Разработка микросервиса ретайлирования на языке программирования Python

П.А. Мишин, П.А. Мишина

Национальный исследовательский Мордовский государственный университет им. Н. П. Огарёва, Саранск

Аннотация: В современном мире всё чаще возникает необходимость обработки географической информации в самых различных формах. В данной работе рассматриваются понятие «тайл», его назначение, особенности, а также процесс ретайлирования, представляющий собой метод создания и обновления тайлов. Данная технология способствует повышению эффективности работы современных картографических сервисов, сокращая время загрузки карт. Последовательно представлены основные этапы разработки микросервиса, реализующего логику ретайлирования. Основным поставщиком данных выступает открытый проект OpenStreetMap (OSM). Набор пространственных данных является основным продуктом OSM и содержит актуальные географические данные и информацию со всего мира. Основу технологического стека составляет язык Python, к которому добавляются специализированные модули для работы с тайлами, а также библиотека для реализации простого и качественного API.

Ключевые слова: Python, тайл, ретайлирование, OpenStreetMap, микросервис, Flask-RESTX, mercantile.

Введение

Тайлы [1] (tile – плитка) – упорядоченные по сетке квадратные изображения, отображающие карту. Тайлы карты, как правило, представляют собой изображения размером 256 на 256 пикселов (см. рис. 1). Данный размер является неким стандартом, однако можно встретить и другие величины. Например, платформа CloudMade предлагает изображения размером 64 на 64 пикселов для мобильных устройств. Ретайлирование [2] – процесс генерации тайлов на основе тайлов более высокого или низкого уровня.

Рис. 1. – Пример тайла (256 на 256 пикселов)

Набор тайлов (tileset) обычно подразумевает достаточное количество тайлов, формирующих очень большое изображение, если они отображаются все сразу, на разных уровнях приближения. Смысл состоит в том, чтобы отображать определённую область карты на веб-сайте, а не показывать всё сразу.

Формирование визуального представления информации – одно из назначений систем хранения пространственных данных [3, 4]. В настоящее время данный процесс осуществляется на основе организации и применения тайловой структуры геоданных [5, 6]. Такая технология основана на комбинации растровой и иерархической моделях. Каждый тайл представляет собой изображение формата PNG (карты, слои) или JPG (спутниковые снимки) и хранится в файле с уникальным именем, которое определяется координатами данного тайла по осям X и Y. Тайлы, с методологической точки зрения, представляют собой структурные единицы информационной конструкции «карта». Количество тайлов, из которого состоит изображение, зависит от масштаба. Изображение на масштабе z1 (самом минимальном) на сервисе Google Maps состоит всего из 4-х тайлов. На следующем масштабе количество тайлов в 4 раза больше, чем на предыдущем, так как каждый тайл разбивается пополам по горизонтали и по вертикали.

Использование тайловой структуры позволяет при просмотре через Интернет загружать не всё изображение целиком, а только необходимую часть, которая отображается на экране, что экономит трафик и время. Также стоит отметить, что тайловая технология позволяет хранить небольшой набор тайлов вместо большого растрового изображения. Таким образом, данная технология на порядки уменьшает объем хранимых растровых изображений.

Рендеринг является достаточно ресурсоемким процессом. Сервер не делает карту в режиме реального времени для каждого пользователя, просматривающего карты. Участки карты предварительно подготавливаются и сохраняются на диске.

Существуют два различных набора тайлов: Mapnik [7] и Osmarender. Тайлы Mapnik в настоящее время находятся на открытом веб-картографическом проекте OpenStreetMap [8, 9]. Базы данных Mapnik обновляются ежечасно, так что большинство изменений данных должны получать отображение в течение часа. Полное обновление планеты осуществляется каждую неделю, чтобы избежать любых неточностей в нанесении данных. Каждый тайл имеет временную метку того, когда он был вынесен и некий флаг, означающий, что он готов для обновления. Визуализации идут по следующим правилам:

- всякий раз, когда вы смотрите на тайл он проверяется: старше ли он семи дней;
- если он старше семи дней, то помечается флагом и, следовательно, обновляется;
- фоновый просчет генерирует список всех тайлов с флагами, а затем переделывает их все;
- как только закончился просчет, то запрашивается список тайлов с флагами снова.

