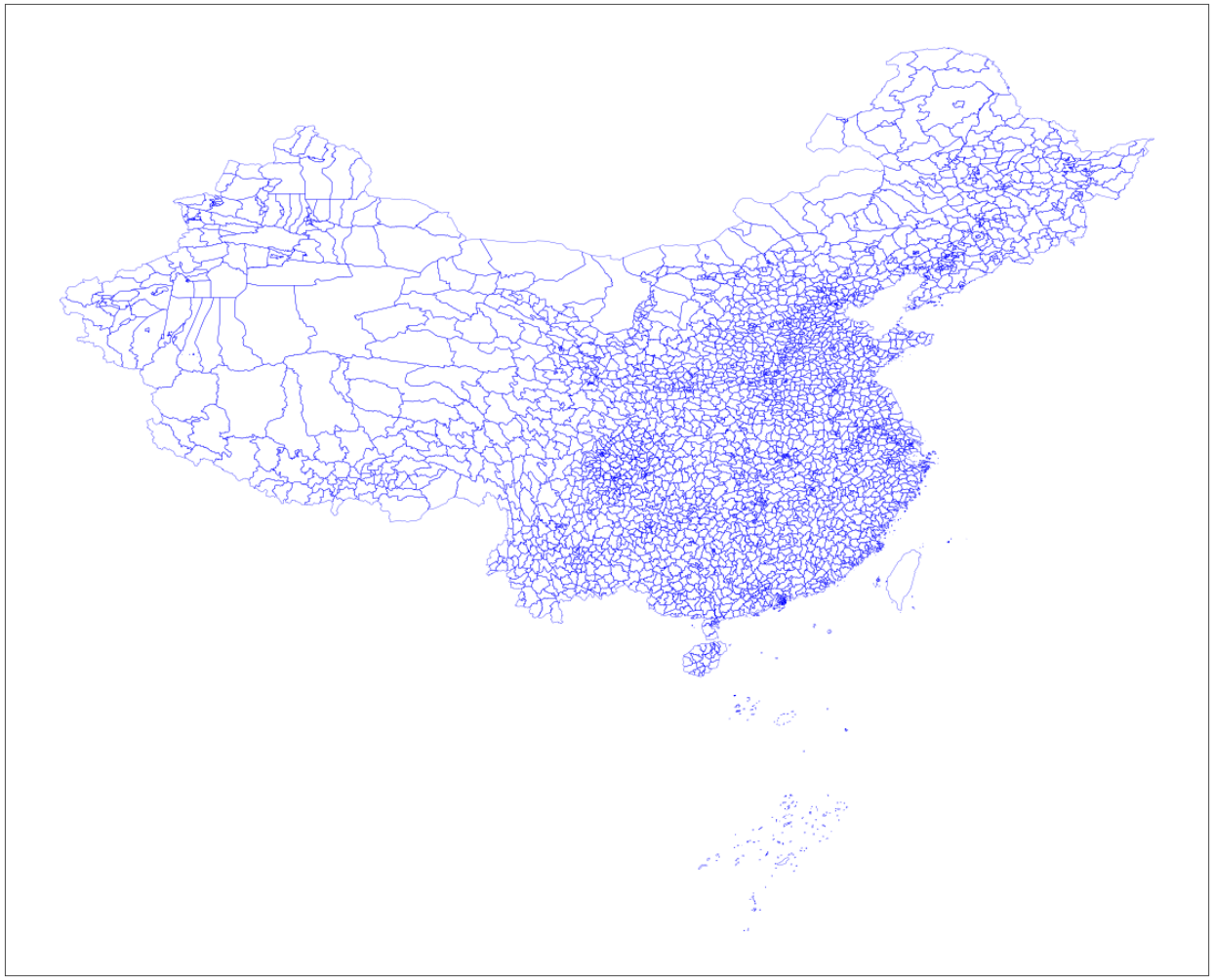
然后是市级行政区。

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_maps

fig = plt.figure(figsize=(15,15))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())

draw_maps(get_adm_maps(level='市'), linewidth=0.5, color='g')

plt.show()
```



最后是区县。

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_maps

fig = plt.figure(figsize=(20,20))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())

draw_maps(get_adm_maps(level='区县'), linewidth=0.3, color='b')

plt.show()
```



1.2.3 合并边界

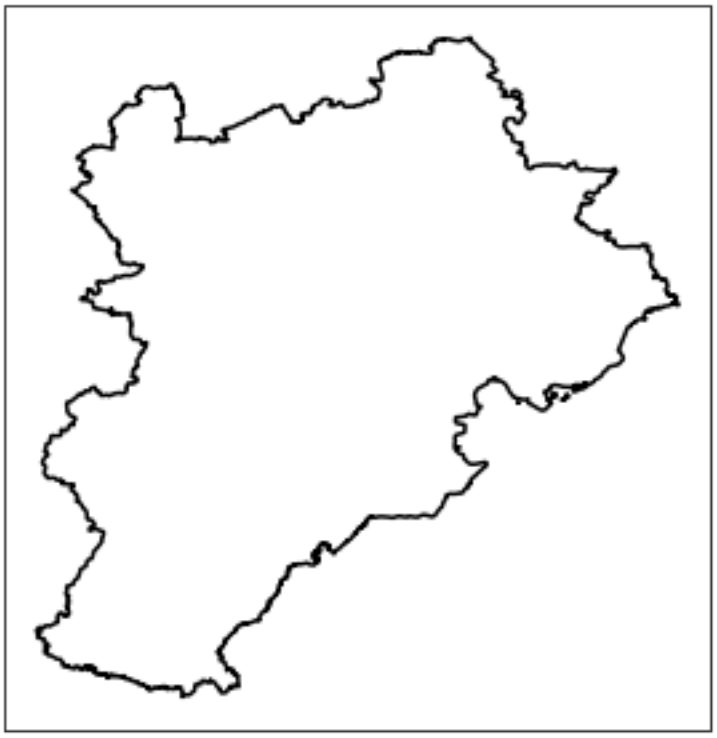
cnmaps 可以很方便地对地图进行合并，例如我们可以将北京、天津、河北的边界对象直接相加获得京津冀的边界对象并绘图。

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_map

beijing = get_adm_maps(province='北京市', only_polygon=True, record='first')
tianjin = get_adm_maps(province='天津市', only_polygon=True, record='first')
hebei = get_adm_maps(province='河北省', only_polygon=True, record='first')

jingjinji = beijing + tianjin + hebei

fig = plt.figure(figsize=(5,5))
```

(下页继续)

(续上页)

