タイルマップサービスを作ってみた

2017 年 4 月 9 日

技術書典 2

Google Map 的な Web 地図サービスの開発

はじめに

さすがに技術書典でこんな本に興味を示してくれた人が、Google Map を知らないということはないでしょう。この本は Google Map のようにぐりぐり動く、いわゆる Slippy Map を使った Web サービスを開発する方法が書かれています。すでに地形図や道路地図は、Google Map だけでなく、地理院地図や OpenStreetMap があります。今回は、この地理院地図や OpenStreetMap を基本図として、その上に GSMaP と呼ばれる降水量データを重ねた情報を提供するサービスを開発します。

GSMaP は、JAXA、NOAA、EUMETSAT が運用している気象衛星のデータを準リアルタイム（といっても 4 時間前のデータですが）の数値データが 1 時間毎に提供されています。JAXA に利用申請をすると、無料で使用することができます。この数値データをまず画像データとして可視化し、その画像に位置情報を付加し、タイルマップを生成していきます。気象データが相手ですから、毎時自動的にデータをアップデートする必要があります。そこで、QGIS のような GUI のプログラムではなく、できるだけコマンドラインだけで作成していきます。

この本で作成する Web アプリは、Debian/GNU Linux 上で動作する事を前提にしています。ですので、Linux の基本的な操作ができる人を対象にしています。また、Web サービスを開発するだけの基礎的な素養がある事を前提にしています。nginx の設定ファイルや Python やシェルスクリプト、C++ 等のソースコードが出てきます。適宜解説に努めますがが、十中八九説明が足りないので、必要に応じてリファレンス等を参照すると良いでしょう。

この本に出てくるソースコードは、GitHub で公開しています。また電子版も GitHub から入手できます。間違い等ありましたら、ISSUE を投げていただければ対応できると思います。リポジトリの URL は以下の通りです。

`https://github.com/chomy/DevelopmentOfTMS`

タイルマップサービス (TMS) とは

GoogleMaps のように、表示する位置や縮尺を自由に変えられる地図を slippy map といいます。日本語では「スクロール地図」でしょうか。さて、この slippy map を作って欲しいと依頼された時、あなたはどのように実装しますか？

まず地図の情報が必要です。それは画像である必要があります。その画像を切り出してブラウザに表示する... ざっくりそんな感じになるでしょうか。まさにこれを実装したのが、タイルマップサービス (Tile Map Service) TMS とよばれるものです。TMS は、正確に言うと Open Source Geospatial Fundation (OSGeo 財団) によって開発されたタイルの仕様で、オープンソースなプロダクトによく使われています。同様の機能を実現した、WMTS (Web Map Tile Service) というものもあります。

タイル

一枚の大きな世界地図を、小片に分割して必要なものだけ送ることで、ネットワークの帯域やメモリの消費を抑えることができます。この分割された地図の小片を「タイル」と呼びます。TMS では、タイルは一辺が 256 ピクセルの正方形の PNG 画像です。次にタイルの切り出し方を考えます。まず、世界地図を 1 枚のタイルで表現することができます。この縮尺をズームレベル 0 と呼びます。ズームレベルは、正の整数値で、1 増える毎に表示する面積が 1/4 になります。すなわち、ズームレベル 1 では、縦 2 枚、横 2 枚の 4 枚のタイルで全世界を表現します。同様にズームレベル 2 では、16 枚になります。これを繰り返していくと、図 2 のようなピラミッド構造になります。

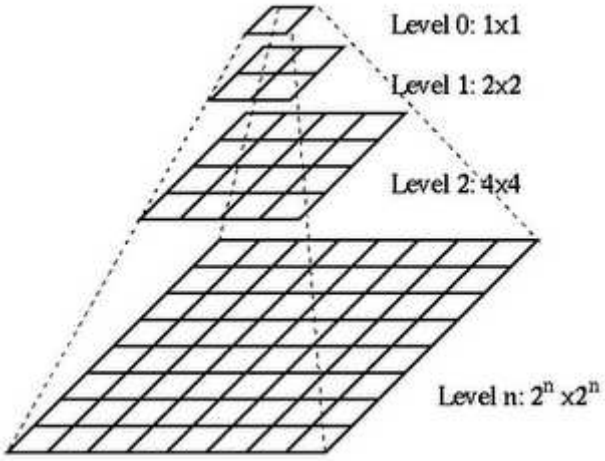

図 1　タイルのピラミッド構造 [1]

このタイルの位置は、x、y の座標で、OpenStreetMap では、緯度経度との関係は以下の通りです [2]。

$$x = \frac{2^z(lon + 180)}{360}$$

$$y = 2^{z-1}(1 - \frac{\ln(\tan(\frac{lat\pi}{180}) + \sec(\frac{lat\pi}{180}))}{\pi})$$

