タイルマップサービスを作ってみた（追補版）

2017年8月11日

C92

Google Map的なWeb地図サービスの開発

はじめに

さすがに C92 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、その上に水循環変動観測衛星「しずく」（GCOM-W）という地球観測衛星が観測した海水面温度 (SST) のデータを重ねた情報を提供するサービスを開発します。このデータは、JAXA に利用申請をすると、無料で使用することができます。

SST は、数値データとして提供されますので、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、定期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、自動化を見据えて、コマンドラインプログラム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おり、Linux の基本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。適宜解説に努めますが、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してください。

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 1 のようなピラミッド構造になります。

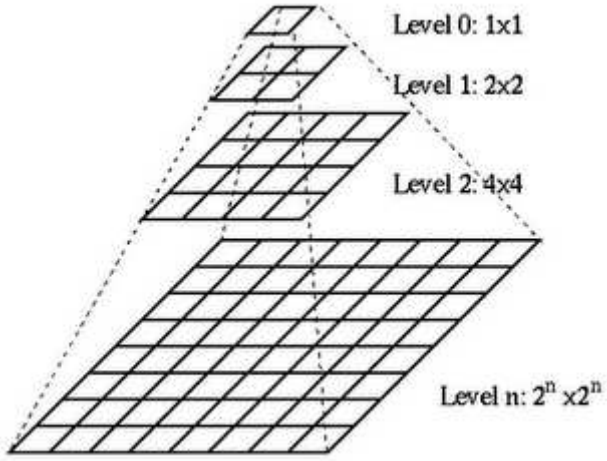

図 1　タイルのピラミッド構造 [1]

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。

それでは、タイルサーバを作っていきましょう。次章では、まず OpenStreetMap や地理院地図に重ねる海水面温度のデータを可視化し、地理情報を持った GeoTiff 形式の画像を作成します。

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。

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。

数値を画像にするためには、値と色の対応を決めなければなりません。この対応は、python の plot ライブラリである、matplotlib を使いましょう。どのような色使いにするかは、matplotlib のサイトにサンプルがあるのでそちらを参照してください。(https://matplotlib.org/examples/color/colormaps_reference.html) 今回はこのような図でよく使われる、jet を使用しました。具体的には、Listing 1 の make_colortable 関数で作っています。

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 でこのスクリプトのデジタルデータを公開しています。

listing 1 の関数 get_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可視化しています。変換した、RGBA のチャンネルを 41 行目からはじまる array_to_raster 関数で GeoTiff を作成しています。50 行目の for ループで各チャンネルを書き込んでいます。47 行目で 1 ピクセルあたりの座標の変化量を、55 から 57 行目で測地系の情報を指定しています。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 を指定しています。