Таким образом, если никто не смотрит на место, оно не будет повторно вынесено в список. Тайлы предоставляются отсортированными по интересу или вниманию. Маркировка тайлов флагами не отмечает все подтайлы (на других масштабах).

Материалы и методы

Термин «Slippy map» относится к главному отображению карты на OpenStreetMap. Веб-интерфейс ожидает, что плитки будут отображаться по URL-адресам, следующим по одной схеме, поэтому все URL-адреса сервера плиток выглядят довольно похоже. Первая часть URL-адреса указывает сервер плиток. Координаты плиток обычно указываются с помощью /zoom/x/y.png.

Параметр масштабирования (zoom) представляет собой целое число от 0 (уменьшенное) до 18 (увеличенное). Обычно 18 – это максимум, однако некоторые плиточные серверы могут превышать это значение.

Данный сервис [10] позволяет на основе заданных параметров сгенерировать тайл на основе тайлов более высокого или низкого уровня.

Файл сервиса tile_provider.py отвечает за генерацию шаблона ссылки для обращения к серверам провайдеров.

Файл retiler.py содержит основную логику генерации тайлов.

Файл api.py обеспечивает для сервиса API формата:

GET /retile/{provider}/{level}/{resolution}/{z}/{x}/{y}.png

Основные классы:

– TileProvider
– Retiler

Классы исключений:

– InvalidInputParamException

Классы маршрутизации:

– GetRetile

На рис. 2 показана диаграмма классов, сформированная в IDE PyCharm.

Класс TileProvider инициализируется параметрами из файла конфигурации (url_template, tile_size, max_zoom, token). Метод download_tile с помощью метода create_url формирует ссылку для запроса исходного тайла и возвращает его в виде изображения.