```
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
draw_map(jingjinji)

plt.show()
```



1.2.4 剪切地图

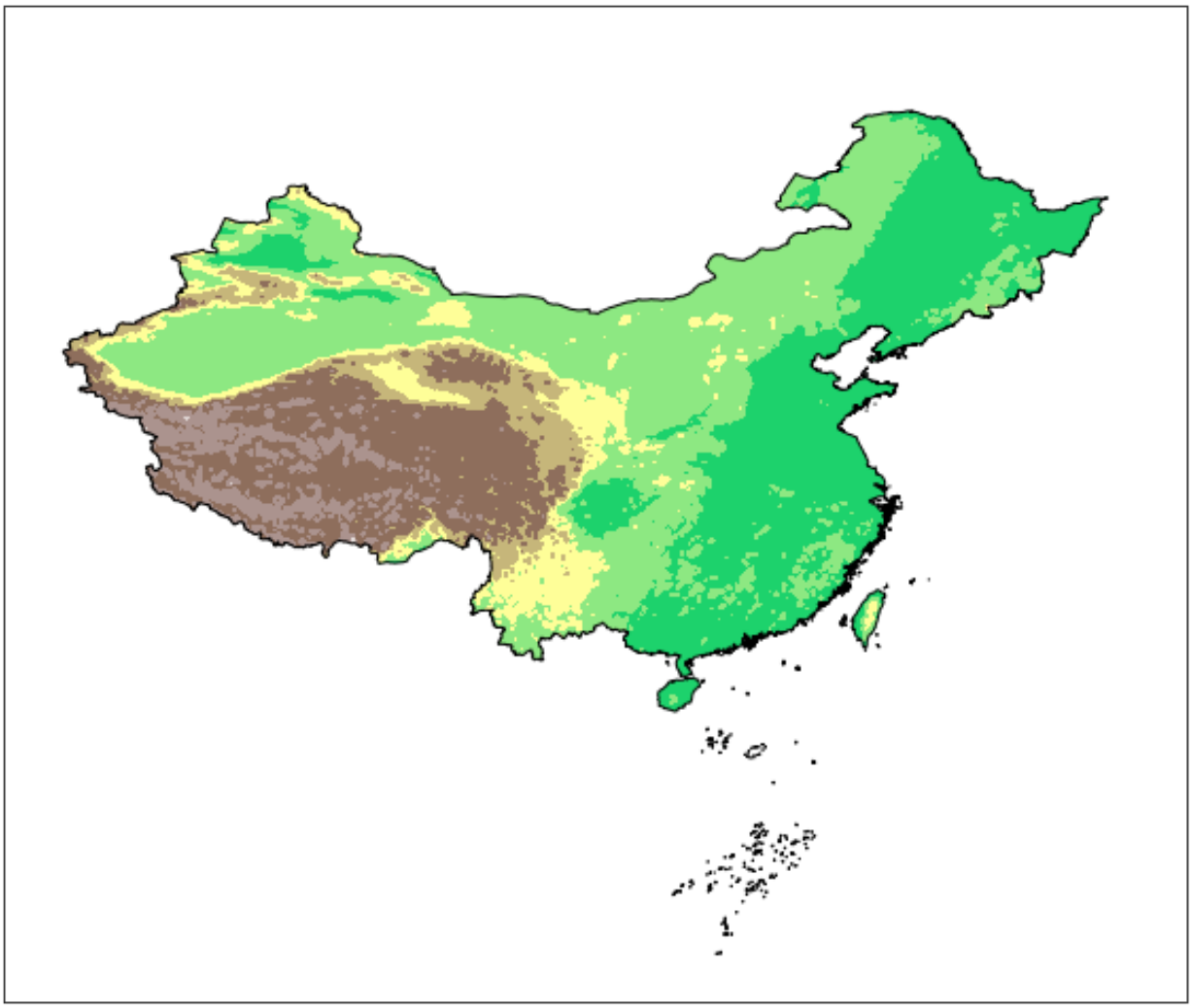
剪切填色等值线 (contourf) 图

```
from cnmaps import get_adm_maps, clip_contours_by_map, draw_map
from cnmaps.sample import load_dem

lons, lats, data = load_dem()

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
map_polygon = get_adm_maps(country='中华人民共和国', record='first', only_
↪polygon=True)

cs = ax.contourf(lons, lats, data,
                 cmap=plt.cm.terrain,
                 levels=np.linspace(-2800, data.max(), 10),
```

(下页继续)

(续上页)

```
                transform=ccrs.PlateCarree())

clip_contours_by_map(cs, map_polygon)
draw_map(map_polygon, color='k', linewidth=1)

