臺澎dtm切割介面

背景

資料之選擇

- 近年來內政部20m DTM([數位地形模式][wiki] [^1])數據並沒有顯著的更新（詳dtm檔案收集情況），由於2020版本DTM在苗栗地區有顯著的缺值，因此使用2018年版本。
- 解析度：雖內政部也有1m解析度的DTM，但以噪音模式動輒10～100公頃範圍，將造成計算困難。全島之總體處理與儲存也有很大的挑戰。
- 此處選擇以20m解析度尚稱合理。

DTM的前處理

- 最花時間的步驟為直角座標系統與經緯度系統的轉換，如果每次轉換將會造成速度瓶頸。
- 轉換後的儲存格式：按照GPT的建議，以記憶體映射的方式儲存最為有效，詳見Geotiff格式DTM之前處理。

系統架構策略考量

- 前台
  - 原生的leaflet與html搭配是最穩定、功能齊全且易於發展的方案。
  - streamlit網頁設計雖然較為簡潔，但元件不多、與js、css等還是有很大的扞格。
  - 由於等高線是專案關切主題、而不是地面資訊，此處選擇Mapbox的圖磚，較內政部圖磚、openTopoMap等來得清晰簡潔、遮蔽較少，有較高的地圖品質（雖然尚未全面中文化）。
- API伺服器
  - 此處只有傳送邊界座標(`bounds`:東北、西南經緯度座標共4個數字)、以及切割後處理好的地形圖檔(`.png`或`.dxf`後者詳)，算是單純，不考慮複雜且功能完整的API伺服器。
  - GPT建議以Flask來撰寫最簡潔穩定，且最受歡迎。
  - Flask伺服器可以同時營運html及多個API，程式架構單純。
- 資料庫
  - 因為是格柵資料，具有很高的系統性，不需要建立關聯或非關聯資料庫與查詢系統
  - 直接以格柵檔案進行存取最有效率
  - 如需更換DTM來源，所需的轉換作業也最單純。

前台畫面與應用

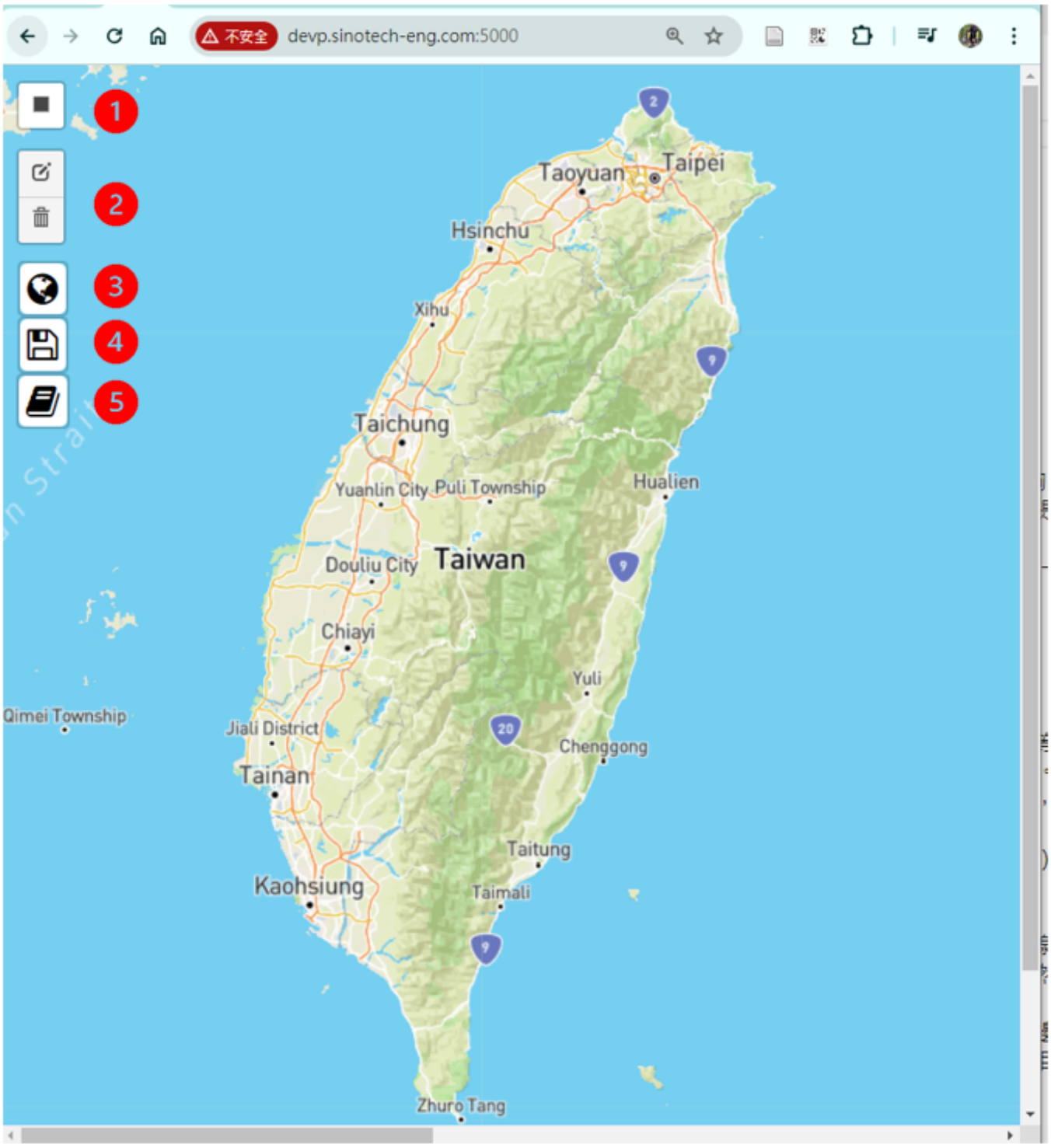
服務網址

- devp.sinotech-eng.com:5000

進入畫面與滑鼠內設功能

- 進入伺服器後，滑鼠默認自動進入矩形圖框切割功能、選取縮放(`fitBounds`)之後，會清除前次選取圖框、並再次進入切割狀態。
  - 此狀態下可接受滾輪放大縮小、但不能接受平移(pan)
  - 如果要移動地圖，需要按`esc`鍵、或點選垃圾桶`clear all`來停止切割功能。

- 畫面如下，說明如後。



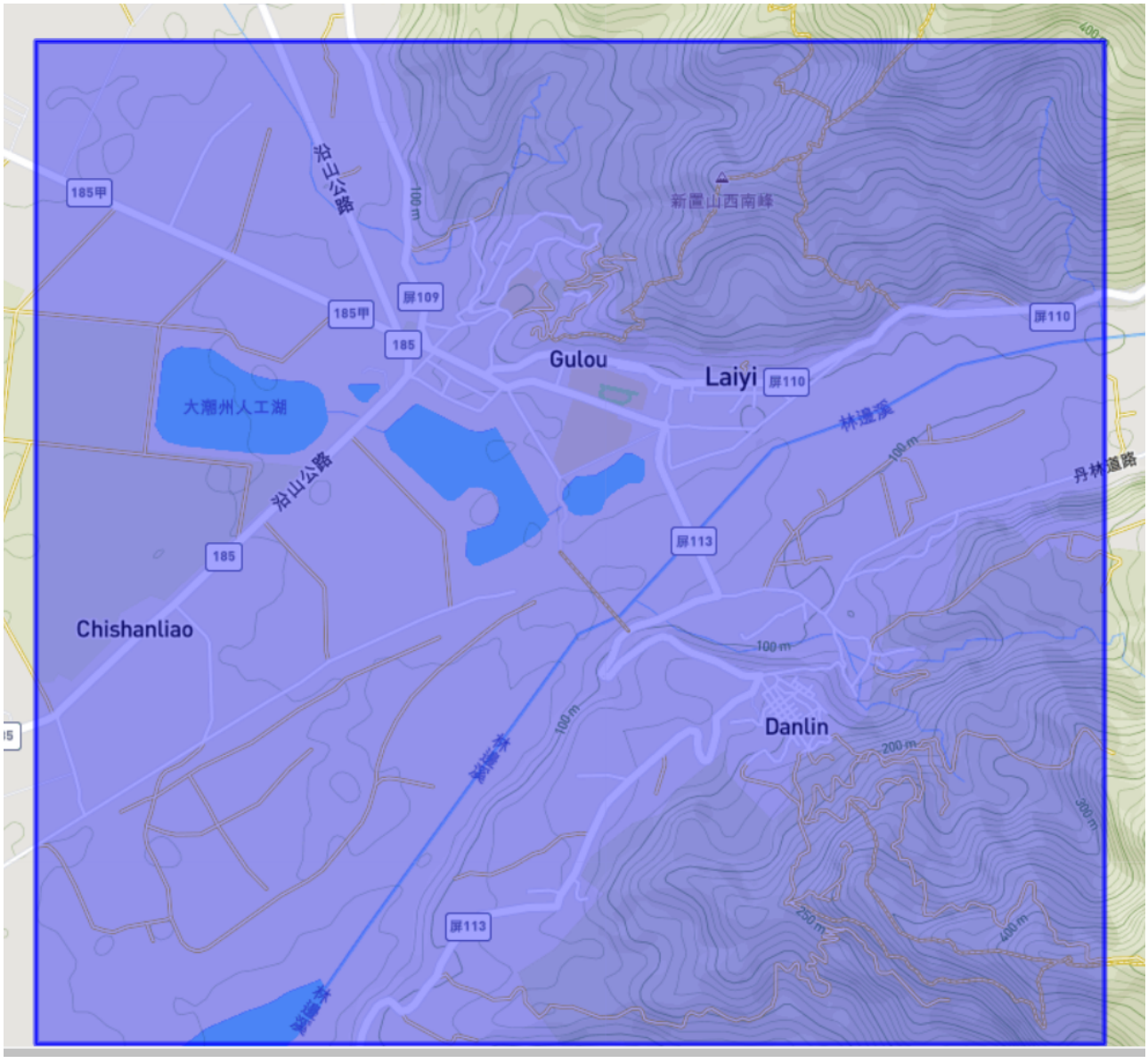
功能鍵說明

1. 矩形圖框切割功能：滑鼠雖然內設具有矩形圖框切割功能，但經取消(按esc鍵)後，如要繼續選取，可以點選黑色白底四方形鍵再次啟動切割功能。
2. 編輯圖框與清除選取
3. 將選取結果送交等高線繪製API程式，圖檔可直接與地圖等高線定性比較(詳下)。
    - 點選此鍵前需先按esc鍵取消繼續切割。
    - 點選後系統將會在主機上產生等高線matplotlib圖檔，同時也在客戶端下載目錄儲存一份。檔名為terr_隨機碼.png。
    - 高度值：選取範圍內最低到最高共9個間隔。

