Создание автономного картографического приложения на базе изображений без привязки

Обсудить в форуме Комментариев — 15

Эта страница опубликована в основном списке статей сайта
по адресу http://gis-lab.info/qa/zoomable-image-leaflet.html

Рассмотрен процесс подготовки и подключения изображений без привязки в картографический JavaScript-движок Leaflet

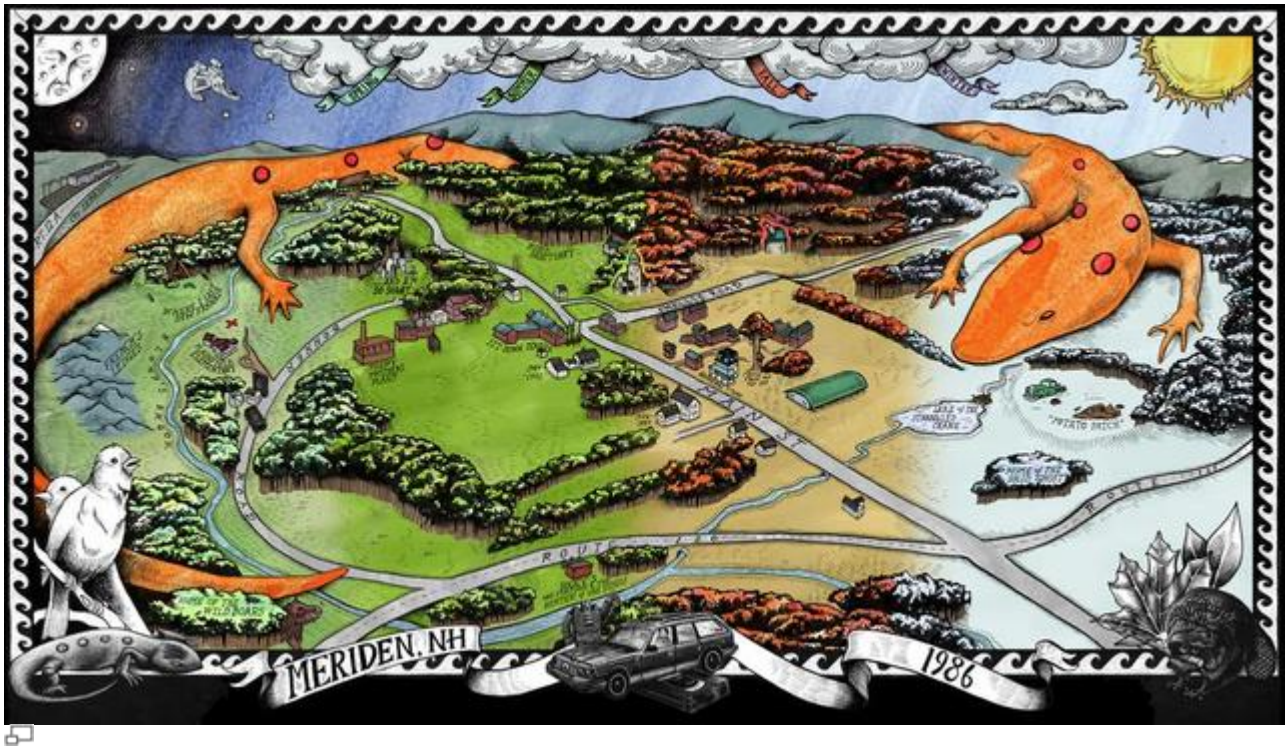
Содержание

Введение

Картографические JavaScript-движки, такие как OpenLayers или Leaflet, порой находят своё применение в таких областях, для которых они изначально вроде бы и не предназначались. Так, например, международное агентство Reuter продемонстрировало использование Leaflet для интерактивного взаимодействия с фотографиями из зала вручения кинопремии «Оскар»:

Пример №1 нестандартного использования картографического движка

Еще один необычный пример - интерактивный тур по городу на базе его вымышленной карты:



Пример №2 нестандартного использования картографического движка

Таких примеров можно привести много, их объединяет то, что все они построены на базе изображений, которые не имеют никакой географической привязки. Использование картографических движков для таких изображений добавляет возможность навигации (pan), а после небольшой предварительной обработки - возможность их масштабирования (zoom).

Именно вопросу такой предварительной подготовки и посвящена основная часть данной статьи. В качестве языка программирования будем использовать Python, в качестве JavaScript-движка - Leaflet.

Если перед вами стоит схожая задача, но вы не хотите вникать в технические моменты, тогда просто воспользуйтесь одним из онлайн-сервисов (например HUGEpic). Принцип работы с такими сервисами заключается в том, что вы загружаете свое изображение, а на выходе получаете готовую "карту", пример.