plt.show()
```



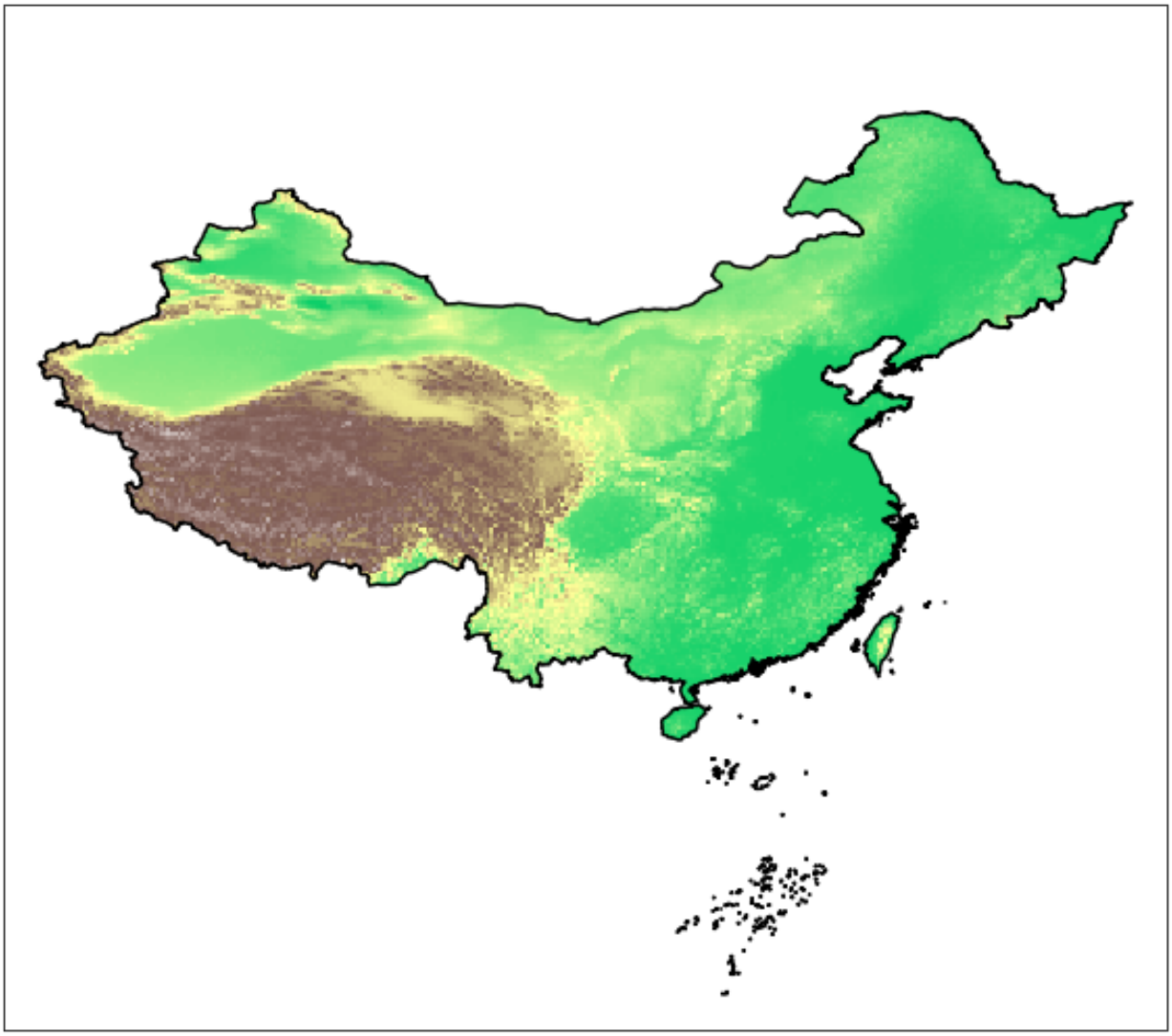
剪切填色 (pcolormesh) 图

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_map, clip_pcolormesh_by_map
from cnmaps.sample import load_dem

lons, lats, dem = load_dem()
```

(下页继续)

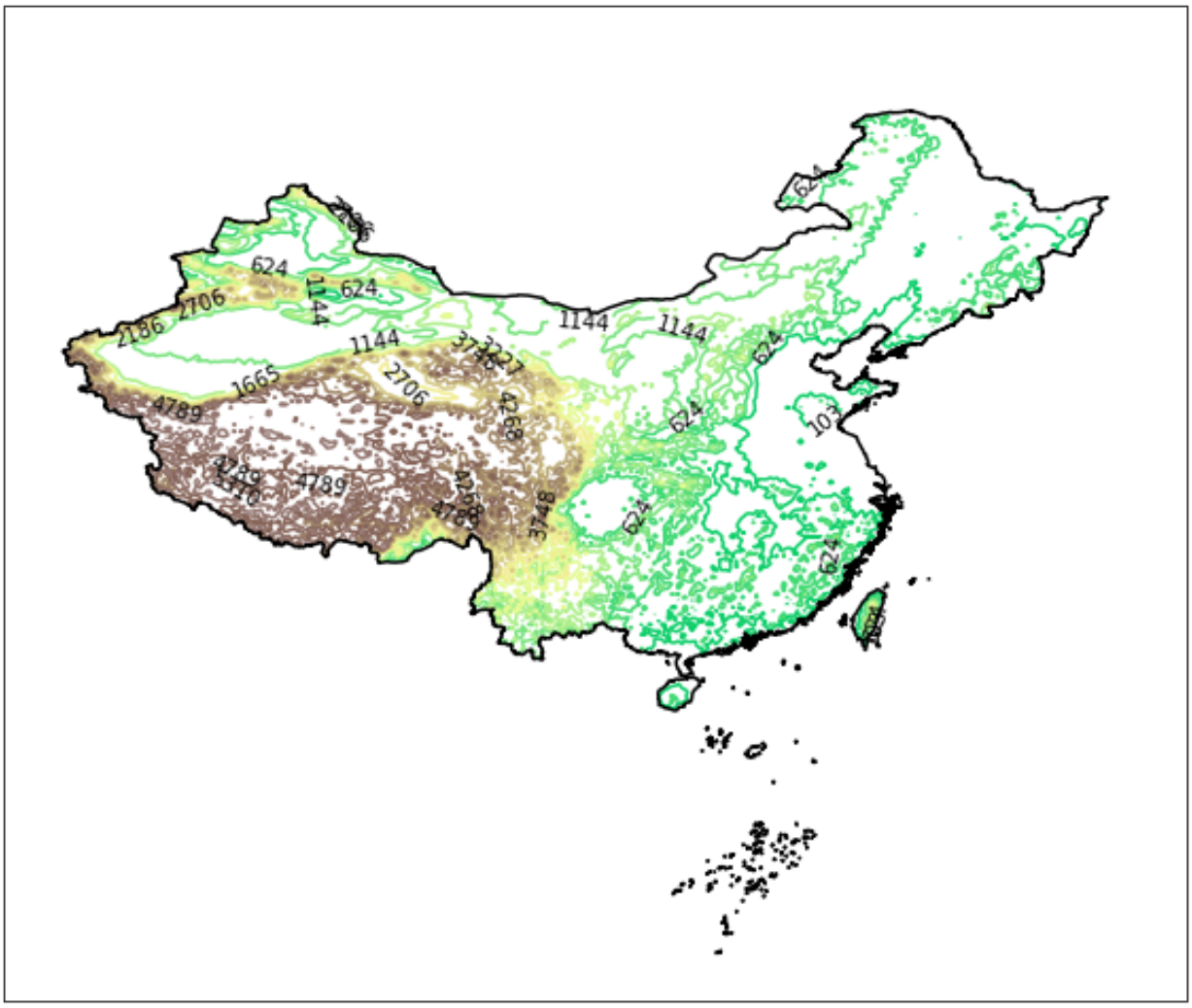
(续上页)

```
fig = plt.figure(figsize=(10, 10))

map_polygon = get_adm_maps(country='中华人民共和国', record='first', only_
↪polygon=True)

ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
mesh = ax.pcolormesh(lons, lats, dem, cmap=plt.cm.terrain, vmin=-2800, transform=ccrs.
↪PlateCarree())
clip_pcolormesh_by_map(mesh, map_polygon)
draw_map(map_polygon, color='k')
ax.set_extent(map_polygon.get_extent())

plt.show()
```