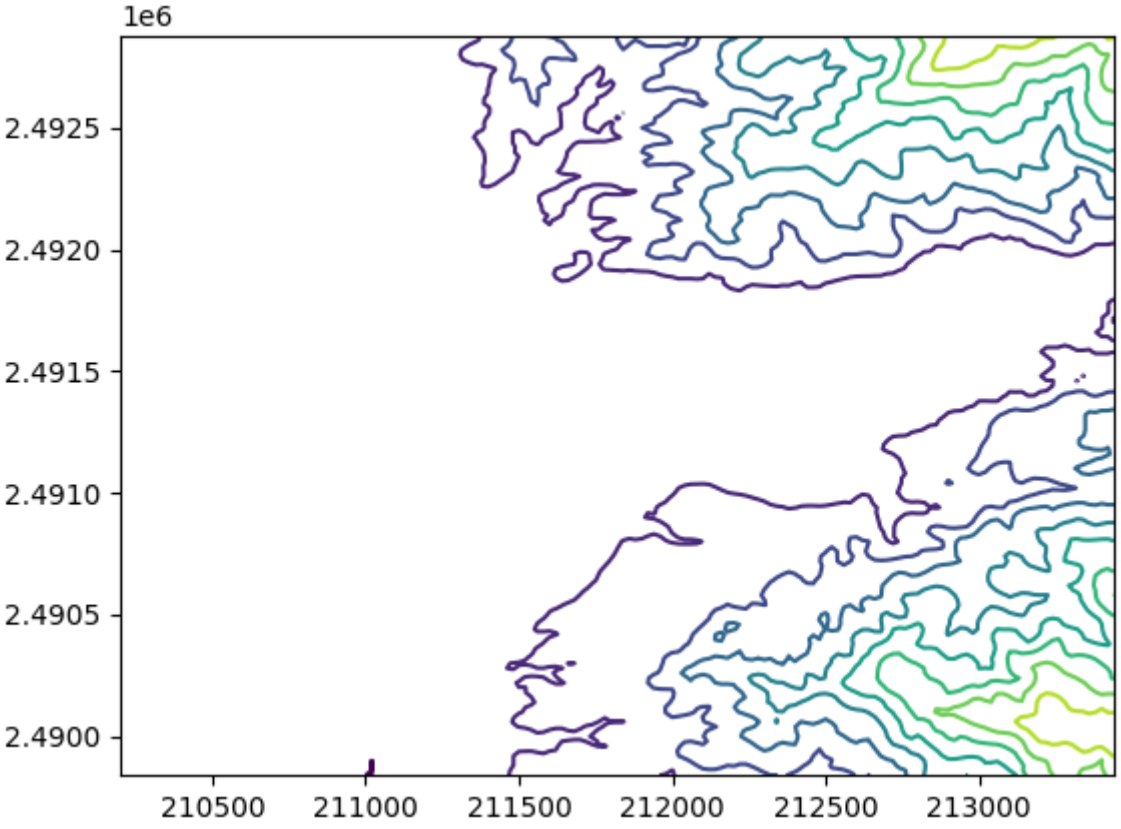
4. 功能與3.幾乎一樣，但為每公尺的等高線`dxf`檔
    - 檔名為`terr_隨機碼.dxf`
    - 可以用線上dxf viewer來檢視，如sharecad.org
5. 使用手冊
    - 公司內vuepress
    - 公司外github page

案例比較

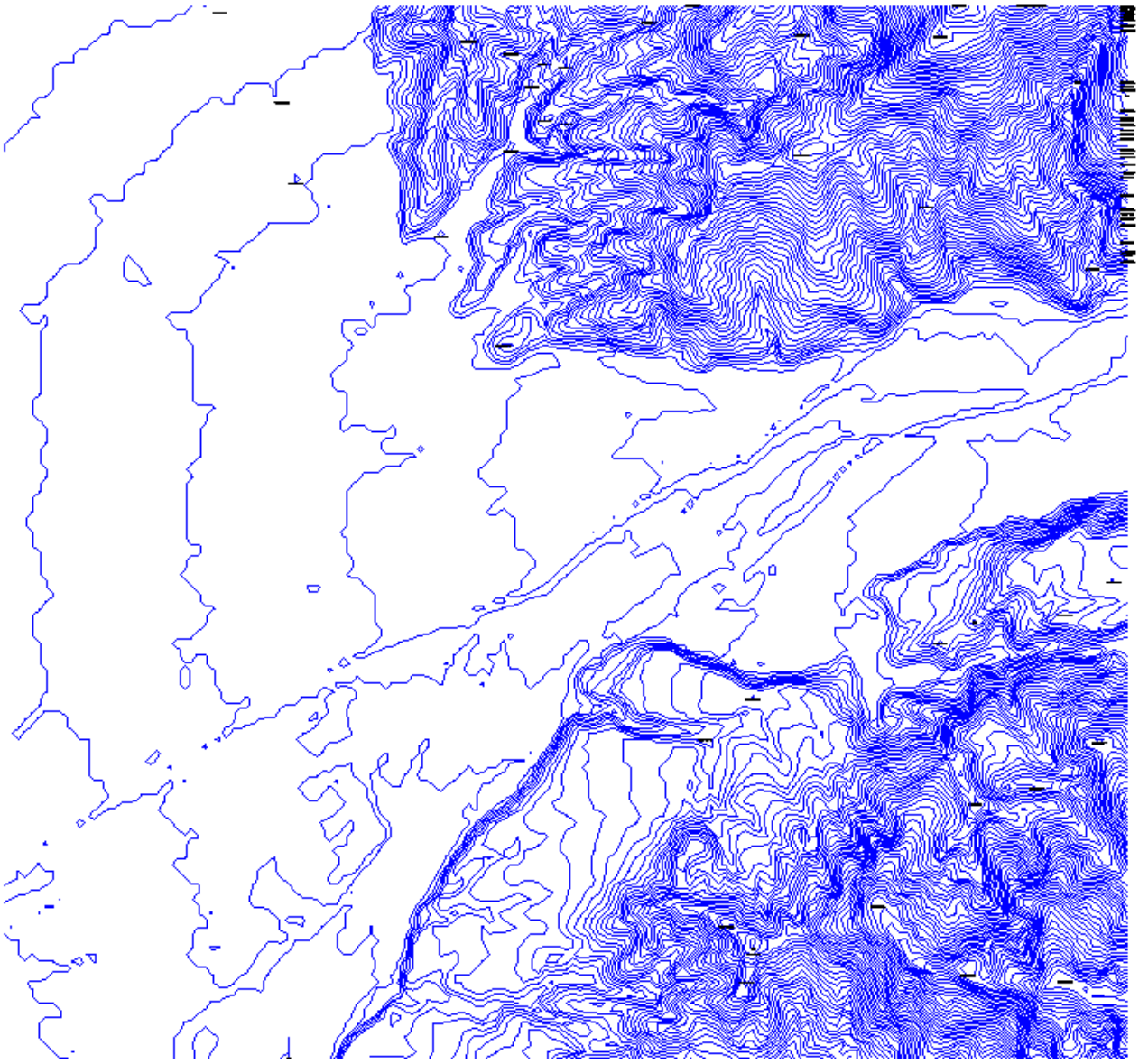
- 個案位置：屏東縣來義鄉
- mapbox切割範圍（定稿為紅色框線）



- matplotlib等高線圖：最低～最高10個等級



- dxf檔案
  - viewer:使用sharecad.org
  - 每一公尺一層



[^1]: "數值高程模型（DEM）或數字地表模型（DSM）是一種3D電腦圖形表示，用於表現地形資料，代表地形或覆蓋物體，通常是指行星、月球或小行星的地形。"全球DEM"指的是一個離散的全球網格。DEM在地理資訊系統（GIS）中經常被使用，並且是數字製作的地形圖最常見的基礎。數字地形模型（DTM）特指地面表面，而DEM和DSM可能代表樹頂冠層或建築物屋頂。[wiki][wiki] "

[wiki]: https://zh.wikipedia.org/zh-tw/数字地面模型 "數值高程模型（DEM）或數字地表模型（DSM）是一種3D電腦圖形表示，用於表現地形資料，代表地形或覆蓋物體，通常是指行星、月球或小行星的地形。"全球DEM"指的是一個離散的全球網格。DEM在地理資訊系統（GIS）中經常被使用，並且是數字製作的地形圖最常見的基礎。數字地形模型（DTM）特指地面表面，而DEM和DSM可能代表樹頂冠層或建築物屋頂。 (wiki)"

---

dtm檔案收集情況

背景

內政部2012

- 解析度：30m (28.88m)

- path `/home/QGIS/Data/dtm`
- filename: `twdtm_asterv2_30m-tm2_twd97.tif`
- same: `taiwan.tif`

```
$ tiffinfo twdtm_asterv2_30m-tm2_twd97.tif
TIFFReadDirectory: Warning, twdtm_asterv2_30m-tm2_twd97.tif: unknown field with
tag 33550 (0x830e) encountered.
TIFFReadDirectory: Warning, twdtm_asterv2_30m-tm2_twd97.tif: unknown field with
tag 33922 (0x8482) encountered.
TIFFReadDirectory: Warning, twdtm_asterv2_30m-tm2_twd97.tif: unknown field with
tag 42112 (0xa480) encountered.
TIFFReadDirectory: Warning, twdtm_asterv2_30m-tm2_twd97.tif: unknown field with
tag 42113 (0xa481) encountered.
TIFF Directory at offset 0x21559648 (559257160)
  Image Width: 14400 Image Length: 19224
  Tile Width: 128 Tile Length: 128
  Bits/Sample: 16
  Sample Format: signed integer
  Compression Scheme: None
  Photometric Interpretation: min-is-black
  Samples/Pixel: 1
  Planar Configuration: single image plane
  Tag 33550: 28.884201,28.884201,0.000000
  Tag 33922: 0.000000,0.000000,0.000000,42033.952431,2878094.424011,0.000000
  Tag 42112: <GDALMetadata>
  <Item name="PyramidResamplingType" domain="ESRI">NEAREST</Item>
  <Item name="STATISTICS_MINIMUM" sample="0">0</Item>
  <Item name="STATISTICS_MAXIMUM" sample="0">3893</Item>
  <Item name="STATISTICS_MEAN" sample="0">125.13239506028</Item>
  <Item name="STATISTICS_STDDEV" sample="0">438.65405799382</Item>
</GDALMetadata>

  Tag 42113: 32767
```