ここで、*lon*、*lat* は、それぞれ度単位の経度、緯度、*z* はズームレベルです。ちなみに xy の原点は... って聞かれると、TMS は左下で、WMTS は右上で、OpenStreetMap は、Y flipped TMS とやらで、左上だったり、結構闇だったりします。上の式は OpenStreetMap のものなので、TMS とは異なるので注意が必要です。

タイルの配布

さて、タイルのことはわかってきました。それでは、どのようにタイルを取得するすれば良いのでしょう。TMS はブラウザに地図を表示するためのものですので、もちろん Web サーバから HTTP で取得します。URL は、以下のようになります。

```
http://tms.example.org/1.0.0/name/z/x/y.png
```

x、y はタイルの座標で、z はズームレベル、1.0.0 というのは TMS のバージョンです。name はタイルの名前です。例えば、地理院地図の 50 万分の 1 のタイルの URL は

```
http://cyberjapandata.gsi.go.jp/xyz/std/z/x/y.png
```

となります。実際のタイルサーバでは、TMS のバージョンがなかったり、結構自由です。正確には地理院地図は TMS と名乗っていません。ただ、TMS も WMTS も地理院地図も OpenStreetMap も、URL で XYZ を指定してタイルを取得するのでこれらを総称して XYZ タイルや XYZ レイヤなどと呼ばれています。仕様はがっちり決められていても、実装が追いつかないのはよくある話ですね。

この本でも、がっちり TMS を実装するのではなく、ざっくり XYZ タイルサービスを作成することにしましょう。

タイルの作成

では、タイルを作成しましょう。元になる、測地系といった地理空間情報が付いた画像があれば簡単です。例えば、world.tiff という、GeoTiff ファイルがあったとしましょう。GeoTiff ファイルは、地理空間情報のタグが付加された Tiff 画像です。地理空間情報が付いたフォーマットに Shape 等もありますが、21 世紀なので GeoTiff で良いでしょう。タイルの作成には、GDAL に入っている gdal2tiles.py を使います。GDAL は、”ぐーだる” とか” ぐだーる” と呼ばれている、Geo な世界のスイスアーミーナイフ的な存在で、本来は C/C++ のライブラリなのですが、付属するプログラムが優秀でツール群と誤解されているものです。後で、プログラミングライブラリとして使用しますが、タイルの生成には、GDAL をツール群として使います。GDAL のインストールは面倒なイメージがあるので、Linux のパッケージや OSGeoLive といったディストロを使うと良いでしょう。特に OSGeoLive は Geo な事をやる時に必要なツールはほとんど入っているので、特におすすめです。

タイルの作り方ですが、gdal2tiles.py に GeoTiff ファイル名を指定するだけです。生成するタイルのズームレベルの範囲を指定することもできます。たとえば、map.tiff から、ズームレベル 0 から 8 までのタイルを生成するには、以下のコマンドを実行します。

```
$ gdal2tiles.py -z 0-8 map.tiff
```

タイルの生成には、大量の画像を生成するためものすごーく時間がかかります。ちなみにズームレベル 0 から 8 までのタイルの総数は 87,381 枚です。

さて次章では、タイルを生成する元ととなる、GeoTiff ファイルを作成します。

GeoTiff ファイルの作成

この章では、タイルの元になる GeoTiff ファイルを作成します。地図を作るのは（ゼンリンさんから情報を買ったり、OSM の XML から描画したり）非常に大変ですので、今回は地球観測衛星の観測データを可視化して、OpenStreetMap や地理院地図にオーバーレイするためのタイルを作成しましょう。

今回対象とする衛星は、JAXA の GCOM-W1「しずく」です。しずくは、地球の水に関するデータを観測するための衛星で、AMSER2 というセンサーを搭載しています。このセンサーは、海水面温度や地上水分量、降水量等を観測しており、観測値を JAXA が数値データを公開しています。データは JAXA の GCOM-W データ提供サービス [3] の Web ページでユーザ登録後ダウンロードすることができます。ユーザ登録にハードルが高そうですが、中の人の話の又聞きによると、最近 JAXA は社会貢献活動を求められているらしく、データを実業界で使って欲しいということですので申請すると、びっくりするほどあっさり承認されます。

データの可視化

今回は、海水面温度、Sea Surface Temperature (SST) を可視化してタイルを作成します。まず数値データをダウンロードします。詳細は、データ提供サイトのドキュメントを参照するとして、SST_10 のディレクトリから gzip 圧縮された拡張子が h5 のファイルをダウンロードしましょう。h5 というファイルは、HDF5 というフォーマットのファイルで、階層構造を持った複数のデータを格納できるフォーマットです。観測した数値データだけでなく、観測日時、センサーや解析に関する情報等も 1 つのファイルに格納できるすぐれものです。仕様も公開されており、様々なプログラミング言語のライブラリが公開されています。