剪切等值线 clabel

```
from cnmaps import get_adm_maps, clip_clabels_by_map, clip_contours_by_map, draw_map
from cnmaps.sample import load_dem

lons, lats, data = load_dem()

map_polygon = get_adm_maps(
    country='中华人民共和国', record='first', only_polygon=True)
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())
contours = ax.contour(lons,
                      lats,
                      data,
                      cmap=plt.cm.terrain,
                      levels=np.linspace(-2500, data.max(), 20),
                      transform=ccrs.PlateCarree())
clip_contours_by_map(contours, map_polygon)
clabels = ax.clabel(contours,
                      levels=contours.levels,
                      colors='k',
                      fmt='%i',
                      inline=True)
clip_clabels_by_map(clabels, map_polygon)
draw_map(map_polygon, color='k')
```



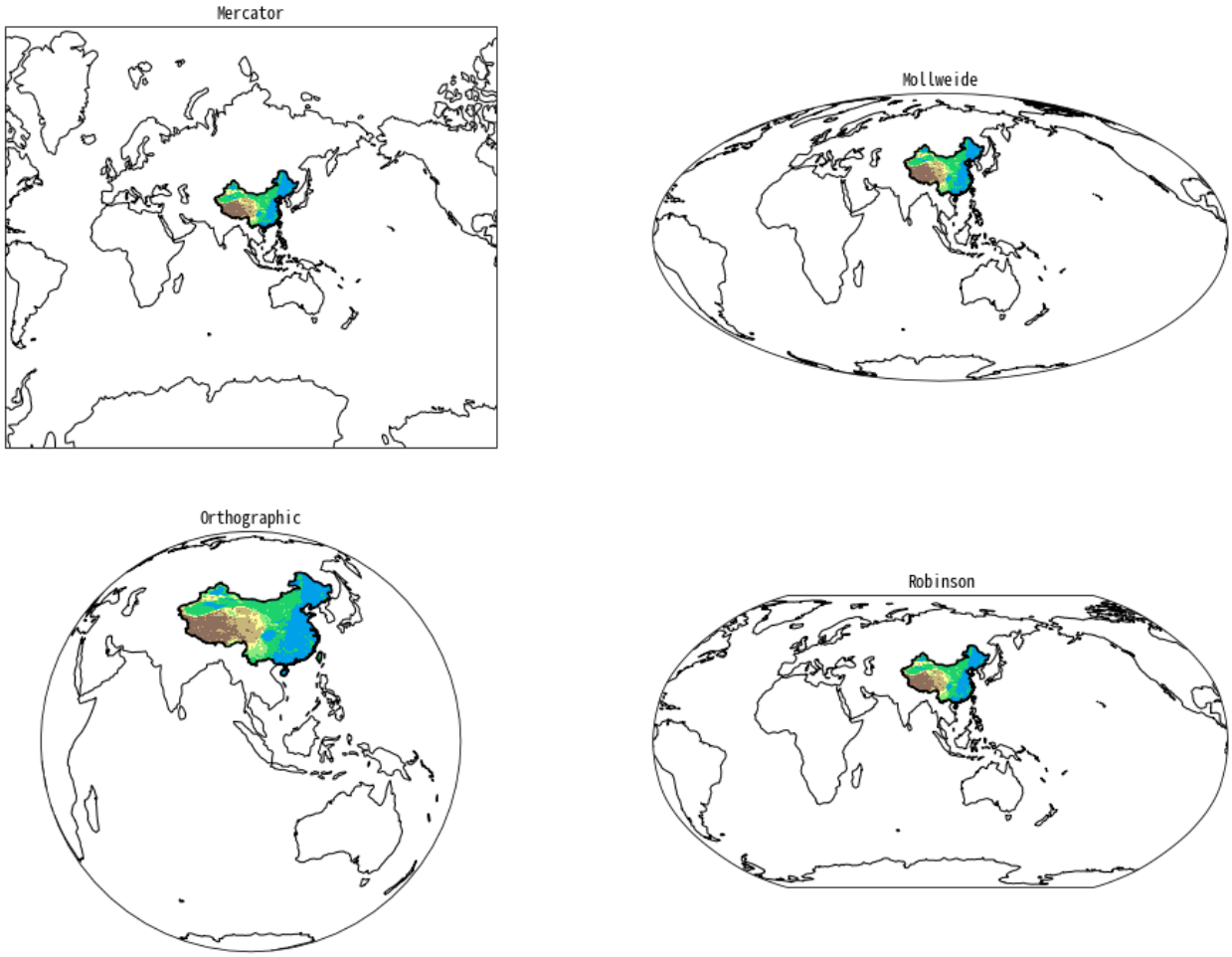
1.2.5 多投影支持

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_map, clip_contours_by_map
from cnmaps.sample import load_dem

lons, lats, dem = load_dem()

PROJECTIONS = [
    ('Mercator', ccrs.Mercator(central_longitude=100)),
    ('Mollweide', ccrs.Mollweide(central_longitude=100)),
    ('Orthographic', ccrs.Orthographic(central_longitude=100)),
    ('Robinson', ccrs.Robinson(central_longitude=100))
]
```

(下页继续)

(续上页)

```
fig = plt.figure(figsize=(16, 12))
fig.tight_layout()

china = get_adm_maps(country='中华人民共和国', record='first', only_polygon=True)

for i, prj in enumerate(PROJECTIONS):
    ax = fig.add_subplot(2,2,i+1, projection=prj[1])
    cs = ax.contourf(lons, lats, dem, cmap=plt.cm.terrain, transform=ccrs.
↪PlateCarree())
    clip_contours_by_map(cs, china)

    draw_map(china, color='k')
    ax.set_extent(china.get_extent(buffer=3))
    ax.set_global()
    ax.coastlines()
    plt.title(prj[0])

plt.show()
```



1.3 综合示例

1.3.1 绘制项目 LOGO 底图

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps

fig = plt.figure(figsize=(5,5))
proj = ccrs.Orthographic(central_longitude=100.0, central_latitude=30)
ax = fig.add_subplot(111, projection=proj)

ax.stock_img()
china, sourth_sea = get_adm_maps(level='国', only_polygon=True)

ax.set_global()
```