- `taiwan2.tiff`解析度與範圍都一樣，似乎為格式版本的差異(tif vs tiff)

```
kuang@master /home/QGIS/Data/dtm
$ tiffinfo taiwan2.tiff
TIFFReadDirectory: Warning, taiwan2.tiff: unknown field with tag 33550 (0x830e)
encountered.
TIFFReadDirectory: Warning, taiwan2.tiff: unknown field with tag 33922 (0x8482)
encountered.
TIFFReadDirectory: Warning, taiwan2.tiff: unknown field with tag 42113 (0xa481)
encountered.
TIFF Directory at offset 0x210266e4 (553805540)
  Image Width: 14400 Image Length: 19224
  Bits/Sample: 16
  Sample Format: signed integer
  Compression Scheme: None
  Photometric Interpretation: min-is-black
```

```
  Samples/Pixel: 1
  Rows/Strip: 1
  Planar Configuration: single image plane
  Tag 33550: 28.884201,28.884201,0.000000
  Tag 33922: 0.000000,0.000000,0.000000,42033.952431,2878094.424011,0.000000
  Tag 42113: 32767
```

內政部2016

- 解析度：20m

```
kuang@master /home/QGIS/Data/dtm_20mTaiPenhu
$ tiffinfo dem_20m.tif
TIFFReadDirectory: Warning, dem_20m.tif: unknown field with tag 33550 (0x830e)
encountered.
TIFFReadDirectory: Warning, dem_20m.tif: unknown field with tag 33922 (0x8482)
encountered.
TIFFReadDirectory: Warning, dem_20m.tif: unknown field with tag 34735 (0x87af)
encountered.
TIFFReadDirectory: Warning, dem_20m.tif: unknown field with tag 34736 (0x87b0)
encountered.
TIFFReadDirectory: Warning, dem_20m.tif: unknown field with tag 34737 (0x87b1)
encountered.
TIFFReadDirectory: Warning, dem_20m.tif: unknown field with tag 42113 (0xa481)
encountered.
TIFF Directory at offset 0x1730edde (389082590)
  Image Width: 10175 Image Length: 19112
  Bits/Sample: 16
  Sample Format: signed integer
  Compression Scheme: None
  Photometric Interpretation: min-is-black
  Samples/Pixel: 1
  Rows/Strip: 1
  Planar Configuration: single image plane
  Tag 33550: 20.000000,20.000000,0.000000
  Tag 33922: 0.000000,0.000000,0.000000,148310.000000,2801730.000000,0.000000
  Tag 34735:
1,1,0,17,1024,0,1,1,1025,0,1,1,1026,34737,8,0,2048,0,1,4326,2049,34737,7,8,2054,0,
1,9102,2057,34736,1,6,2059,34736,1,5,3072,0,1,32767,3074,0,1,32767,3075,0,1,1,3076
,0,1,9001,3080,34736,1,1,3081,34736,1,0,3082,34736,1,3,3083,34736,1,4,3092,34736,1
,2
  Tag 34736:
0.000000,121.000000,0.999900,250000.000000,0.000000,298.257224,6378137.000000
  Tag 34737: unnamed|WGS 84|
  Tag 42113: -32767
```

內政部2020

- 解析度：20m

```
kuang@master /home/QGIS/Data/dtm_20mTaiPenhu/2020台灣本島及離島(小琉球、龜山島、綠島及蘭嶼)
$ tiffinfo taiwan2020.tif
TIFFReadDirectory: Warning, taiwan2020.tif: unknown field with tag 33550 (0x830e) encountered.
TIFFReadDirectory: Warning, taiwan2020.tif: unknown field with tag 33922 (0x8482) encountered.
TIFFReadDirectory: Warning, taiwan2020.tif: unknown field with tag 34735 (0x87af) encountered.
TIFFReadDirectory: Warning, taiwan2020.tif: unknown field with tag 34736 (0x87b0) encountered.
TIFFReadDirectory: Warning, taiwan2020.tif: unknown field with tag 34737 (0x87b1) encountered.
TIFFReadDirectory: Warning, taiwan2020.tif: unknown field with tag 42113 (0xa481) encountered.
TIFF Directory at offset 0x2d1a9eb8 (756719288)
  Image Width: 10035 Image Length: 18852
  Bits/Sample: 32
  Sample Format: IEEE floating point
  Compression Scheme: None
  Photometric Interpretation: min-is-black
  Orientation: row 0 top, col 0 lhs
  Samples/Pixel: 1
  Rows/Strip: 1
  SMin Sample Value: -24.28
  SMax Sample Value: 3947.27
  Planar Configuration: single image plane
  Tag 33550: 20.000000,20.000000,1.000000
  Tag 33922: 0.000000,0.000000,0.000000,150980.000000,2799160.000000,0.000000
  Tag 34735:
1,1,0,31,1024,0,1,1,1025,0,1,2,1026,34737,13,0,2048,0,1,4326,2054,0,1,9102,2056,0,1,7030,2057,34736,1,0,2058,34736,1,1,2059,34736,1,2,3072,0,1,32767,3074,0,1,32767,3075,0,1,1,3076,0,1,9001,3078,34736,1,3,3079,34736,1,4,3080,34736,1,5,3081,34736,1,6,3082,34736,1,7,3083,34736,1,8,3084,34736,1,9,3085,34736,1,10,3086,34736,1,11,3087,34736,1,12,3088,34736,1,13,3089,34736,1,14,3090,34736,1,15,3091,34736,1,16,3092,34736,1,17,3094,34736,1,18,3095,34736,1,19,4099,0,1,9001
  Tag 34736:
6378137.000000,6356752.314245,298.257224,0.000000,0.000000,121.000000,0.000000,250000.000000,0.000000,0.000000,0.000000,0.000000,0.000000,0.000000,0.000000,0.000000,0.000000,0.999900,0.000000,0.000000
  Tag 34737: GCS_WGS_1984|
  Tag 42113: -32767
```

Geotiff格式DTM之前處理

背景

前處理目標與必要性

- 直角座標數位高程模型資訊格柵檔案的處理並不困難，但是因為格點數量龐大，座標轉換會花很多時間，如果每一次處理都要轉換，將會造成瓶頸。
- GeoTiff格式檔案並未儲存每個格點的經緯度或TWD97座標，後者還可以線型計算，前者則需轉換，計算過程需時較久，偏偏一般地圖裁切介面都是使用經緯度座標系統。

座標轉換的精確性

- 直角座標系統轉經緯度一般採取Lambert轉換，轉換的精準度與南北割線緯度(true latitude)有關，一般範例以10/40度為基準，轉換將會比實際略為偏南。
- 此處以臺灣本島的最南與最北端緯度代入，來降低轉換的誤差。

輸出檔案格式與讀寫的有效性

- 作為大型矩陣切割(過濾)的方式，可以用pandas.DataFrame篩選，也可以用`np.where`線性化篩選。然前者將會儲存成十數G的csv，而後者分開以二進位檔案儲存，將可大量壓縮暫存檔案的容量。
- 全島GeoTiff檔案經壓縮約為500MB、解開後地形檔、格點中心座標（lat、lon）三個2D陣列合計約為2GB。讀寫、篩選都較DataFrame有效率。

程式說明

這段程式碼使用 Python 中的 `rasterio` 和 `pyproj` 庫來處理地理空間數據，具體操作如下：

輸入與翻轉

1. **讀取 GeoTIFF 文件：**
    - 使用 `rasterio.open` 打開名為 `taiwan2018.tif` 的 GeoTIFF 文件。
    - 獲取影像的寬度、高度和波段數。
2. **讀取影像數據並翻轉：**
    - 使用 `img.read()[0,:,:]` 讀取第一個波段的數據，
    - 並使用 `np.flipud` 進行南北方向數據的翻轉（upside down）。

座標轉換

1. **獲取地理坐標及影像的變換參數：**
    - 使用 `img.xy(0,0)` 獲取影像左上角的地理坐標。
    - 獲取影像的變換參數 `transform`，其中包括像元大小 `dx` 和 `dy`（取絕對值）。
    - 計算影像的南北座標y值範圍。
2. **計算影像像元的地理坐標：**
    - 創建 x 和 y 座標的 numpy 陣列。
    - 將 x 和 y 座標中心化。
    - 使用 `np.meshgrid` 創建2維網格點每一點中心的TWD97座標值。
3. **定義投影和轉換地理坐標：**
    - 設置投影參數（這裡使用了 Lambert Conformal Conic 投影）。
    - 使用 `pnyc` 將網格點轉換為經緯度。

儲存檔案

1. 將地理資訊參數（包括原點坐標、影像尺寸和像元大小）保存到 `params.txt` 文件。
2. **將數據保存到文件：**
    - 將影像數據、經度和緯度數據保存為 `.dat` 文件。

程式碼附加說明

- 以下是完整的代碼與代碼分段說明：

輸入與基本設定

```
import rasterio
import numpy as np
from pyproj import Proj

# 打開 GeoTIFF 文件
fname = 'taiwan2018.tif'
img = rasterio.open(fname)

# 獲取影像寬度、高度和波段數
nx, ny, nz = img.width, img.height, img.count
```

翻轉南北方向

- GeoTiff

```
# 讀取影像數據並翻轉
data = np.flipud(img.read()[0,:,:])

# 獲取影像左上角的地理坐標
x0, y0 = img.xy(0, 0)

# 獲取影像的變換參數
transform = img.transform
dx, dy = transform.a, abs(transform.e)

# 計算影像的 y 座標範圍
y0 = y0 - dy * ny

# 創建 x 和 y 座標的 numpy 陣列
x = np.array([x0 + dx * i for i in range(nx)])
y = np.array([y0 + dy * i for i in range(ny)])

# 將 x 和 y 座標中心化
xcent, ycent = x[nx // 2], y[ny // 2]
x -= xcent
y -= ycent
```

```
# 使用 np.meshgrid 創建網格點
xg, yg = np.meshgrid(x, y)
```

座標轉換

```
# 定義投影參數
Longitude_Pole, Latitude_Pole = img.lnglat()
pnyc = Proj(proj='lcc', datum='NAD83', lat_1=21.8, lat_2=25.4,
            lat_0=Latitude_Pole, lon_0=Longitude_Pole, x_0=0, y_0=0.0)

# 將網格點轉換為經緯度
lon, lat = pnyc(xg, yg, inverse=True)
```