このコードを使って生成した GeoTiff の例を図 2 に示します。印刷物だと色がわからないので、是非

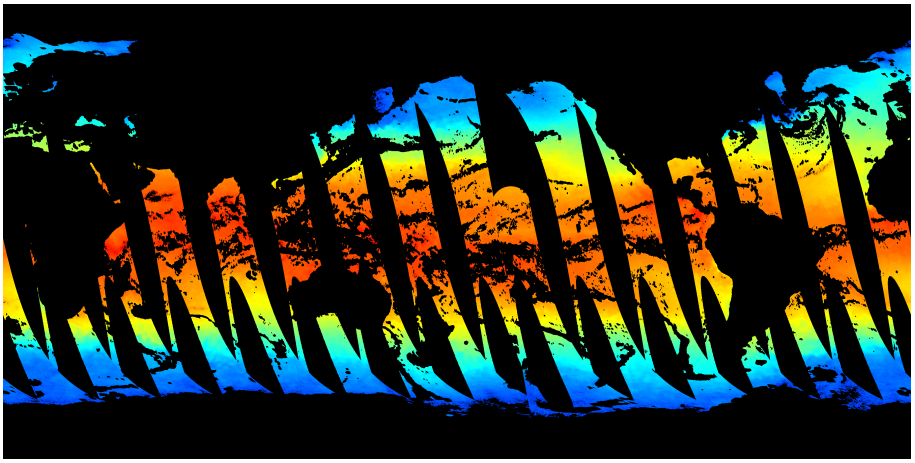

図 2　海水面温度

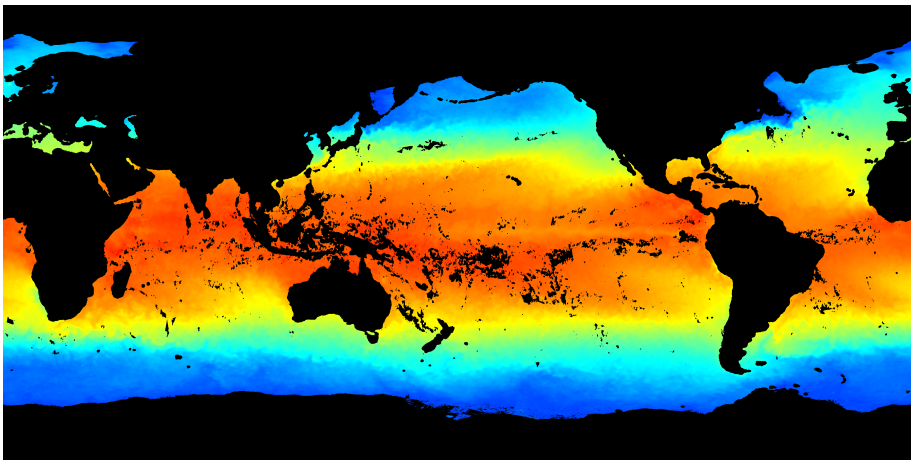

図 3　海水面温度 3 日間平均値

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図 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。

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
from matplotlib import cm

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):
			i = index(data[y][x])
			if(i>=1000):
				i = 999
			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():
	cm.jet.N = 256
	ct = gdal.ColorTable(gdal.GPI_RGB)
	for i in range(cm.jet.N):
		ct.SetColorEntry(i, tuple(map(lambda x:int(x*255), cm.jet(i))))

	ct.SetColorEntry(0, (0,0,0,0))
	return ct


def array_to_raster(outfile, data):
	ct = make_colortable()
	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))

	ci = (gdal.GCI_RedBand, gdal.GCI_GreenBand, gdal.GCI_BlueBand,gdal.GCI_AlphaBand)
	for i in range(4):
		b = out.GetRasterBand(i+1)
		b.SetColorInterpretation(ci[i])
		b.WriteArray(band[i])
		b.FlushCache()
	srs = osr.SpatialReference()
	srs.ImportFromEPSG(4326)
```

```
57        out.SetProjection(srs.ExportToWkt())
58
59
60 def averaged_data(files):
61        val = np.zeros((1800,3600),np.int32)
62        count = np.zeros((1800,3600),np.int16)
63
64        for f in files:
65               h5 = h5py.File(f)
66               data = expand_data(h5)
67               h5.close()
68
69               mask = data>-99
70               tmp = np.zeros((1800,3600),np.int16)
71               tmp[mask] = 1
72               count = np.add(count,tmp)
73
74               tmp = np.copy(data)
75               tmp[np.logical_not(mask)] = 0.
76               val = np.add(val, tmp)
77
78        mask = (count == 0)
79        val[mask] =-32767
80        count[mask] = 1
81        return np.divide(val.astype(np.double), count.astype(np.double))
82
83 def usage():
84        print '%s␣output_file␣h5file(s)␣...'%sys.argv[0]
85
86 if(__name__=='__main__'):
87        if(len(sys.argv) < 2):
88               usage()
89        else:
90               data = averaged_data(sys.argv[2:])
91               array_to_raster(sys.argv[1], data)
```

タイルサーバの作成

Web Server の準備

TMS は、タイルを HTTP プロトコルで配布します。ですので、ネットワークで到達可能な場所に Web サーバを準備しなければなりません。例えば、AWS の EC2 やさくら VPS、IDC クラウドなどでインスタンスを立てれば良いでしょう。OS は Linux であればディストリビューションは問いませんが私は Debian を使っているので、Debian を使用した例をご紹介します。Ubuntu でもほとんどは変わらないはずです。BSD 派の人は、試してはいませんが、FreeBSD でも OpenBSD でも NetBSD でも OK なはずです。

一般に公開せずに、ただやってみるだけであれば、Virtualbox 等の仮想 PC でも良いでしょう。Virtualbox であれば無料でダウンロードすることができます。

さて Web Server は、nginx を使います。apache が慣れているのであれば apache でも良いでしょう。apt でインストールしましょう。（正直、HTTP で画像が配布できれば良いので、何でもよいです）

タイルの作成

では、タイルを作成しましょう。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、/var/www/html/に生成するには、以下のコマンドを実行します。