(下页继续)

(续上页)

```
ax.add_geometries(china, crs=ccrs.PlateCarree(), edgecolor='r', facecolor='r')
ax.add_geometries(sourth_sea, crs=ccrs.PlateCarree(), edgecolor='r')
ax.spines['geo'].set_edgecolor('white')

plt.show()
```

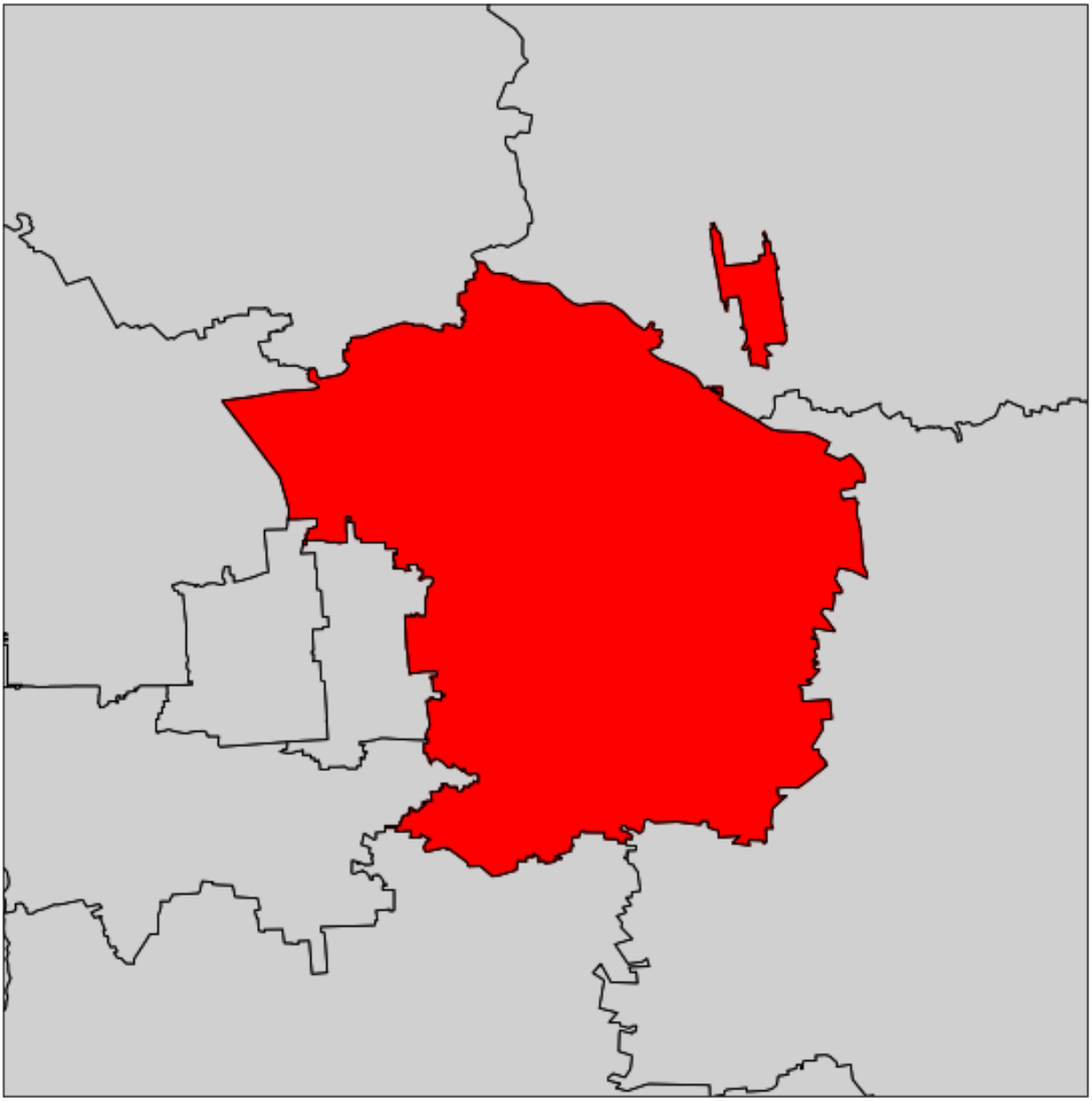
1.3.2 绘制北京市朝阳区行政图

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_maps

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())

beijing = get_adm_maps(city='北京市', record='first', only_polygon=True)
chaoyang = get_adm_maps(district='朝阳区', level='区县', record='first', only_
↪polygon=True)