儲存2維陣列

- 直角座標之數值，雖會被運用在後續等高線的計算，但屬線性計算，速度很快，因此就不儲存了。
- 按照GPT建議，以記憶體映射方式之二進位檔案存取最有效率。
- 儲存方式，先將陣列記憶體與檔案連在一起。直接將數據上傳到記憶體，就會儲存到檔案了，沒有（不需要）`write`指令。

```
# 定義數據形狀
shape = (ny, nx)

# 保存數據到 .dat 文件
fnames = ['data', 'lon', 'lat']
arrays = [data, lon, lat]
for f in range(3):
    filename = fnames[f] + '.dat'
    memmap_array = np.memmap(filename, dtype='float32', mode='w+', shape=shape)
    memmap_array[:] = arrays[f][:,:]

# 保存參數到 params.txt 文件
params_str = f"{x0} {y0} {nx} {ny} {dx} {dy}\n"
with open('params.txt', 'w+') as f:
    f.write(params_str)
```

這段代碼將讀取 GeoTIFF 文件中的數據，將其轉換為經緯度，並將結果保存為 .dat 文件和一個參數文件。您可以根據需要進一步處理或可視化這些數據。

程式下載

{% include download.html content="img2mem.py" %}

DTM等值線圖的產生

背景

- 等高線dxf圖檔事實上是`matplotlib.pyplot.contour`過程的中間產物，由每條折線上的逐點座標所組成，一般程式設計者是不會需要知道每條等高線上的詳細座標的。
- 除了最終成果的控制目的之外，視覺化驗證也是本項工作一個很重要的理由。
- 處理好的龐大矩陣，也需要藉由本程式來測試看看讀取、篩選過程的工作效率到底怎樣、是否還需要進一步優化。
- 這支程式有函式及獨立運作等2個版本。
    - 函式型態，可以在API伺服器者、或其他程式場合呼叫
    - 獨立運作，雖然需要手動輸入圖框邊界座標，但還算是一個滿方便順手的工具。

程式說明

這段程式碼讀取存儲在 `.dat` 文件中的經緯度和數據，並根據給定的地理界限生成等高線圖，最終保存為 PNG 文件。它使用了 `matplotlib` 庫來繪製圖形，並使用 `tempfile` 來生成臨時文件名。以下是代碼的詳細解釋：

匯入必要的庫

- `numpy` 用於數據處理。
- `matplotlib` 用於繪圖。
- `tempfile` 和 `BytesIO` 用於處理臨時文件和內存中的文件。

讀取數據文件

- `rd_mem` 函數讀取經緯度和DTM數據的 `.dat` 文件，返回一個包含這3項數據的序列。
- 使用記憶體映射的方式讀取
    - 定義矩陣形狀與讀取檔案同一個指令完成
    - 副程式無法執行`exec()`指令，GPT建議使用`dict`或`list`方式依序讀取、回應呼叫的程式。

生成等高線圖

- 雖然等高線圖不是本次專案的目標，但是因為DXF檔案檢視不易，DTM數據量又很大，需要有效率的工具來進行過程驗證。
- `cntr` 函數根據圖框給定的地理邊界 `swLL`（西南角）和 `neLL`（東北角），在數據中以`np.where()`篩選符合條件的經緯度索引(`idx`)。
- 如果沒有找到符合條件的數據，返回錯誤信息。
- 根據找到的經緯度索引、計算繪圖的直角座標(TWD97)範圍、以及等高線圖的級別（9個層級共10條等高線）。
- 使用 `matplotlib` 繪製等高線圖，並保存為 PNG 文件。

程式碼

以下是完整的代碼：

```
# 匯入必要的庫
import numpy as np
import matplotlib
matplotlib.use('Agg')  # 使用 Agg 后端
import matplotlib.pyplot as plt
```

```
import tempfile as tf
from io import BytesIO

# 讀取數據文件
def rd_mem(shape):
    fnames = ['lat', 'lon', 'data']
    d = []
    for f in fnames:
        filename = f + '.dat'
        d.append(np.memmap(filename, dtype='float32', mode='r', shape=shape))
    return d

# 生成(swLL, neLL)範圍的等高線圖
def cntr(swLL, neLL):

    # 讀取GeoTiff檔案的基本參數、用在產生網格點的TWD97座標
    with open('params.txt', 'r') as f:
        line = [i.strip('\n') for i in f][0]
    x0, y0, nx, ny, dx, dy = (float(i) for i in line.split())
    nx, ny = int(nx), int(ny)
    shape = (ny, nx)

    # 讀取全島緯度、精度、以及高程等3項矩陣
    lat, lon, data = rd_mem(shape)

    # 負值歸0，避免拉大最低、最高值的區間
    data = np.where(data < 0, 0, data)

    # 利用經緯度進行範圍切割：共有4個條件都必須符合。
    idx = np.where((lat >= swLL[0]) & (lat <= neLL[0]) & (lon >= swLL[1]) & (lon
<= neLL[1]))

    # 'LL not right!'是關鍵字，app.py中會檢核。
    if len(idx[0]) == 0:
        return 'LL not right!', list(swLL) + list(neLL)

    # TWD97座標值：
    ## 1維
    x = [x0 + dx * i for i in range(nx)]
    y = [y0 + dy * i for i in range(ny)]
    ## 2維（y,x）
    xg, yg = np.meshgrid(x, y)

    # 取4個方向的TWD97座標極值，用於標定位置
    bounds = [np.min(xg[idx[0], idx[1]]), np.max(xg[idx[0], idx[1]]),
np.min(yg[idx[0], idx[1]]), np.max(yg[idx[0], idx[1]])]

    # 起訖點的索引
    ib = [x.index(bounds[0]), x.index(bounds[1]), y.index(bounds[2]),
y.index(bounds[3])]

    # 範圍內地形的極值、用於計算間距(固定10層)
    cmin = data[ib[2]:ib[3]+1, ib[0]:ib[1]+1].min()
    cmax = data[ib[2]:ib[3]+1, ib[0]:ib[1]+1].max()
```

```
    levels = np.linspace(cmin, cmax, 10)

    # 新的圖面(使用Agg避免顯示器干擾)
    fig, ax = plt.subplots()

    ## 繪圖
    ax.contour(x[ib[0]:ib[1]+1], y[ib[2]:ib[3]+1], data[ib[2]:ib[3]+1,
ib[0]:ib[1]+1], levels=levels)

    ## 存檔、輸出到BytesIO供前後端API傳輸
    ran = tf.NamedTemporaryFile().name.replace('/', '').replace('tmp', '')
    fname = 'terr_' + ran + '.png'
    output = BytesIO()
    fig.savefig(output, format='png')
    output.seek(0)  # 重置指针位置
    fig.savefig('./pngs/' + fname, format='png')

    return fname, output
```

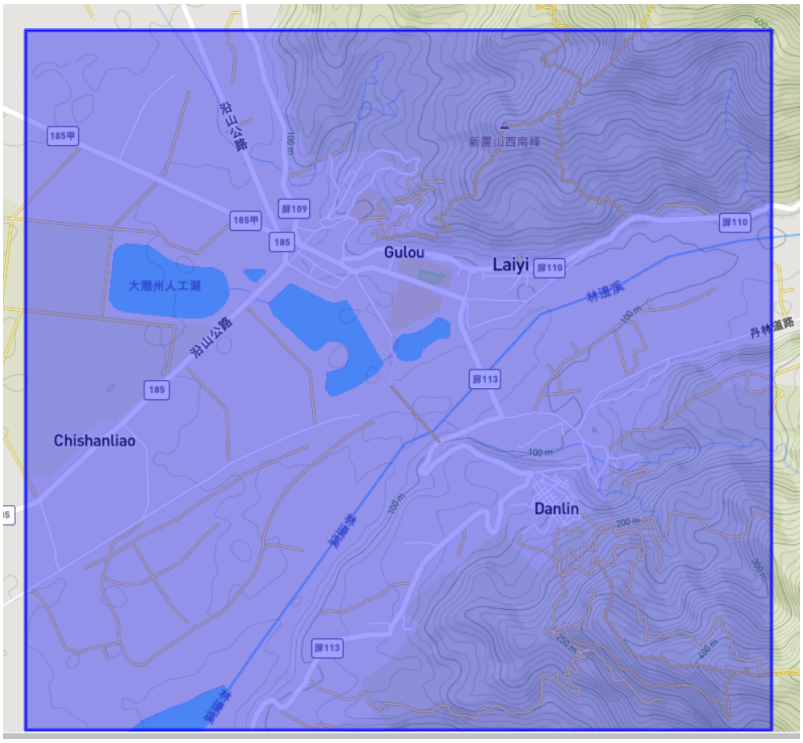
這段代碼的關鍵步驟包括：

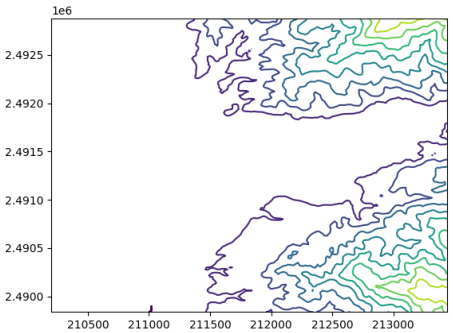
- 讀取地理數據。
- 查找給定界限內的數據。
- 繪製等高線圖並保存為 PNG 文件。

程式下載

- 副程式版本{% include download.html content="mem2cntr.py" %}，供API呼叫、自動回復。
- 獨立程式版本{% include download.html content="mem2ccontour.py" %}，獨立手動運作。

圖檔結果比較





等值線圖DXF檔的產生

背景

- 等高線dxf圖檔事實上是`matplotlib.pyplot.contour`過程的中間產物，由每條折線上的逐點座標所組成，一般程式設計者是不會需要知道每條等高線上的詳細座標的。
- 除了最終成果的控制目的之外，視覺化驗證也是本項工作一個很重要的理由。
- 處理好的龐大矩陣，也需要藉由本程式來測試看看讀取、篩選過程的工作效率到底怎樣、是否還需要進一步優化。
- 這支程式只有函式版本，可以在API伺服器者、或其他程式場合呼叫

程式說明

這段程式碼讀取存儲在 `.dat` 文件中的經緯度和數據，並根據給定的地理界限生成等高線圖，最終保存為 PNG 文件。它使用了 `matplotlib` 庫來繪製圖形，並使用 `tempfile` 來生成臨時文件名。以下是代碼的詳細解釋：

匯入必要的庫

- `numpy` 用於數據處理。
- `matplotlib` 用於繪圖。
- `tempfile` 和 `BytesIO` 用於處理臨時文件和內存中的文件。
- `ezdxf`程式庫，用於創建和操作 DXF 文件。

讀取數據文件

- `rd_mem` 函數讀取經緯度和DTM數據的 `.dat` 文件，返回一個包含這3項數據的序列。
- 使用記憶體映射的方式讀取
    - 定義矩陣形狀與讀取檔案同一個指令完成
    - 副程式無法執行`exec()`指令，GPT建議使用`dict`或`list`方式依序讀取、回應呼叫的程式。