ではまず数値データを可視化しましょう。h5 ファイルの、Geophysical_data という項に、1800x3600 の 2 次元データが入っています。各値は、緯度、経度 0.1 度毎のメッシュになっています。この一つ一つの数値を色に変換して縦 1800 ピクセル、横 3600 ピクセルの画像にします。簡単ですね。

値と色の対応は、実際に科学で使用されている配色を使いましょう。私は octave という MATLAB クローンを使って作成しました。jet という関数があるのでそれを使って jet.txt という RGB とアルファチャンネルを 0 から 255 までの値と対応付けます。生成方法は、後日追記します。コミケまでに... 技術書典で購入された方には、前述の github ページで PDF を公開します。夏コミ後に見に行ってみてください。

ダウンロードした HDF5 ファイルから、GeoTiff へ変換する Python スクリプトを listing 1 に示しています。GitHub でデジタルデータを公開しています。可視化部分は、listing 1 の 11 から 35 行目に記述しています。関数 make_colortable で、jet.txt からカラーテーブルを作成します。そして関数 gey_band で、一つ一つの数値を RGB とアルファチャンネルに変換しています。変換した、RGBA のチャンネルを 37 行目からはじまる array_to_raster 関数で GeoTiff を作成しています。46 行目の for ループで各チャンネルを書き込んでいます。51 から 53 行目で測地系の情報を、43 行目で 1 ピクセルあたりの座標の変化量を指定しています。今回のデータは、緯度経度なので、測地系に緯度経度を使用していることを意味する EPSG4326 を指定してい

図 2　タイルのピラミッド構造 [1]

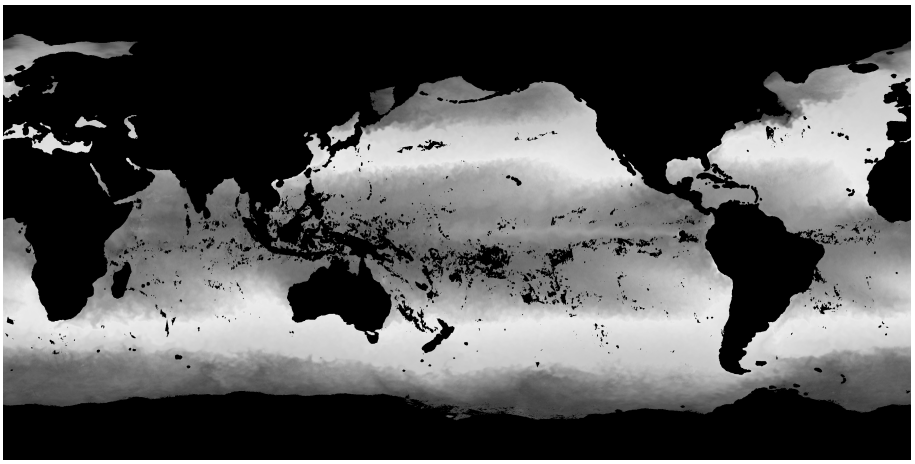

図 3　海水面温度 3 日間平均値

ます。測地系に関しても、夏コミまでに書ければいいなぁ〜

このコードを使って生成した GeoTiff の例を図**??**に示します。印刷物だと色がわからないので、是非 GitHub の PDF をご覧ください。印刷の都合上、欠損値を黒にしています。生成した画像では透明です。低軌道衛星で観測範囲も狭いので、欠損値が多いですね。この衛星は 3 日で全球をスキャンできるので、3 日分の観測値を平均した画像を 3 に示します。ちなみに複数のファイル名を指定して、平均をとっているコードは 56 行目から始まる関数に書いてあります。numpy を使ってすべてのピクセルをスキャンせずに演算しており、numpy のパワーを実感できます。ちなみに、カラーテーブルを参照するのに最低 1 回はフルピクセルスキャンが必要なので、私のノート PC(MacbookAir Late2013) で、変換に 40 秒ほどかかります。

ブラウザへの描画

作成した GeoTiff を使ってタイルを作成したら、Leaflet または OpenLayers という JavaScript ライブラリを使って、ブラウザに描画することができます。続きはコミケ or GitHub で。