```
$ gdal2tiles.py -z 0-8 map.tiff /var/www/html/
```

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。

Web サーバのホスト名が、tms.example.org だったとすると、http://tms.example.org/z/x/y.png という URL でタイルを取得することができます。

Leaflet によるブラウザへの表示

タイルサーバを作成したら、ブラウザへの表示を行ってみましょう。GoogleMap のようにマウスでぐりぐり動く地図を作るための JavaScript ライブラリがすでに公開されています。特に、OpenLayers と Leaflet が二大巨塔です。今回は Leaflet を使ってブラウザに表示してみます。

OpenStreetMap の表示

まずは、ベースタイルとなる、OpenStreetMap を表示する JavaScript を書いてみます。OpenStreetMap を表示するだけなら簡単です。Listing 2 と 3 に OSM を表示する HTML と JavaScript を示します。Listing 3 の JavaScript は、map.js というファイル名で保存されているとします。

Listing 2 では、Leaflet の CSS と Script を CDN から読み込んでいるだけです。CDN の URL や integrity は、Leaflet Tutorial (http://leafletjs.com/examples/quick-start/) にあります。HTML の BODY にある、map という ID が付いた DIV 要素に地図が表示されます。DIV 要素は、デフォルトではサイズが 0 なので、Stylesheet で地図を表示するサイズを指定します。これを忘れると、ちゃんと動いているのに表示されないとハマる事になります。今回は、style 属性に書いています。

Listing 2　OpenStreetMap の表示 (HTML)

```
<!doctype html>
<html lang="ja">
<head>
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.1.0/dist/leaflet.css"
       integrity="sha512-wcw6ts8Anuw10Mzh9Ytw4pylW8+
           NAD4ch3lqm9lzAsTxg0GFeJgoAtxuCLREZSC5lUXdVyo/7yfsqFjQ4S+aKw=="
       crossorigin=""/>
<script src="https://unpkg.com/leaflet@1.1.0/dist/leaflet.js"
   integrity="sha512-mNqn2Wg7tSToJhvHcqfzLMU6J4mkOImSPTxVZAdo+lcPlk+
       GhZmYgACEe0x35K7YzW1zJ7XyJV/TT1MrdXvMcA=="
   crossorigin=""></script>
</head>
<body>
       <div id="map" style="height:400;width:100%"></div>
       <script src="./map.js"></script>
</body>
</html>
```

Listing 3 のスクリプトのほうは、Map オブジェクトのインスタンスを作成して OSM のタイルの URL を指定した、TyleLayer インスタンスを作成後、Map インスタンスに追加しているだけです。Map クラスの setView メソッドで地図の中心座標と、ズームレベルを指定しています。この例では、北緯 35.6299 度、東経 129.7919 度、ズームレベル 16 を指定しています。ビッグサイトのあたりが表示されるはずです。

Listing 3　OpenStreetMap の表示 (JavaScript)