篩選DTM

- 雖然等高線圖不是本次專案的目標，但是因為DXF檔案檢視不易，DTM數據量又很大，需要有效率的工具來進行過程驗證。
- `dxf` 函數根據圖框給定的地理邊界 `swLL`（西南角）和 `neLL`（東北角），在數據中以`np.where()`篩選符合條件的經緯度索引(`idx`)。
- 如果沒有找到符合條件的數據，返回錯誤信息。

生成 DXF 文件

- `dxf` 函數根據給定的地理界限 `swLL`（西南角）和 `neLL`（東北角），在數據中查找符合條件的經緯度索引。
- 如果沒有找到符合條件的數據，返回錯誤信息。
- 根據找到的經緯度索引計算繪圖的範圍和等高線圖的級別。
- 使用 `matplotlib` 繪製等高線圖，並將這些線條和文本標籤添加到 DXF 文件中。

- 使用 `ezdxf` 創建一個新的 DXF 文檔，並保存等高線和標籤。
- 將 DXF 文件保存到內存中，並返回文件名和內存中的文件對象。

程式碼

以下是完整的代碼：

```
# 匯入必要的庫
import numpy as np
import matplotlib
matplotlib.use('Agg')  # 使用 Agg 后端
import matplotlib.pyplot as plt
import tempfile as tf
from io import BytesIO

# ezdxf程式庫
import ezdxf
from ezdxf import colors
from ezdxf.enums import TextEntityAlignment

# 讀取數據文件
def rd_mem(shape):
    fnames = ['lat', 'lon', 'data']
    d = []
    for f in fnames:
        filename = f + '.dat'
        d.append(np.memmap(filename, dtype='float32', mode='r', shape=shape))
    return d

# 生成(swLL, neLL)範圍的等高線圖
def dxf(swLL, neLL):

    # 讀取GeoTiff檔案的基本參數、用在產生網格點的TWD97座標
    with open('params.txt', 'r') as f:
        line = [i.strip('\n') for i in f][0]
    x0, y0, nx, ny, dx, dy = (float(i) for i in line.split())
    nx, ny = int(nx), int(ny)
    shape = (ny, nx)

    # 讀取全島緯度、精度、以及高程等3項矩陣
    lat, lon, data = rd_mem(shape)

    # 負值歸0，避免拉大最低、最高值的區間
    data = np.where(data < 0, 0, data)

    # 利用經緯度進行範圍切割：共有4個條件都必須符合。
    idx = np.where((lat >= swLL[0]) & (lat <= neLL[0]) & (lon >= swLL[1]) & (lon
<= neLL[1]))

    # 'LL not right!'是關鍵字，app.py中會檢核。
    if len(idx[0]) == 0:
        return 'LL not right!', list(swLL) + list(neLL)
```

```
    # TWD97座標值：
    ## 1維
    x = [x0 + dx * i for i in range(nx)]
    y = [y0 + dy * i for i in range(ny)]
    ## 2維（y,x）
    xg, yg = np.meshgrid(x, y)

    # 取4個方向的TWD97座標極值，用於標定位置
    bounds = [np.min(xg[idx[0], idx[1]]), np.max(xg[idx[0], idx[1]]),
np.min(yg[idx[0], idx[1]]), np.max(yg[idx[0], idx[1]])]

    # 起訖點的索引
    ib = [x.index(bounds[0]), x.index(bounds[1]), y.index(bounds[2]),
y.index(bounds[3])]

    # 範圍內地形的極值、用於計算間距(固定間距5M或1M)
    cmin = data[ib[2]:ib[3]+1, ib[0]:ib[1]+1].min()
    cmax = data[ib[2]:ib[3]+1, ib[0]:ib[1]+1].max()

    # 開始新的DXF檔案
    doc = ezdxf.new(dxfversion="R2010")
    clrs = {5: colors.RED, 1: colors.BLUE}

    for intv in [5, 1]:
        N = int((cmax - cmin) // intv)
        if N == 0:
            continue
        levels = np.linspace(cmin, cmax, N)

        # 新的圖面(使用Agg避免顯示器干擾)
        fig, ax = plt.subplots()

        ## 繪圖
        cs = ax.contour(x[ib[0]:ib[1]+1], y[ib[2]:ib[3]+1], data[ib[2]:ib[3]+1,
ib[0]:ib[1]+1], levels=levels)
        N = len(cs.collections[:])
        if N == 0:
            continue

        # 新的圖層
        doc.layers.add("TEXTLAYER" + str(intv), color=clrs[intv])
        msp = doc.modelspace()
        for l in range(N):
            lines = cs.collections[l].get_paths()
            for line in lines:
                verts = line.vertices
                xv = verts[:, 0]
                yv = verts[:, 1]
                if len(xv) <= 1:
                    continue
                points = [p for p in zip(xv, yv)]

                # 繪製等高(折)線
```

```
                for i in range(len(points) - 1):
                    msp.add_line(points[i], points[i + 1], dxfattribs={"color":
clrs[intv]})

            # 每條等高線加上數字標籤
            msp.add_text(
                str(levels[l]),
                dxfattribs={
                    "layer": "TEXTLAYER" + str(intv)
                }).set_placement(points[-1], align=TextEntityAlignment.CENTER)

    ## 存檔、輸出到BytesIO供前後端API傳輸
    ran = tf.NamedTemporaryFile().name.replace('/', '').replace('tmp', '')
    fname = 'terr_' + ran + '.dxf'
    output = BytesIO()
    doc.write(output, fmt='bin')
    output.seek(0)  # 重置指針位置
    doc.saveas('./dxfs/' + fname)

    return fname, output
```

這段程式碼在給定的地理界限內從 `.dat` 文件中讀取數據，並生成一個包含等高線圖的 DXF 文件。具體步驟如下：

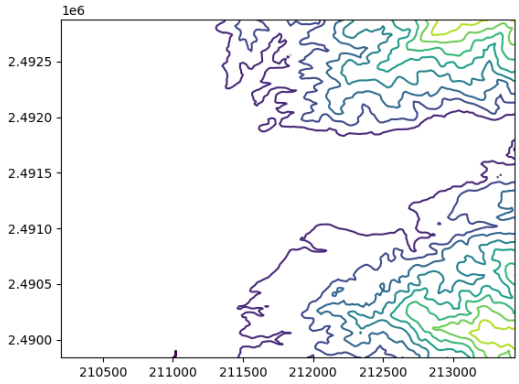
這段代碼的關鍵步驟包括：

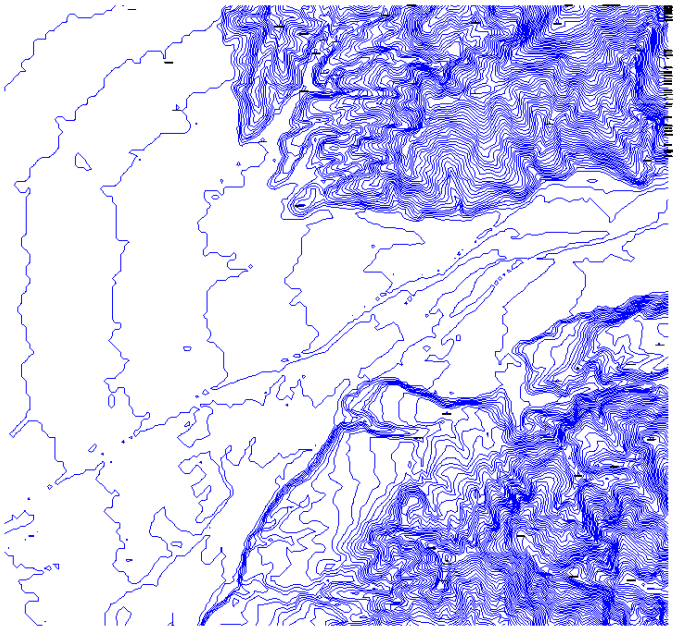
- 讀取地理數據。
- 查找給定界限內的數據。
- 繪製等高線圖
- 生成DXF 文件，其中包含了數據的等高線和對應的標籤，便於在 CAD 軟件中進行查看和分析。