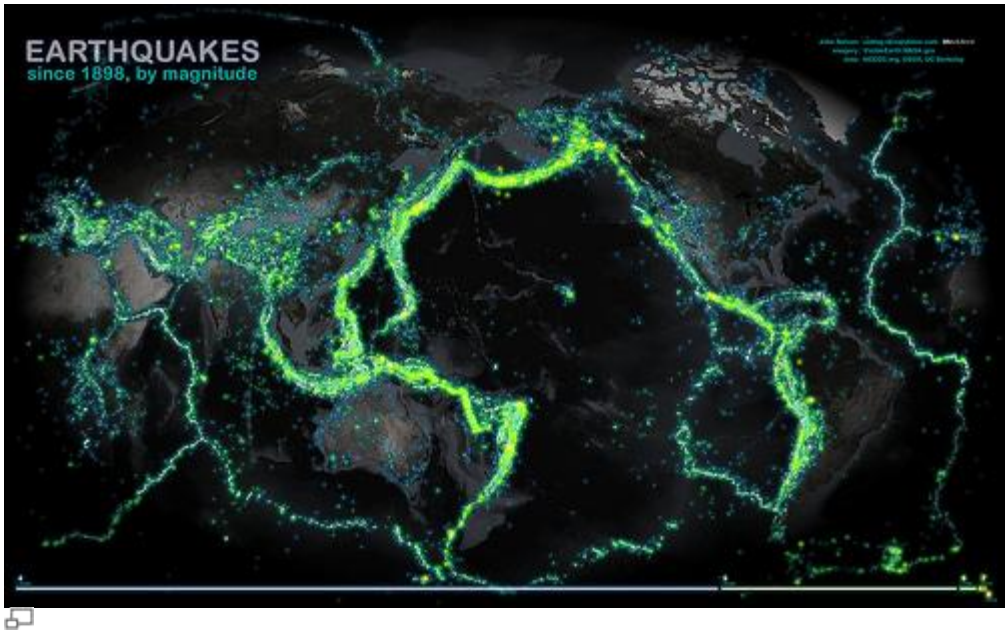
Обработка исходного изображения

Постановка задачи

Исходное изображение в принципе можно подключить "как есть", в OpenLayers для этого есть класс OpenLayers.Layer.Image, а в Leaflet - L.ImageOverlay, однако в общем случае это плохая идея, так как изображение может быть большим и пользователю придется очень долго ждать, пока оно загрузится, поэтому первое, что нужно сделать с изображением - это разбить его на тайлы. Поскольку мы хотим иметь возможность не только двигать наше изображение, но и изменять масштаб, то тайлы должны быть построены для нескольких масштабных уровней.

Рабочее изображения

Работать будем с визуализацией землетрясений, произошедших с 1898 года. Подробнее об изображении тут. Размер изображения - 3410 x 2058.



Исходное изображение (не оригинал, размер уменьшен)

Разбивка на тайлы

Переходим к написанию скрипта, который будет разбивать на тайлы наше изображение. Мы будем использовать инструменты стандартной библиотеки Python, а также одну стороннюю библиотеку - библиотеку для работы с изображениями - PIL, поэтому прежде чем продолжить работу, убедитесь, что у вас установлен PIL.

Используемая в дальнейшем схема разбивки на тайлы - как в OpenStreetMap (то есть нумерация строк и столбцов тайлов начинается с левого верхнего угла). Алгоритм разбивки изображения на тайлы достаточно прост, но имеет некоторую особенность. Представьте, что мы взяли наше изображение размера 3410 x 2058 и пытаемся разбить его на тайлы 256 x 256. Очевидно, что 3410 и 2058 не делятся без остатка на 256, а это означает, что тайлы последнего столбца и последней строки получатся не квадратными (82 x 256 и 256 x 10 соответственно). Поэтому, прежде чем переходить к нарезке тайлов, изображение должно быть приведено к такому виду, что его ширина и высота будут кратны размеру тайла. Размеры поправок вычисляются следующим образом (src_width, src_height - ширина и высота исходного изображения, tile_size - размер тайла, % - операция получения остатка от деления). В Python это записывается так:

```
dw = tile_size - src_width % tile_size
dh = tile_size - src_height % tile_size
```

Области расширения за счет поправок следует сделать прозрачными, в нашем коде за эту операцию будет отвечать функция *adjustBounds*.