```
1 var mymap = L.map('map').setView([35.6299,139.7949], 16);
2
3 L.tileLayer('http://tile.openstreetmap.jp/{z}/{x}/{y}.png', {
4                maxZoom: 18,
5                attribution: 'Map␣data␣&copy;␣'
6                       + '<a␣href="http://openstreetmap.org">OpenStreetMap</a>␣contributors,
                            ␣'
7                       + '<a␣href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA
                            </a>'
8 }).addTo(mymap);
```

TileLayer クラスのコンストラクタにタイルの URL と、著作権情報を指定しています。タイルの URL はすでに説明しました。著作権情報は、OSM の場合は必須です。地理院地図でも同様です。どのように記述するかは、各 Web ページをご覧ください。

- OpenStreetMap http://wiki.openstreetmap.org/wiki/JA:Legal_FAQ
- 地理院地図 http://maps.gsi.go.jp/help/use.html

タイルのオーバーレイ

では、これに今回作ったタイルを重ねましょう。Listing 3 を見ればわかるように、tileLayer のインスタンスを作って、map に add すれば良いでしょう。作成したタイルを重ねたものを表示する JavaScript を Listing 4 に示します。違いは、10 行目以降です。今回作成したタイルサーバは、TMS 準拠ですので OSM や GoogleMap と緯度が増える方向が異なります。そこで、tileLayer の tms パラメータに true を指定しなければならないことに注意してください。

Listing 4　タイルのオーバーレイ (JavaScript)