程式下載

- 副程式版本{% include download.html content="mem2dxf.py" %}，供API呼叫、自動回復。

圖檔結果比較





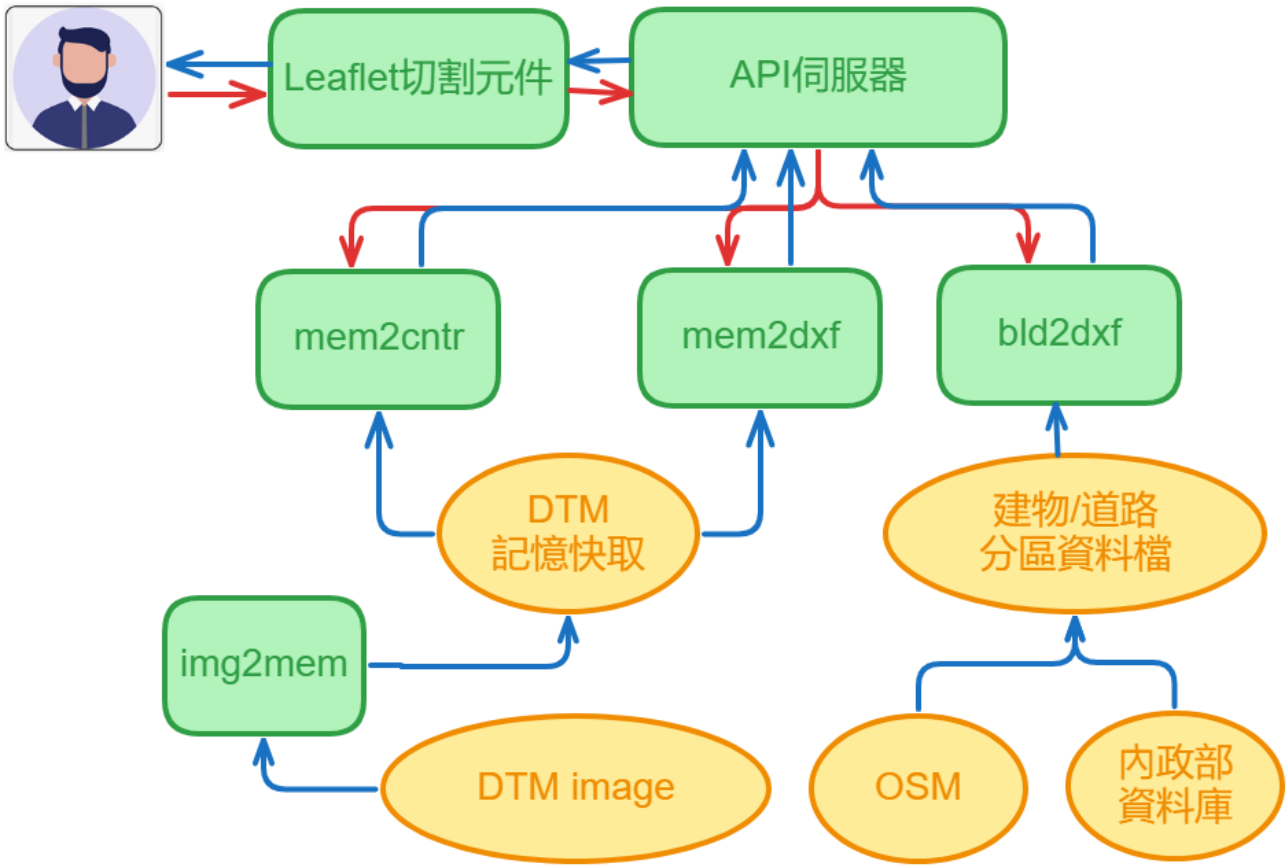
matplotlib等高線圖	dxf檔案

API伺服器的設計

背景

整體流程

- 前台leaflet、API伺服器、製圖函式(cntr及dxf)、資料檔與前處理等作業方式，如圖所示。



- 這支程式(app.py)接收前端html leaflet元件的呼叫、傳入bounds座標值，引用cntr、dxf2支函式進行檔案切割、製圖，最後將結果檔回傳給前端，儲存在使用者本機的下載目錄。

API選擇策略

- GPT、perplexity各大AI都建議使用Flask，不過此處還是補充一下決策的背景。
- node.js：在格隔柵資料處理過程過於繁瑣、不予考慮。
- streamlet
  - python家族之一
  - 有2個模組可以做伺服器，Gunicorn：`st.run(app, host="0.0.0.0", port=8080)`，獨立伺服器：`st.streamlit(app, run_url="streamlit")`
  - 維運html，則使用`st.markdown()`或`st.write()`，列印鑲嵌在程式碼內的html內容。不適用太過複雜的html。
- Flask、Django、Streamlit三者比較如下

框架	特性與能力	優勢	劣勢
Flask	輕量、靈活、微框架	易學易用，非常適合中小型應用	只有基本功能，還需要額外的庫和工具
Django	進階、全端框架	快速開發、健壯、可擴展的架構，非常適合大型、複雜的應用	陡峭的學習曲線，靈活性不如 Flask
Streamlit	用於建立資料應用程式的開源框架	即時資料視覺化，非常適合創建顯示和操作資料的應用程式	不是專門為建立API而設計的

其他方案也詳列如下

框架	特性與能力	優勢	劣勢
FastAPI	現代、高效能Web框架	效能快、記憶體佔用小、基於Python類型提示	仍然比較新，不像Flask和Django那樣廣泛使用
Falcon	輕量級高效能Web框架	快速、有效率、易於使用，非常適合建立RESTful API	微框架，需要額外的程式庫和工具
Pyramid	靈活的開源Web框架	應用廣泛，基於WSGI標準	全端框架，陡峭的學習曲線
Tornado	可擴充、非阻塞的網路框架	處理大量並發連接，非常適合高效能API	微框架，需要額外的函式庫和工具

兩個圖檔並存服務的考量

- `matplotlib.pyplot.contour`是產生dxf圖檔必經(最有效率)的過程，輸出檔案只是附加產品
- 原本序列的關係，因地形數據處理的效率提高了，似乎也沒有必要維持序列，獨立運作也並無不可。

程式說明

這段程式碼建立了一個使用 Flask 框架的 API 服務，其中包含兩個端點 `/api/v1/get_dxf` 和 `/api/v1/get_cntr`。這兩個端點分別用於生成 DXF 文件和 PNG 圖像，並將其返回給客戶端。具體的工作流程如下：

導入必要的庫

- `Flask` 用於創建 web 應用。
- `BytesIO` 用於處理內存中的文件。
- `cntr` 和 `dxf` 分別從 `mem2cntr` 和 `mem2dxf` 模組中導入，用於生成 PNG 圖像和 DXF 文件。

創建 Flask 應用

- `app = Flask(__name__)` 初始化 Flask 應用。

靜態文件的端點

- `serve_html` 函數返回 `index.html` 文件。
- html文件內容詳見index.html的設計

生成 DXF 文件的端點

- `get_dxf` 函數接收 POST 請求中的 JSON 數據，提取西南和東北角的經緯度，
- 並調用 `dxf` 函數生成 DXF 文件（詳mem2dxf.py）。
- 生成的文件以`BytesIO()`附件形式返回。

生成 PNG 圖像的端點

- `get_cntr` 函數接收 POST 請求中的 JSON 數據，提取西南和東北角的經緯度，
- 調用 `cntr` 函數生成 PNG 圖像。

- 生成的圖像也以`BytesIO()`附件形式返回。

主程序入口

- 當程序以腳本形式運行時，啟動 Flask 服務。
- ip及端口在此設定
- 啟動偵錯模型

呼叫方式

- API程式可以直接使用curl指令從後端主機呼叫，可以不需要前端程式，在測試階段可以用`curl`來得到api程式的結果。
- 如以下呼叫`cntr`程式的指令

```
url=http://devp.sinotech-eng.com:5000/api/v1/get_file
curl -X POST $url -H "Content-Type: application/json" -d '{"sw_lat":
22.507545744146224, "sw_lon": 120.61295698396863, "ne_lat": 22.535073497331727,
"ne_lon": 120.64468000957278}' --output a.png
```

- 為串連前端程式，一般使用js指令`fetch`來連接API伺服器，詳見前端網頁的設計#_save方法

代碼說明

以下是完整的代碼：

```
from flask import Flask, request, jsonify, send_file, send_from_directory
import pandas as pd
from io import BytesIO
from mem2cntr import cntr, rd_mem
from mem2dxf import dxf

app = Flask(__name__)

# 靜態文件的端點
@app.route('/')
def serve_html():
    return send_from_directory('.', 'index.html')

# 生成 DXF 文件的端點
@app.route('/api/v1/get_dxf', methods=['POST'])
def get_dxf():
    try:
        data = request.json
        print("Received data:", data)  # 調試輸出
        sw_lat = data.get('sw_lat')
        sw_lon = data.get('sw_lon')
        ne_lat = data.get('ne_lat')
        ne_lon = data.get('ne_lon')
```

```
        # 檢查是否傳入了所有必要的數據
        if not all([sw_lat, sw_lon, ne_lat, ne_lon]):
            return jsonify({"error": "缺少必要的經緯度參數"}), 400

        # 將 DataFrame 寫入內存中的 CSV 文件
        fname, output = dxf((sw_lat, sw_lon), (ne_lat, ne_lon))
        if fname == 'LL not right!':
            return jsonify({"error": "經緯度參數超過範圍"}), 400

        return send_file(output, mimetype='application/dxf', download_name=fname,
as_attachment=True)

    except KeyError as e:
        return jsonify({"error": f"缺少必要的字段: {str(e)}"}), 400
    except Exception as e:
        print("Exception occurred:", str(e))  # 調試輸出
        return jsonify({"error": str(e)}), 500

@app.route('/api/v1/get_cntr', methods=['POST'])
def get_cntr():
    try:
        data = request.json
        print("Received data:", data)  # 調試輸出
        sw_lat = data.get('sw_lat')
        sw_lon = data.get('sw_lon')
        ne_lat = data.get('ne_lat')
        ne_lon = data.get('ne_lon')

        # 檢查是否傳入了所有必要的數據
        if not all([sw_lat, sw_lon, ne_lat, ne_lon]):
            return jsonify({"error": "缺少必要的經緯度參數"}), 400

        # 將 DataFrame 寫入內存中的 CSV 文件
        fname, output = cntr((sw_lat, sw_lon), (ne_lat, ne_lon))
        if fname == 'LL not right!':
            return jsonify({"error": "經緯度參數超過範圍"}), 400

        return send_file(output, mimetype='image/png', download_name=fname,
as_attachment=True)

    except KeyError as e:
        return jsonify({"error": f"缺少必要的字段: {str(e)}"}), 400
    except Exception as e:
        print("Exception occurred:", str(e))  # 調試輸出
        return jsonify({"error": str(e)}), 500

# 主程式：開啟偵錯、IP、端口
if __name__ == '__main__':
    print("Starting Flask server...")
    app.run(debug=True, host='devp.sinotech-eng.com', port=5000)
```

此應用程序提供了一個簡單的網頁界面，並且可以通過 API 調用來生成並下載 DXF 文件和 PNG 圖像。確保 `mem2cntr` 和 `mem2dxf` 模組正確地被導入並且運行正常。

運轉維護

app.py紀錄

- app.py增添下列指令

```
import logging
from datetime import datetime

app = Flask(__name__)

# 設定日誌配置
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s %(levelname)s: %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S',
    handlers=[
        logging.FileHandler('user_activity.log'),
        logging.StreamHandler()
    ]
)

@app.route('/log_console_output', methods=['POST'])
def log_console_output():
    # 獲取前端傳來的console輸出
    console_output = request.get_json().get('console_output')
    logging.info(f"Console output: {console_output}")
    error_output = request.get_json().get('errort')
    logging.info(f"Error output: {error_output}")
    bound_output = request.get_json().get('Saved bounds')
    logging.info(f"Bound output: {bound_output}")

    return jsonify({'status': 'success'})
...
        print("Received data:", data)  # 调试输出
        sw_lat = data.get('sw_lat')
        sw_lon = data.get('sw_lon')
        ne_lat = data.get('ne_lat')
        ne_lon = data.get('ne_lon')


        with open('user_activity.log', 'a') as log_file:
            log_file.write(f"Received data: {data}\n")
```

index.html啟動紀錄

```
...
 <script>
window.addEventListener('error', function(event) {
  // 獲取錯誤資訊
  var errorMessage = event.message;
  var errorStack = event.error.stack;

  // 使用AJAX將錯誤資訊傳送到Flask後端
  fetch('/log_console_output', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      console_output: `Error: ${errorMessage}\nStack: ${errorStack}`
    })
  })
  .then(response => {
    console.log('Console output logged successfully');
  })
  .catch(error => {
    console.error('Error logging console output:', error);
  });
});
 </script>
...
```