ax.add_geometries(beijing, crs=ccrs.PlateCarree(), edgecolor='#D0D0D0', facecolor='
↪#D0D0D0')
```

(下页继续)

(续上页)

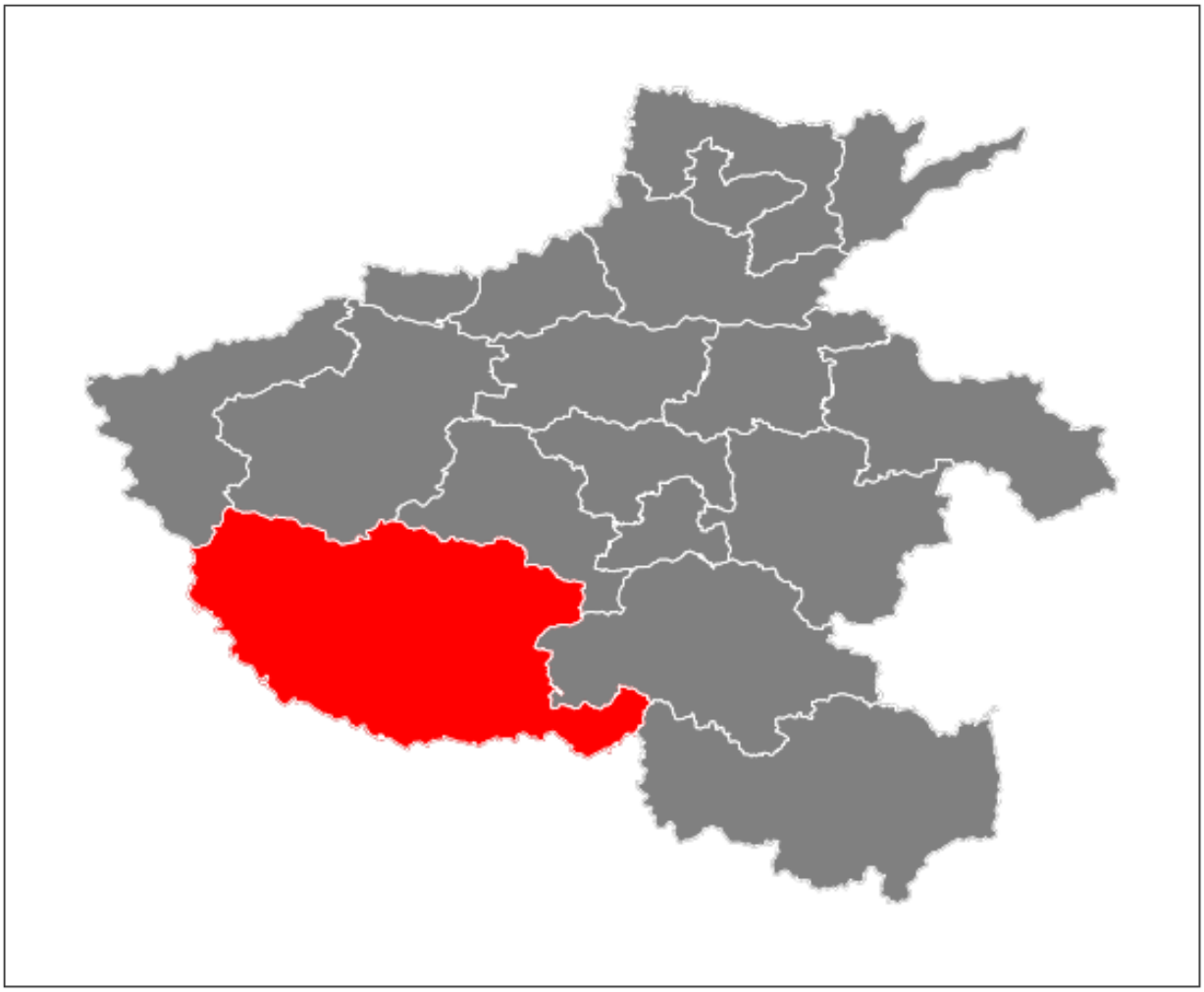
```
draw_maps(get_adm_maps(city='北京市', level='区县'), color='k', linewidth=0.8)
ax.add_geometries(chaoyang, crs=ccrs.PlateCarree(), edgecolor='r', facecolor='r')
ax.set_extent(chaoyang.get_extent(buffer=0.1))

plt.show()
```



1.3.3 绘制河南省行政图

河南省行政图，南阳市高亮

```
import cartopy.crs as ccrs
import matplotlib.pyplot as plt
from cnmaps import get_adm_maps, draw_maps

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection=ccrs.PlateCarree())

province = get_adm_maps(province='河南省', record='first', only_polygon=True)
city = get_adm_maps(city='南阳市', record='first', only_polygon=True)
ax.add_geometries(province, crs=ccrs.PlateCarree(), edgecolor='grey', facecolor='grey
↪')
ax.add_geometries(city, crs=ccrs.PlateCarree(), edgecolor='r', facecolor='r')
draw_maps(get_adm_maps(province='河南省', level='市'), color='w', linewidth=0.8)
ax.set_extent(province.get_extent(buffer=0.5))