```
1 var mymap = L.map('map').setView([35.6299,139.7949], 2);
2
3 L.tileLayer('http://tile.openstreetmap.jp/{z}/{x}/{y}.png', {
4                maxZoom: 18,
5                attribution: 'Map␣data␣&copy;␣'
6                       + '<a␣href="http://openstreetmap.org">OpenStreetMap</a>␣contributors,
                            ␣'
7                       + '<a␣href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA
                            </a>'
8 }).addTo(mymap);
9
10 L.tileLayer('http://tms.example.org/{z}/{x}/{y}.png', {
11        maxZoom: 2,
12        tms: true}).addTo(mymap);
```

作成した HTML と JS をタイルサーバにデプロイすれば、オーバーレイしたマップを図 4 のように表示することができます。紙媒体だと、モノクロ印刷なので良くわからないと思います。GitHub にカラーの PDF をアップロードしておきますので、そちらをご覧ください。`https://github.com/chomy/DevelopmentOfTMS`

これで、地球観測衛星の観測データを可視化して、タイルとして公開するタイルサーバが完成しました。これは (細かい事は別にして)TMS に準拠した XYZ タイルサーバですので QGIS といった、GIS ソフトはもちろん様々なサービスで使用することができます。

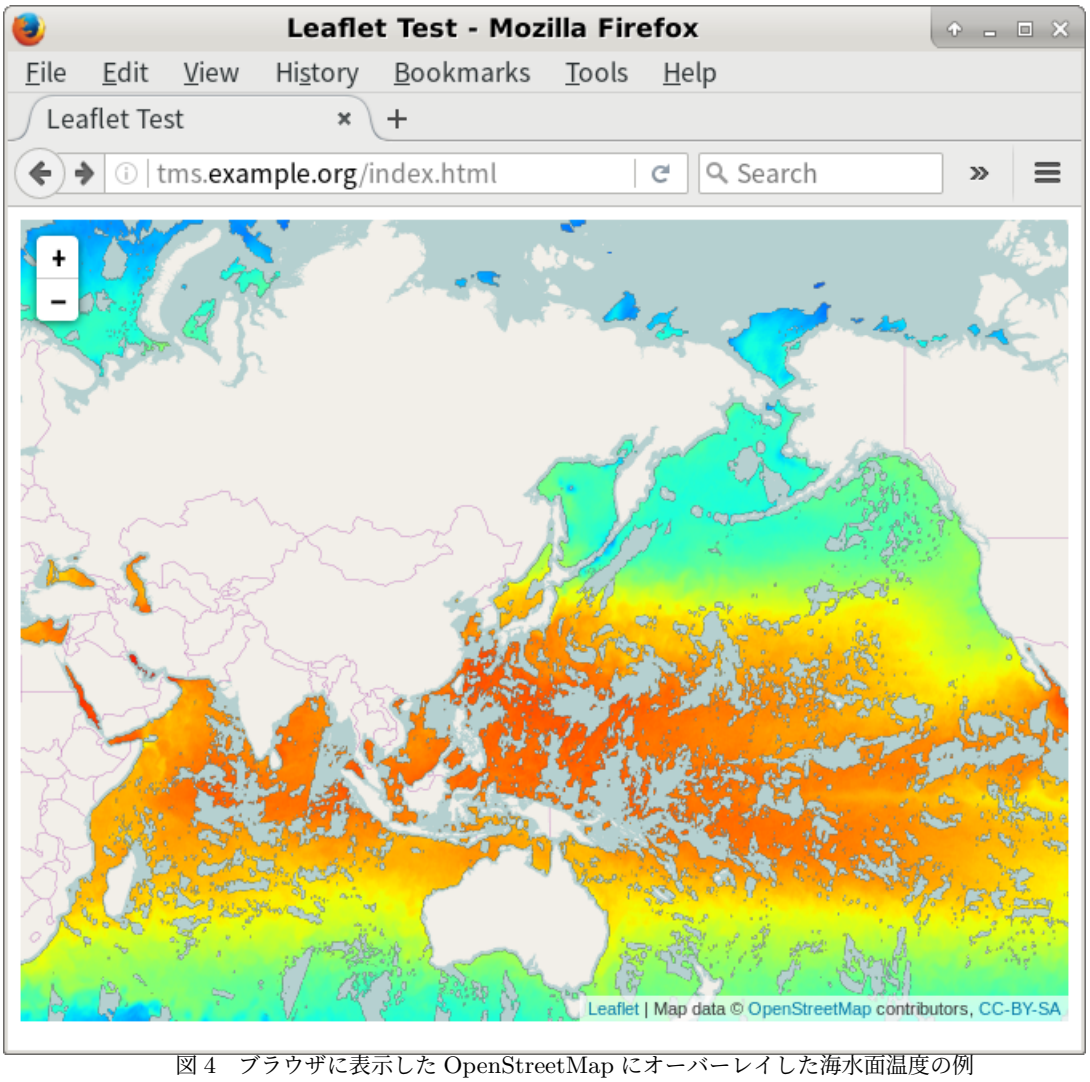

図 4　ブラウザに表示した OpenStreetMap にオーバーレイした海水面温度の例

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。

今回は、4 月に技術書典 2 で配布したものに、カラーテーブルの作成、Leaflet による描画等を大幅加筆しました。技術書典 3 もサークル参加が決まっているので、OpenLayers4 による描画や要求されたタイルをダイナミックに生成するタイルサーバ等の実装についても加筆できればなーと思っています。もちろん加筆後も、PDF で GitHub でダウンロード可能にします。

この仕事は、5 年ほど前からやっている aerial-project というプロジェクトの一部で、ずいぶん長くやっています。特に今の時季は、台風があるので、JTWC の進路予想と海水面温度を重ねて、勢力の推定とかできたりします。

とはいえ、そろそろ飽きたので違う事をやろうかとも思ってます。例えば、SDR(Software Defined Radio) という、ソフトウェアで無線をごにょごにょするデバイスで遊んでいるので、そのネタとか。（だがしかし、このネタでは UNIX 島では売れないんだよなー）あと、C++14 で導入された右辺値参照をアセンブラで検証するネタとか。（怖い人がいっぱいいる...）

まぁなんにせよ何か書きます。新刊になるか、このネタの続きになるかは未定ですが。

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 8 月　Keisuke Nakao (@jm6xxu)

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