定期檢查與啟動

- 上班日每天7時進行檢查，如果沒有運轉，則予以啟動
- 如果前一天有人執行，則將紀錄檔案重新更名。

```
$ crontab -l|grep ck_up
0  7  *  *  1-5 /nas2/kuang/MyPrograms/CADNA-A/ck_up.cs >& /dev/null 2>&1

$ cat ck_up.cs
#!/bin/bash
#default running at DEVP.sinotech-eng.com:5000
cd /nas2/kuang/MyPrograms/CADNA-A
n=$(ps -ef|grep app.py|wc -l)
if [ $n -lt 3 ];then
  ~/.conda/envs/pyn_env/bin/python app.py
  echo excuted
fi

if [ -e "user_activity.log" ];then mv user_activity.log user_activity.log.$(date -
d "yesterday" +%Y%m%d);fi
```

前端網頁的設計

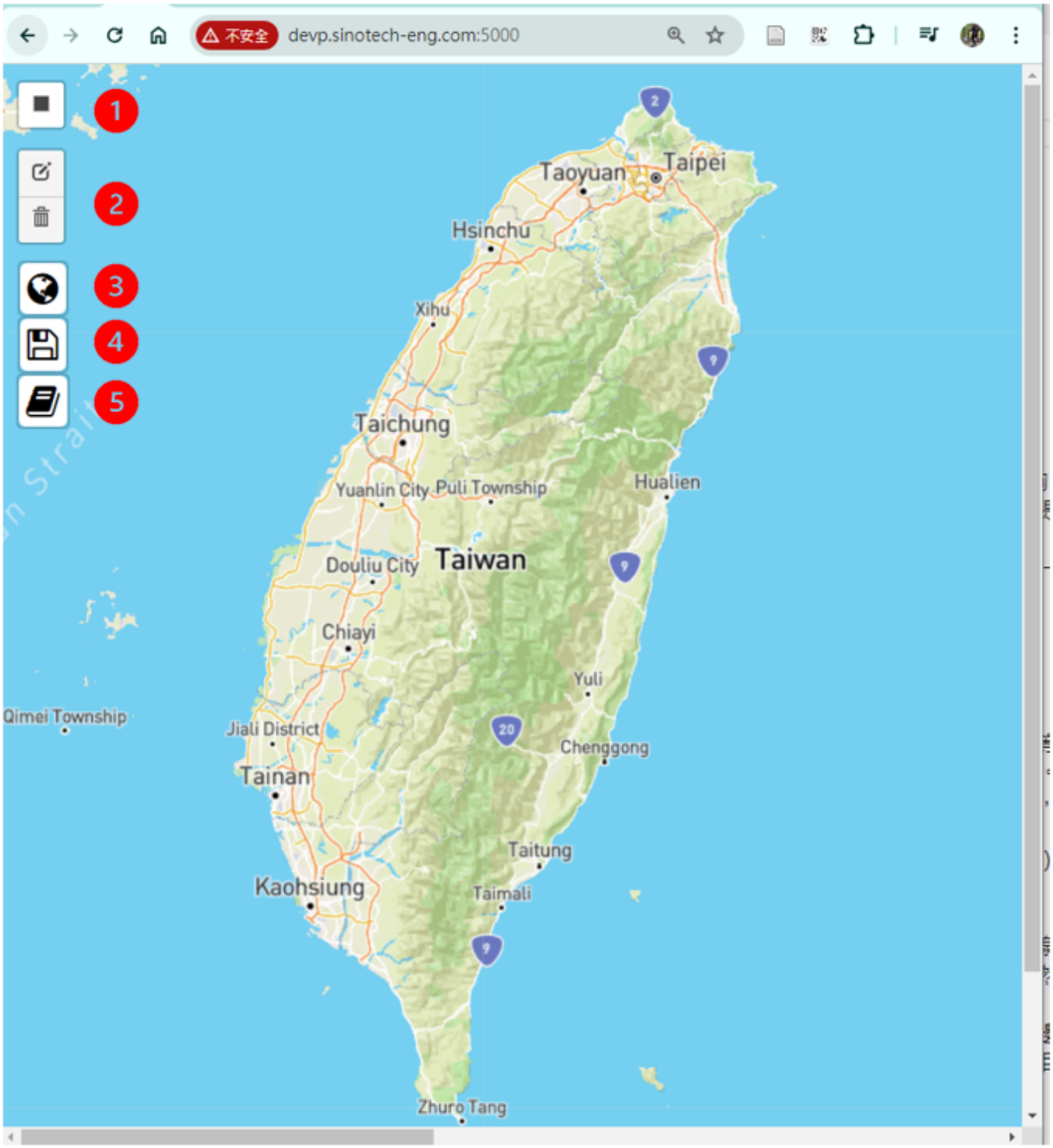
背景

- 整個專案的靈魂在於這個地圖介面，高品質、高效率的介面將會是成敗的關鍵。
- 圖框的選取不是難事，API回應結果該如何命名？是讓使用者另存、還是直接下載？還需要仔細琢磨一下。
- 專案的整體架構可以參考這裡

目的

這個前端網頁的功能基本上是個地圖篩選器，藉由矩形圖框界定選取範圍、點選儲存icon來觸發Flask API程式，將圖檔返回儲存到使用者的下載區。

html圖面如下，內容詳index.html，實例請造訪devp.sinotech-eng.com:5000。



大略說明

這段 HTML 和 JavaScript 代碼創建了一個基於 Leaflet 的地圖應用，並且提供了五項交互功能，包括繪製矩形切割範圍、保存範圍內的數據作為 PNG 或 DXF 文件，以及打開作業手冊。以下是每個部分的簡要說明：

1. **HTML 基本結構：**
    - `<!DOCTYPE html>` 宣告文件類型。
    - `<html>` 和 `<head>` 包含標題和基本樣式設置。
    - 使用 `<meta>` 標籤設置兼容模式。
    - 引入 Leaflet、Leaflet Draw 和 Font Awesome 的樣式和腳本。
2. **樣式設置：**

    - 設置基本的樣式，包括 `.style1` 和 `.with-background`。

3. **地圖初始化：**
    - 使用 Leaflet 創建一個地圖，並設置初始視圖和圖層。
4. **繪圖控制：**
    - 初始化一個 `FeatureGroup` 來保存繪製的圖形。
    - 創建一個 `DrawControl` 來啟用繪製矩形的工具。
5. **自定義控制按鈕：**
    - 創建三個自定義按鈕，分別用於保存 PNG 文件、保存 DXF 文件和打開作業手冊。
    - 使用 `L.Control.extend` 創建自定義控制，並添加到地圖上。
6. **事件處理：**
    - 當繪製新的矩形時，清除之前的圖層，並將新的矩形添加到地圖上。
    - 當矩形繪製完成後，自動放大地圖以適應選取框範圍。

Leaflet js設計

,

底圖的考量

- 很多leaflet內設圖磚連結開放政策修改，因此底圖的選用還是選用穩定性高的提供者為宜。
- 由於等高線是專案關切主題、而不是其他地面資訊，因此不考慮內政部圖磚、openStreetMap等，以避免等高線被文字遮蔽。
- 此處選擇Mapbox的圖磚，較、openTopoMap等來得清晰簡潔、遮蔽較少，有較/高的地圖品質（雖然尚未全面中文化）。
- mapbox API token
    - 登記
    - 費用機制：瀏覽、低量路線計算並不收費、高量路線計算才會收費。
- 改用其他圖磚
    - 更改連結及相應的token即可

其他地圖設定

- 中心點及初始縮放比例
- `zoomControl: false`，不顯示左上方默認的縮放控制按鈕("+ - ")。縮放功能都在滾輪上實現，使用按鈕還要移動滑鼠到左上方，又與圖框範圍選擇的功能衝突，因此將其取消。

```
var map = L.map('map',{ zoomControl: false }).setView([23.5, 121.],8);
```

- 特色組工具（FeatureGroup
    - 這個工具將會一次定義圖層內的物件
    - 下列指令將會新增`DrawItems`群組，以備後面程式可以批次定義其內容。

```
var drawnItems = new L.FeatureGroup();
map.addLayer(drawnItems);
```

- 繪圖控制(L.control)
  - 控制物件的新增、編輯與刪除。共有2各項目需定義，edit將連到前述特色組圖層，draw則規範要新增元件的顏色、圖框及填滿等特性。
  - 矩形物件是我們唯一所需要的圖形物件，其他先關閉。
  - 選取範圍之填滿、即使增加透明度，仍會造成視覺上的干擾。因此最後決定取消填滿、改以紅色實現圖框。

```
var drawControl = new L.Control.Draw({
   edit: {
      featureGroup: drawnItems
         },
   draw: {
         polyline: false,
         polygon: false,
         circle: false,
         marker: false,
         circlemarker: false,
         rectangle: {
            shapeOptions: {
               color: 'red',
               fillOpacity: 0.0      // 填充透明度，0 表示完全透明
               }
            }
         }
});
map.addControl(drawControl);
```

API程式之觸發

- saveButtonP/X是2個相似度很高的自訂 Leaflet 控制項，目的在儲存png、dxf2種格式的等高線圖檔，
- 因為相似度很高，此處只須說明一項即可。
- 傳送數據檔案等功能，以一個變數型態實現，這個變數也會生成圖面上的按鈕。
- 首先釐清按鈕的設定，再定義點擊之後觸發的事件(啟動API程式)

按鈕的位置、外觀與觸發

這段程式碼定義了一個自訂的 Leaflet 控制項，這個控制項的設計是在地圖的左上角添加一個按鈕，當按下時，會將地圖儲存為 PNG 圖像。

```
var saveButtonP = L.Control.extend({
   options: {
         position: 'topleft'
   },
   onAdd: function(map) {
```

```
        var container = L.DomUtil.create('div', 'leaflet-bar leaflet-control
leaflet-control-custom');
            container.innerHTML = '<i class="fa fa-globe fa-2x with-background"
title="儲存PNG檔案"></i>';
            L.DomEvent.on(container, 'click', this._save, this);
            return container;
    },
```