Одним из важных параметров процедуры разбивки на тайлы - это максимальный масштабный уровень. Рассчитывается он как логарифм по основанию 2 числа max(w,h), округленный до ближайшего большего целого, где w - ширина изображения, h - высота. В Python это записывается так:

```
max_zoom = int(math.ceil(math.log((max(w, h) / tile_size), 2)))
```

Если посчитать, то мы получим:

```
max_zoom = int(math.ceil(math.log((max(3410, 2058) / 256), 2))) = 4
```

В итоге алгоритм разбивки на тайлы сводится к циклу от самого детального масштабного уровня (в нашем случае от 4) до 0. В цикле выполняются следующие операции: на 4-м уровне берется исходное изображение, к нему применяется функция *adjustBounds*, нарезаются тайлы и раскладываются по соответствующим каталогам, после чего размер исходного изображения (не того, к которому применена функция *adjustBounds*) уменьшается в 2 раза и так далее.

Полная версия скрипта:

```
# -*- coding: utf-8 -*-
import sys
import os
import math
from PIL import Image

# source path
img_path = sys.argv[1]

# tile size
tile_size = 256

# tile directory
tile_path = 'tiles'


def adjustBounds(src_img):

    # get size of original image
    src_width, src_height = src_img.size

    # calculate size of target image (background)
    target_width = src_width + (tile_size - src_width % tile_size)
    target_height = src_height + (tile_size - src_height % tile_size)

    # create transparent background
    target_img = Image.new('RGBA', (target_width, target_height))

    # combine original image and background
    target_img.paste(src_img, (0, 0))

    return target_img


img = Image.open(img_path)
w, h = img.size[0], img.size[1]

# calculate max zoom level
max_zoom = int(math.ceil(math.log((max(w, h) / tile_size), 2)))

for z in range(max_zoom, -1, -1):

    adjusted_image = adjustBounds(img)

    numcolumns = adjusted_image.size[0] / tile_size
    numrows = adjusted_image.size[1] / tile_size

    for x in range(numcolumns):

        # create z/x/ directory
        path = os.path.join(tile_path, str(z), str(x))
        if not os.path.isdir(path):
            os.makedirs(path)

        for y in range(numrows):
            bounds = (x * tile_size, y * tile_size, (x + 1) * tile_size, (y + 1) *
tile_size)
            tile = adjusted_image.crop(bounds)
            tile.save('%s/%s.png' % (path, y))

    w, h = img.size[0], img.size[1]
    img = img.resize((w / 2, h / 2), Image.ANTIALIAS)
```

Поместим файл со скриптом (*tiling.py*) в ту же папку, где лежит исходное изображение (*earthquakes.jpg*) и запускаем обработку:

```
python tiling.py earthquakes.jpg
```

После выполнения данного скрипта в этой же папке будет создан каталог *tiles*, в который будут помещены тайлы.

Подключение тайлов в Leaflet

В этой же директории, где находится файл *tiling.py* и каталог *tiles* создадим файл *display.html* и поместим в него следующий код:

```
<html>
<head>
    <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.css" />
    <script src="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.js"></script>
</head>
<body>
    <div id="map" style="width: 750px; height: 350px;"></div>
    <script>
        map = L.map('map', {
          maxZoom: 4,
          minZoom: 0,
          crs: L.CRS.Simple
        }).setView(-60,100, 1);

        L.tileLayer('./tiles/{z}/{x}/{y}.png', {
            attribution: 'These bicycles are for other drivers &copy;',
            noWrap: true
        }).addTo(map);
    </script>
</body>
</html>
```

Отметим, что проекция объекта карты выставлена в значение *L.CRS.Simple*. Согласно документации, именно это значение используется в случае опубликования изображений, не имеющих географической привязки. Открываем созданный файл в браузере и наблюдаем следующий результат:

Заключение

В ходе данной статьи была описана процедура разбивки изображения без привязки на тайлы, а также подключение их в Leaflet. Отметим, что операцию по тайлированию изображения можно было выполнить, используя утилиту gdal2tiles, которая входит в состав библиотеки GDAL:

```
gdal2tiles.py -p raster -z 0-4 earthquakes.jpg
```

Описанный в статье вариант решения задачи не требует использования GDAL, плюс (что самое, наверное, важное) код довольно простой и наглядно дает понимание того, как вычисляются масштабные уровни и как выполняется непосредственно сама разбивка на тайлы. По коду gdal2tiles разбираться в этих вопросах гораздо сложнее, так как это более универсальный инструмент. И еще, gdal2tiles помимо генерирования тайлов также создает html-файлы с включенными в них движками OpenLayers и Google (но не Leaflet) и подключает на карту набор созданных тайлов, то есть по-сути делает все то же самое о чем идет речь в данной статье, но как-бы за кадром, мы же попытались представить этот процесс более наглядно.

Полезные ссылки

1. Using leaflet.js with non-geographic imagery
2. Zoomable image with Leaflet
3. Images as Maps

Обсудить в форуме Комментариев — 15

Последнее обновление: 2014-05-15 00:08

Дата создания: 28.07.2013
Автор(ы): Denis Rykov