plt.show()
```



1.4 API 参考

1.4.1 maps

maps 模块主要存放与地图边界对象相关的类和函数。

class cnmaps.maps.**MapPolygon**(*shapely.geometry.MultiPolygon*)

地图多边形类

该是基于 shapely.geometry.MultiPolygon 的自定义类, 并实现了对于加号操作符的支持.

drop_inner_duplicate(*map_polygon*)

地图对象的自我纠正，剔除内含的多余多边形，常见于多个地图多边形对象合并时

参数 map_polygon (cnmaps.maps.MapPolygon) – 地图边界对象，可以通过 get_adm_maps() 获取

返回 经过纠正后的 MapPolygon 对象

返回类型 *cnmaps.maps.MapPolygon*

get_extent(*buffer=2*)

获取范围坐标

参数 **buffer**(*float or int*) –外扩缓冲边缘, 单位为 °, 该值越大, 所取的范围越大. 默认为 2.

返回 坐标范围点, 该值可直接传入 `ax.set_extent()` 使用

返回类型 tuple

cnmaps.maps.**get_adm_names**(*province: str = None*, *city: str = None*, *district: str = None*, *level: str = '省'*, *country: str = '中华人民共和国'*, *source: str = '高德'*)

获取行政名称

参数

- **province**(*str*) –省/自治区/直辖市/行政特区中文名, 必须为全称, 例如查找河北省应收入 '河北省' 而非 '河北'. 默认为 None.
- **city**(*str*) –地级市中文名, 必须为全称, 例如查找北京市应输入 '北京市' 而非 '北京'. 默认为 None.
- **district**(*str*) –区/县中文名, 必须为全称. 默认为 None.
- **level**(*str*) –边界等级, 目前支持的等级包括 '省', '市', '区县'. 其中 '省' 级包括直辖市、特区等; '市' 级为地级市, 若为直辖市, 则名称与 '省' 级相同, 比如北京市的省级和市级都是 '北京市'; '区' 和 '县' 属于同一级别的不同表达形式. 默认为 '省'.
- **country**(*str*) –国家名称, 必须为全称. 默认为 '中华人民共和国'.
- **source**(*str*) –数据源. 默认为 '高德'.

返回 满足条件的名称列表

返回类型 list

cnmaps.maps.**get_adm_maps**(*province: str = None*, *city: str = None*, *district: str = None*, *level: str = '省'*, *country: str = '中华人民共和国'*, *source: str = '高德'*, *db: str = DB_FILE*, *engine: str = None*, *record: str = 'all'*, *only_polygon: bool = False*, **args*, ***kwargs*)

获取行政地图的边界对象

参数

- **province**(*str*) –省/自治区/直辖市/行政特区中文名, 必须为全称, 例如查找河北省应收入 '河北省' 而非 '河北'. 默认为 None.

- **city** (*str*) –地级市中文名, 必须为全称, 例如查找北京市应输入 ' 北京市' 而非 ' 北京' . 默认为 None.
- **district** (*str*) –区/县中文名, 必须为全称. 默认为 None.
- **level** (*str*) –边界等级, 目前支持的等级包括 ' 省',' 市',' 区县'. 其中 ' 省' 级包括直辖市、特区等; ' 市' 级为地级市, 若为直辖市, 则名称与 ' 省' 级相同, 比如北京市的省级和市级都是 ' 北京市' ; ' 区' 和 ' 县' 属于同一级别的不同表达形式. 默认为 ' 省'.
- **country** (*str*) –国家名称, 必须为全称. 默认为 ' 中华人民共和国'.
- **source** (*str*) –数据源. 默认为 ' 高德'.
- **db** (*str*) –sqlite db 文件路径. 默认从配置文件中取.
- **engine** (*str*) –输出引擎, 默认为 None, 输出为 list 列表, 目前支持’geopandas’, 若为 geopandas, 则返回 GeoDataFrame 对象. 默认为 None.
- **record** (*str*) –返回记录的形式, 选项包括 'all' 和 'first' ; 若为 'first' , 则无论查询结果又几条, 仅返回第一条记录, 若为 'all' , 则返回全部数据, 若 engine==None 则返回 list, 若 engine=='geopandas' , 则返回 GeoDataFrame 对象. 默认为 'all' .
- **only_polygon** (*bool*) –是否仅返回地图边界对象 (MapPolygon), 若为 True 则返回结果为 MapPolygon 对象或以 MapPolygon 对象组合的 list, 若为 False , 则返回的结果包含元信息, MapPolygon 对象存储在 'geometry' 键中. 默认为 False .

返回 根据输入参数查找到的地图边界的元信息及边界对象

返回类型 GeoDataFrame or list

1.4.2 drawing

drawing 模块主要存放与绘图相关的函数

cnmaps.drawing.**clip_contours_by_map**(*contours*, *map_polygon*)

使用地图边界对象对等值线对象进行裁剪

参数

- **contours** (*cartopy.mpl.contour.GeoContourSet*) –等值线对象, 该对象是调用 ax.contour() 或 ax.contourf() 方法的返回值, 注意: 对象须带有投影信息
- **map_polygon** (cnmaps.maps.MapPolygon) –地图边界对象, 可以通过 get_adm_maps() 获取

cnmaps.drawing.**clip_pcolormesh_by_map**(*mesh*, *map_polygon*)

使用地图边界对象对填色网格线对象进行裁剪

参数

- **mesh** (*cartopy.mpl.geocollection.GeoQuadMesh*) –GeoQuadMesh 对象, 该对象是调用 ax.pcolormesh() 方法的返回值，注意: 对象须带有投影信息
- **map_polygon** (cnmaps.maps.MapPolygon) – 地图边界对象，可以通过 get_adm_maps() 获取

cnmaps.drawing.**clip_clabels_by_map**(*clabel_text*, *map_polygon*)

剪切 clabel 文本, 一般配合 contour 函数使用

注意: 该函数仅对于 cartopy>=0.19.0 版本有效

参数

- **clabel_text** (*matplotlib.text.Text*) –matplotlib.text.Text 对象, 由 clabel 函数返回
- **map_polygon** (cnmaps.maps.MapPolygon) – 地图边界对象，可以通过 get_adm_maps() 获取

cnmaps.drawing.**draw_map**(*map_polygon*, ***kwargs*)

绘制单个地图边界线

参数 map_polygon (cnmaps.maps.MapPolygon) –地图边界线对象

cnmaps.drawing.**draw_maps**(*maps*, ***kwargs*)

绘制多个地图边界

参数 maps (*list or GeoDataFrame*) –地图边界线对象

1.4.3 regions

regions 模块主要存放组合后的边界对象

cnmaps.regions.**region_polygons**

区域性组合地图多边形数据字典，包含的键有:

东北地区、华北地区、华中地区、华南地区、华东地区、西南地区、西北地区、川渝、京津冀、江浙沪、长三角

1.4.4 sample

sample 模块主要存放示例数据

cnmaps.sample.**load_dem**(*area_name*, ***kwargs*)

参数 **area_name** (*str*) –区域名称, 目前仅支持 ' 京津冀', 若为 None 则取全国. 默认为 None.