以下是程式碼的分解：

1. `var saveButtonP = L.Control.extend({ ... });`:
    - 這一行定義了一個新的控制項類別 `saveButtonP`，它繼承自 Leaflet 提供的 `L.Control` 類別。
    - `extend` 方法允許您創建一個新類別，該類別繼承自父類別的屬性和方法。
2. `options: { position: 'topleft' }`:
    - 這定義了控制項的預設選項。在這種情況下，控制項將被放置在地圖的左上角。
3. `onAdd: function(map) { ... }`:
    - 當控制項被添加到地圖時，會調用這個方法。它負責創建控制項的 UI 元素。
    - `map` 參數是控制項被添加到的 Leaflet 地圖對象。
4. `var container = L.DomUtil.create('div', 'leaflet-bar leaflet-control leaflet-control-custom');`:
    - 這一行創建了一個新的 HTML `div` 元素，它將作為控制項 UI 的容器。
    - 使用 `L.DomUtil.create` 方法創建元素，並將類名 `leaflet-bar`、`leaflet-control` 和 `leaflet-control-custom` 添加到元素中以對其進行樣式設置。
    - 樣式`leaflet-control-custom`是新增元件之集結，會在下面進一步設定。
5. `container.innerHTML = '<i class="fa fa-globe fa-2x with-background" title="儲存PNG檔案"></i>';`:
    - 這一行將容器的內部 HTML 設置為一個 HTML 元素，其中包括一個 Font Awesome 圖標 (fa-globe) 和一個標題，翻譯為"儲存 PNG 檔案"。
6. `L.DomEvent.on(container, 'click', this._save, this);`:
    - 這一行為容器添加了一個事件監聽器，監聽點擊事件。當容器被點擊時，將調用 `_save` 方法。
    - `this` 關鍵字指的是 `saveButtonP` 控制項對象。
7. `return container;`:
    - 這一行返回容器元素，這是將被添加到地圖的 UI 元素。

總之，這段程式碼定義了一個自訂的控制項，在 Leaflet 地圖的左上角添加一個按鈕。當按鈕被點擊時，它會將地圖儲存為 PNG 圖像。

_save方法

```
    _save: function(e) {
          L.DomEvent.stopPropagation(e);
          L.DomEvent.preventDefault(e);
          if (currentRectangle) {
             var bounds = currentRectangle.getBounds();
             var result = {
                northEast: bounds.getNorthEast(),
                southWest: bounds.getSouthWest()
             };
             console.log("Saved bounds:", result);
             //alert("Bounds saved! Check the console for details.");
             // 在此處添加將結果保存到後端或本地存儲的代碼

             fetch('/api/v1/get_cntr', {
                method: 'POST',
                headers: {
                      'Content-Type': 'application/json'
                },
                body: JSON.stringify({
                      sw_lat: result.southWest.lat,
                      sw_lon: result.southWest.lng,
                      ne_lat: result.northEast.lat,
                      ne_lon: result.northEast.lng
                })
          }).then(response => {
             if (!response.ok) {
                throw new Error('Network response was not ok');
             }
             var contentDisposition = response.headers.get('content-disposition');
             var filename = contentDisposition.split('filename=')[1].trim();
             return response.blob().then(blob => {
                // 创建一个链接，下载返回的文件
                var url = window.URL.createObjectURL(blob);
                var a = document.createElement('a');
                a.href = url;

                // 设置下载的文件名
                a.download = filename || 'download'; // 使用 API 返回的文件名，如果没
有则使用默认值
                document.body.appendChild(a);
                a.click();
                window.URL.revokeObjectURL(url); // 释放 URL 对象
                //a.remove();
             });
          }).catch(error => console.error('Error:', error));
    }
}
```

這段程式碼定義了一個名為 `_save` 的方法，它是 `saveButtonP` 控制項的一部分。這個方法在點擊控制項按鈕時被調用，並將地圖的範圍信息儲存到後端或本地存儲中。

以下是方法的內容：

1. `L.DomEvent.stopPropagation(e);` 和 `L.DomEvent.preventDefault(e);`:
    - 這兩個方法用於防止其他事件繼續發生、和預防預設行為，以確保點擊控制項按鈕時不會觸發其他不預期的事件。
2. `if (currentRectangle) { ... }`:
    - 這個條件判斷是否已經選擇了地圖的範圍（`currentRectangle`）。如果已經選擇了範圍，則執行以下代碼。
    - 如果沒有這個記憶並且予以消除，每次產生的圖框將會一直累積下去。
3. `var bounds = currentRectangle.getBounds();`:
    - 這行獲取選擇的範圍的緯度和經度。
    - 這個變數將會被傳送到API程式。
4. `var result = { ... };`:
    - 這個變數將儲存選擇的範圍的緯度和經度。
5. `fetch('/api/v1/get_cntr', { ... });`:
    - 這個方法使用 Fetch API 將範圍信息發送到後端 API `/api/v1/get_cntr`,
    - cntr接收儲存範圍信息之後，將會切割出售範圍內的DTM數據、製作等高線圖，再將其回傳。
6. `response.blob().then(blob => { ... });`:
    - 這個方法將後端返回的資料轉換為 Blob 物件，然後使用這個 Blob 物件創建一個 URL 連結，讓用戶可以下載儲存的範圍信息。
7. `document.body.appendChild(a); a.click(); window.URL.revokeObjectURL(url);`:
    - 這些代碼將創建的連結添加到文檔體中，然後點擊連結以下載範圍信息，最後釋放 URL 物件。
8. `catch(error => console.error('Error:', error));`:
    - 這個 catch block 會捕捉任何發生的錯誤，並將錯誤信息輸出到控制台。

總之，這個方法將地圖的範圍信息儲存到後端或本地存儲中，並將儲存的資料下載到用戶的電腦上。

---

ezdxf的座標系統

背景

- 使用手冊
    - object coordinate system (OCS)
    - User coordinate system (UCS)
- Tutorial

item	num

item	num
`[e for e in doc.modelspace()]`	3858
`[e for e in msp if e.dxf.layer=='98112']`	301
`len([e for e in msp if e.dxf.layer=='98112' and e.dxftype()=='POLYLINE'])`	277(其餘為`e.dxftype()`為`'TEXT'`)
`set([i.plain_text() for i in txt.plain_text()`	26~97共17個不連續值
`set([i.plain_text() for i in [e for e in msp if e.dxf.layer=='98111' and e.dxftype()=='TEXT']])`	20~95每隔5之整數字元、分散在65條 `POLYLINE`
`[i for i in e.points()][:5]`	`[Vec3(313858.7020155214, 2771736.841981305, 86.0),...]` ('TEXT'沒有`points()`)
`e.dxf.elevation`	`Vec3(0.0, 0.0, 86.0)`
`e.get_mode()`	`'AcDb2dPolyline'`

WCS to OCS

```
def wcs_to_ocs(point):
    px, py, pz = Vec3(point)  # point in WCS
    x = px * Ax.x + py * Ax.y + pz * Ax.z
    y = px * Ay.x + py * Ay.y + pz * Ay.z
    z = px * Az.x + py * Az.y + pz * Az.z
    return Vec3(x, y, z)
```

OCS to WCS

```
Wx = wcs_to_ocs((1, 0, 0))
Wy = wcs_to_ocs((0, 1, 0))
Wz = wcs_to_ocs((0, 0, 1))

def ocs_to_wcs(point):
    px, py, pz = Vec3(point)  # point in OCS
    x = px * Wx.x + py * Wx.y + pz * Wx.z
    y = px * Wy.x + py * Wy.y + pz * Wy.z
    z = px * Wz.x + py * Wz.y + pz * Wz.z
    return Vec3(x, y, z)
```

e.ocs()

```
['from_wcs',
 'points_from_wcs',
```

```
 'points_to_wcs',
 'render_axis',
 'to_wcs',
 'transform',
 'ux',
 'uy',
 'uz']
```

Tutorial for OCS/UCS Usage

Placing 2D Circle in 3D Space

```
import ezdxf
from ezdxf.math import OCS

doc = ezdxf.new('R2010')
msp = doc.modelspace()

# For this example the OCS is rotated around x-axis about 45 degree
# OCS z-axis: x=0, y=1, z=1
# extrusion vector must not normalized here
ocs = OCS((0, 1, 1))
msp.add_circle(
    # You can place the 2D circle in 3D space
    # but you have to convert WCS into OCS
    center=ocs.from_wcs((0, 2, 2)),
    # center in OCS: (0.0, 0.0, 2.82842712474619)
    radius=1,
    dxfattribs={
        # here the extrusion vector should be normalized,
        # which is granted by using the ocs.uz
        'extrusion': ocs.uz,
        'color': 1,
    })
# mark center point of circle in WCS
msp.add_point((0, 2, 2), dxfattribs={'color': 1})
```

3D Polyline

- Using UCS to Place 3D Polyline

```
# using an UCS simplifies 3D operations, but UCS definition can happen later
# calculating corner points in local (UCS) coordinates without Vec3 class
angle = math.radians(360 / 5)
corners_ucs = [(math.cos(angle * n), math.sin(angle * n), 0) for n in range(5)]

# let's do some transformations by UCS
transformation_ucs = UCS().rotate_local_z(math.radians(15))  # 1. rotation around
z-axis
```

```
transformation_ucs.shift((0, .333, .333))  # 2. translation (inplace)
corners_ucs = list(transformation_ucs.points_to_wcs(corners_ucs))

location_ucs = UCS(origin=(0, 2, 2)).rotate_local_x(math.radians(-45))
msp.add_polyline3d(
    points=corners_ucs,
    close=True,
    dxfattribs={
        'color': 1,
    }
).transform(location_ucs.matrix)

# Add lines from the center of the POLYLINE to the corners
center_ucs = transformation_ucs.to_wcs((0, 0, 0))
for corner in corners_ucs:
    msp.add_line(
        center_ucs, corner, dxfattribs={'color': 1}
    ).transform(location_ucs.matrix)
```

add_line

```
add_line(start: UVec, end: UVec, dxfattribs=None)→ Line¶
Add a Line entity from start to end.

Parameters:
start – 2D/3D point in WCS
end – 2D/3D point in WCS
dxfattribs – additional DXF attributes
```

```
points=[p for p in zip(xv,yv)]
M=len(points)
for i in range(M-1):
  msp.add_line(points[i], points[i+1], dxfattribs={"color": clrs[intv]})
```

add_polyline2d

```
add_polyline2d(points: Iterable[UVec], format: str | None = None, *, close: bool =
False, dxfattribs=None)→ Polyline¶
Add a 2D Polyline entity.

Parameters:
points – iterable of 2D points in WCS(OCS)
close – True for a closed polyline
format – user defined point format like add_lwpolyline(), default is None
dxfattribs – additional DXF attributes
```

add_polyline3d

3d points in wcs(World coordinate system)

```
add_polyline3d(points: Iterable[UVec], *, close: bool = False, dxfattribs=None)→
Polyline¶
Add a 3D Polyline entity.

Parameters:
points – iterable of 3D points in WCS
close – True for a closed polyline
dxfattribs – additional DXF attributes
```

polyline.points()

```
points()→ Iterator[Vec3]¶
Returns iterable of all polyline vertices as (x, y, z) tuples, not as Vertex
objects.
```

Tutorial for the Geo Add-on

- Tutorial for the Geo Add-on
- Geo Interface

313600.000, 2771200.000 25.0476184564796,121.630312241047 314600,2772200 25.0566038304452, 121.640269221663,