Appendix

Listing 1　変換プログラムのソースコード

```
#!/usr/bin/python

import gdal
import h5py
import osr
import sys
import numpy as np

expand_data = lambda h5: h5['Geophysical Data'][:,:,0]

def get_band(data, ct):
        index = lambda val: 0 if val<=-32767 else int(val/100.*5.08+51.8+0.5)
        r = np.ndarray((1800,3600), np.byte)
        g = np.ndarray((1800,3600), np.byte)
        b = np.ndarray((1800,3600), np.byte)
        a = np.ndarray((1800,3600), np.byte)
        for y in range(1800):
                for x in range(3600):
                        color = ct.GetColorEntry(index(data[y][x]))
                        r[y][x] = color[0]
                        g[y][x] = color[1]
                        b[y][x] = color[2]
                        a[y][x] = color[3]
        return (r,g,b,a)



def make_colortable(filename):
        ct = gdal.ColorTable(gdal.GPI_RGB)
        with open(filename) as f:
                for line in f:
                        row = map(lambda x:int(x), line[:-1].split())
                        ct.SetColorEntry(row[0],tuple(row[1:]))

        return ct

def array_to_raster(outfile, data, ctfile):
        ct = make_colortable(ctfile)
        band = get_band(data,ct)

        driver = gdal.GetDriverByName('GTiff')
        out = driver.Create(outfile, 3600, 1800, 4, gdal.GDT_Byte)
        out.SetGeoTransform((0.0, 0.1, 0, 90, 0, -0.1))

```

```
45          ci = (gdal.GCI_RedBand, gdal.GCI_GreenBand, gdal.GCI_BlueBand,gdal.GCI_AlphaBand)
46          for i in range(4):
47                  b = out.GetRasterBand(i+1)
48                  b.SetColorInterpretation(ci[i])
49                  b.WriteArray(band[i])
50                  b.FlushCache()
51          srs = osr.SpatialReference()
52          srs.ImportFromEPSG(4326)
53          out.SetProjection(srs.ExportToWkt())
54
55
56  def averaged_data(files):
57          val = np.zeros((1800,3600),np.int32)
58          count = np.zeros((1800,3600),np.int16)
59
60          for f in files:
61                  h5 = h5py.File(f)
62                  data = expand_data(h5)
63                  h5.close()
64
65                  mask = data>-99
66                  tmp = np.zeros((1800,3600),np.int16)
67                  tmp[mask] = 1
68                  count = np.add(count,tmp)
69
70                  tmp = np.copy(data)
71                  tmp[np.logical_not(mask)] = 0.
72                  val = np.add(val, tmp)
73
74          mask = (count == 0)
75          val[mask] =-32767
76          count[mask] = 1
77          return np.divide(val.astype(np.double), count.astype(np.double))
78
79  if(__name__=='__main__'):
80          data = averaged_data(sys.argv[2:])
81          array_to_raster(sys.argv[1], data, 'jet.txt')
```

参考文献

[1] “Tile Map Service in Geoide”, http://geoikia.idgis.eu/wiki-english/index.php

[2] “Slippy map tilenames”, http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames

[3] GCOM-W1 データ提供サービス https://gcom-w1.jaxa.jp/auth.html

編集後記

「タイルマップサービスを作ってみた」をお送りしました。が、時間がなくて内容がペラペラで申し訳ありません。夏コミもこのネタでいく予定ですので、それまでに加筆します。

書くべき内容として、カラーテーブルの作成方法や Leaflet や OpenLayers を使ったブラウザ表示アプリ、Web サーバの設定等があります。また、ブラウザから要求された必要なタイルだけダイナミックに生成するサービスもできているので、それについても書きたかった...orz

このような事になった言い訳としては、結婚して自由になる時間が減った、というかリア充過ぎて薄い本のネタなど作っている暇がないというのが実情です。また、研究者から IT セキュリティエンジニアに転職して、家に返ってきてまで PC 触りたくないというのもあります。と、奥様やら仕事やらに責任転嫁していますが、創造性が失われてきたという現実も認識しています。これは結構ヤバい。

この厳しい現実に贖いつつ、夏コミまでには加筆修正しますので、冒頭に書いた GitHub リポジトリで加筆後のコンテンツを入手することができます。というか、お代は印刷代にすらならない事もあり、これまでに書いたものも公開していますのでよろしければご覧ください。

最後に、この同人誌は Debian/GNU Linux、TeXLive2016、psutils、git、GNU Make、vim といった、オープンソースソフトウェアを使って作成されました。また日本語のフォントは IPAex フォントを PDF に埋め込んでいます。このような有益なソフトウェアを開発、維持、管理していただいているすべての皆様に感謝します。また、このページまでたどり着いてくれた読者の方（おそらくあなただけです）に感謝します。ありがとうございました。

2017 年 4 月　Keisuke Nakao (@jm6xxu)