返回 (lons, lats, data)

1.5 贡献者指南

开发一个开源项目，尤其是一个有活力、生命力和影响力的开源项目是一件相当困难的事情，对于 cnmaps 的开发，我深知一个人能力有限，所以非常欢迎各位 Python 和 GIS 爱好者能和我一起来参与这个开源项目的继续开发。

1.5.1 哪些人适合参与到 cnmaps 的开发中？

在回答这个问题之前，我们先来明确一个概念：什么人可以被称为“开源贡献者”?

在我看来，并不是只有撸代码、实现功能的人才算得上“开源贡献者”，那些为开源项目做测试、提交 bug 报告、补充文档以及提出新功能需求和参与讨论的人，都算是“开源贡献者”。或者换句话说，所有努力让开源项目变得更好的人，都是“开源贡献者”。

现在我们来回答哪些人适合参与到 cnmaps 的开发中的问题，我的答案是：

1. 对开源项目有热情且愿意付出时间与人合作一起让开源项目变得更好的人。
2. 会使用或愿意学习使用 GitHub 的 Issue 功能的人。
3. 会使用或愿意学习使用 Git、Python 的人。
4. 有能力且有意愿补充和完善文档的人。
5. 对代码质量有追求的人（有代码洁癖的人）。
6. 对开源项目的功能有自己想法并且愿意分享想法的人。
7. …

1.5.2 如何参与到 cnmaps 的项目中来？

我大致将参与 cnmaps 项目的形式分为 **文档编写**、**测试**、**讨论**、**开发**这四个部分。

文档编写

cnmaps 的文档（也就是当前你正在阅读的这个文档）是使用 Python 语言的 sphinx 框架托管于 GitHub 并与 Readthedocs.org 集成实现的，若你想要完善文档，则可以 fork 本文档的代码仓库：cnmaps-doc ，在你本地进行修改和测试后向主仓库提交 Pull Request，在我（们）审核之后会自动合并到主分支并进行构建发布。

测试

以使用者的角度测试项目的各项功能，在发现 bug 以后，去 Issues 页面 提交议题（Issue），最好能针对发现的 bug 写出测试用例。你也可以在项目的 tests 目录中补充测试用例，这可能需要你能熟练使用 pytest 测试框架。

讨论

如果你对代码并不熟悉，或者对代码质量不自信，但是对项目的功能有自己的想法，也可以通过在 Issues 页面 中提出自己的功能需求的方式参与项目（类似于产品经理提出需求），当然也并不限于需求的讨论，对于 GIS 知识等非代码的一些认知偏差纠正或者指出、修复地图的拓扑错误也算，当然如果你能提供好的数据源也是非常重要的，比如你可以提供一套质量很高的地理边界。

开发

如果是想参与到 cnmaps 的代码的开发，相对来说限制会多一些。首先你必须对代码质量有追求，在平时撸代码的时候会尽量以优雅地方式实现代码，具体来说可能要满足以下几条：

1. 了解 PEP8 和 Google 的 Python 代码规范，并愿意遵守其大部分原则。
2. 在平时的开发中会使用 pylint 等工具对自己的代码进行自动化审查。
3. 能够接受“测试驱动开发”（TDD）的工作方式，认同“质量优于速度”的理念。
4. 会写测试用例，会进行单元测试。
5. 乐于交流，愿意用最直接高效的方式交流，不要求所谓的空杯心态，但也不要过于傲慢。

目前 cnmaps 接受开发贡献的方式是 fork + PR（Pull Request）+ 审查合并，但是在提交 PR 之前需要先有 Issue 议题来描述修改的动机和拟实现的功能及测试用例，所以正确的顺序是打开议题 -> fork 分支 -> 本地开发测试 -> 提交 PR -> 审查合并 -> 关闭议题。

1.6 许可证

1.6.1 中文版

BSD 3-Clause 许可证

Copyright (c) 2022 著作权由李文韬所有。著作权人保留一切权利。

这份许可证，在用户符合以下三条件的情形下，授予用户使用及再散播本软件包装源代码及二进制可执行形式的权利，无论此包装是否经改作皆然：

1. 对于本软件源代码的再散播，必须保留上述的著作权宣告、此三条件表列，以及下述的免责声明。
2. 对于本包二进制可执行形式的再散播，必须连带以文件以及／或者其他附于散播包装中的介质方式，重制上述之著作权宣告、此三条件表列，以及下述的免责声明。
3. 未获事前获取书面许可，不得使用著作权人或本软件贡献者之名称，来为本软件之派生物做任何表示支持、认可或推广、促销之行为。

免责声明：本软件是由著作权人及本软件之贡献者以现状提供，本软件包装不负任何明示或默示之担保责任，包括但不限于就适售性以及特定目的的适用性为默示性担保。著作权人及本软件之贡献者，无论任何条件、无论成因或任何责任主义、无论此责任为因合约关系、无过失责任主义或因非违约之侵权（包括过失或其他原因等）而起，对于任何因使用本软件包装所产生的任何直接性、间接性、偶发性、特殊性、惩罚性或任何结果的损害（包括但不限于替代商品或劳务之购用、使用损失、资料损失、利益损失、业务中断等等），不负任何责任，即在该种使用已获事前告知可能会造成此类损害的情形下亦然。

1.6.2 英文原文

BSD 3-Clause License

Copyright (c) 2022, Wentao Li All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT

1.7 资料引用

1. 谢栋灿. 高德行政区边界获取与整理（shp 格式）[EB/OL]. [2017.11.05]. http://i.xdc.at/2017/11/05/amap-district-to-shapefile/
2. GaryBikini/ChinaAdminDivisonSHP: v2.0, 2021, DOI: 10.5281/zenodo.4167299
3. 张镱锂, 李炳元, 郑度. 青藏高原范围与界线地理信息系统数据 [J/DB/OL]. 全球变化数据仓储电子杂志(中英文), 2014. https://doi.org/10.3974/geodb.2014.01.12.V1.
4. NASA/METI/AIST/Japan Spacesystems and U.S./Japan ASTER Science Team (2019). ASTER Global Digital Elevation Model V003 [Data set]. NASA EOSDIS Land Processes DAAC. Accessed 2022-04-05 from https://doi.org/10.5067/ASTER/ASTGTM.003

1.8 版本日志

1.8.1 1.0.1

发布时间: 2022-04-05

- 修复了安装依赖缺少 geopandas 的问题。

1.8.2 1.0.0

发布时间: 2022-04-05

- 增加了 `cnmaps.maps.get_adm_maps` 函数以支持对市、区县等行政边界的查询，高德的行政边界数据上已下探到区县级别。
  - 支持多条件筛选和一次性返回多个对象功能。
  - 集成了 geopandas 引擎。
- 增加了 `cnmaps.maps.get_adm_names` 函数以支持对支持的行政区关键字的查询。
- 增加了 `cnmaps.drawing.draw_maps` 函数以方便一次性绘制多个地图边界。

- 增加了 cnmaps.regions.region_polygons 可以直接获取例如华北地区、华东地区、京津冀、江浙沪等边界对象。
- 删除了 cnmaps.get_map 函数，其行政边界查询的功能转移至 get_adm_maps ，但用法上会有一些变化。
- 暂时删除了对青藏高原边界的支持。

1.8.3 0.2.1

发布时间: 2022-03-02

- 修复了 Windows 系统中 GBK 编码无法加载数据的问题。

1.8.4 0.2.0

发布时间: 2022-02-16

- 增加了对 pcolormesh 图的裁剪支持。
- 修复了边界错误的问题。

1.8.5 0.1.11

发布时间: 2022-02-14

- 尝试修复安装时出现 gbk 编码异常的问题。

1.8.6 0.1.10

发布时间: 2022-02-13

- 增加功能: cnmaps.get_map 函数: 获取地图。
- 增加功能: cnmaps.draw_map 函数: 绘制地图。
- 增加功能: cnmaps.MapPolygon 类: 地图对象, 包括: 加号 (合并)、减号 (剪切)、逻辑与 (交集) 运算符的支持，get_extent 方法。
- 增加功能: cnmaps.clip_contours_by_map 函数: 基于 MapPolygon 类对等值线图做裁减。
- 增加功能: cnmaps.sample.load_dem 函数: 加载 dem 样例数据。
- 增加功能: cnmaps.clip_clabels_by_map 函数: 基于 MapPolygon 类对标签做裁减。
- 对 cartopy.crs 各类投影的支持。
- 对全国中国国界、全国各省 (特区/直辖市) 地图的预置, 且处理了已知的拓扑错误。
- 集成了 travis CI 自动化测试。

索引

C

D

G

L

M