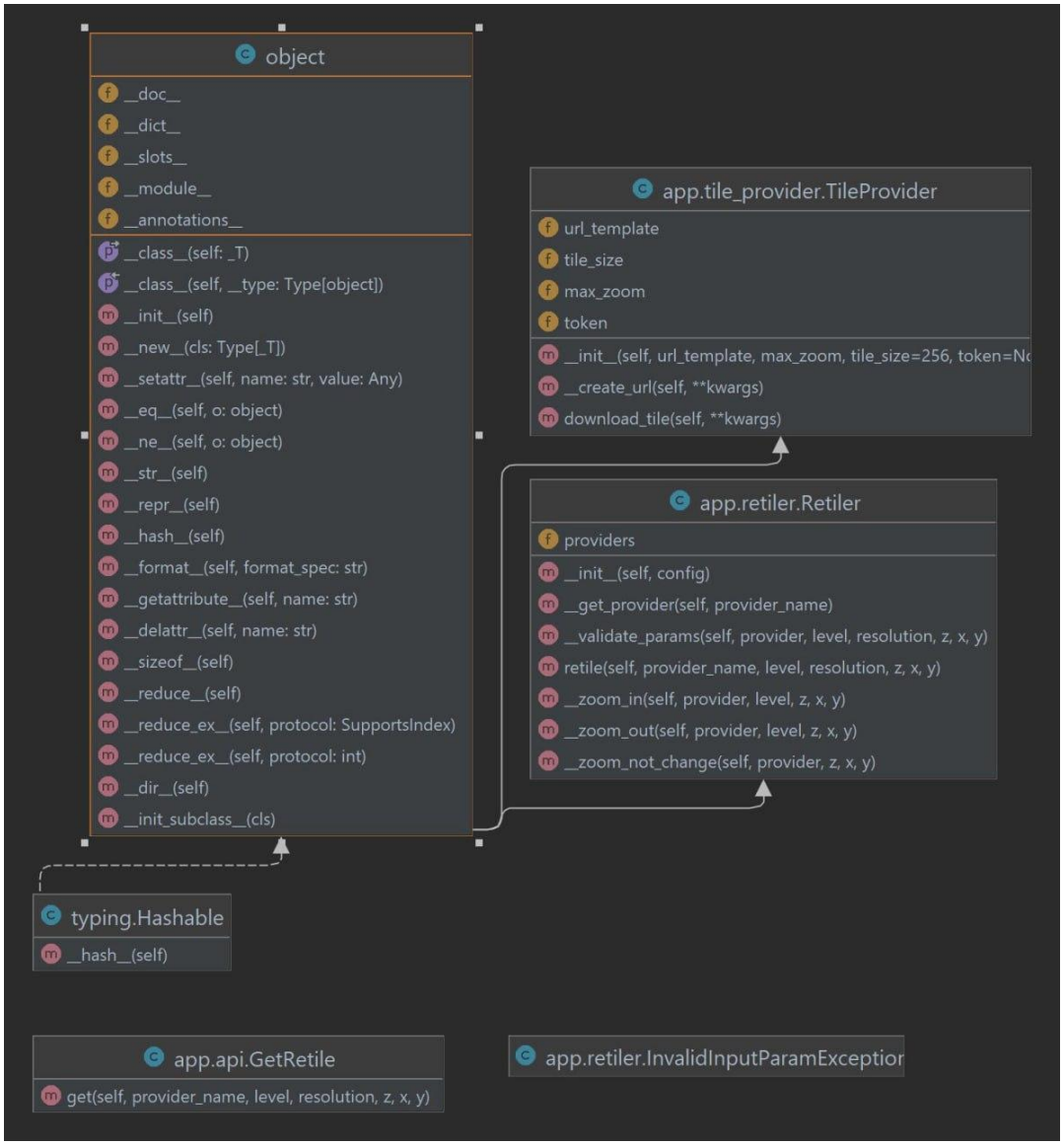

Рис. 2. – Диаграмма классов

Класс Retiler инициализируется файлом конфигурации и на основе класса TileProvider формирует и заносит в поле providers список доступных провайдеров в виде объектов класса TileProvider.

Данный класс содержит метод __get_provider, который принимает название провайдера и проверяет его наличие в списке providers. В случае успеха возвращаемым значением будет соответсвующий элемент списка providers, в противном случае будет вызвано исключение типа InvalidInputParamException.

Метод validate_params класса Retiler выполняет проверку входных параметров (provider, resolution, z, x, y), в случае некорректных входных данных будет вызвано исключение типа InvalidInputParamException.

Метод retile в зависимости от входного параметра level вызывает соответсвующий метод __zoom_in, __zoom_out или __zoom_not_change.

Результаты эксперимента

Рассмотрим структуру и логику работы основных методов генерации тайлов, а также реализацию HTTP-методов во Flask с использованием Flask-RESTX.

Метод __zoom_in отвечает за генерацию нового тайла на основе тайлов более высокого уровня при положительном смещении зума (координата z). Реализация данного метода приведена в листинге 1.

Листинг 1.

```
    def __zoom_in(self, provider, level, z, x, y):
            tile = mercantile.Tile(x=x, y=y, z=z)
            tile_children = list(mercantile.children(tile,
zoom=z + level))
            size = provider.tile_size
            min_x = min([t.x for t in tile_children])
            min_y = min([t.y for t in tile_children])
            max_x = max([t.x for t in tile_children])
            max_y = max([t.y for t in tile_children])
```

```
            map_image = Image.new('RGB', (size * (max_x -
min_x + 1), size * (max_y - min_y + 1)))
            for t in tile_children:
                img = provider.download_tile(x=t.x, y=t.y,
z=t.z)
                map_image.paste(img, box=((t.x - min_x) *
size, (t.y - min_y) * size))
            return map_image
```

Входными параметрами приведенного метода являются:

- provider – поставщик данных, например, Mapbox, Google и т.п;
- level – уровень смещения зума (координата z), на основе которого необходимо производить ретайлирование, например, если level = 2, z = 14, тайл будет рассчитываться на основе тайлов 16 зума;
- z – координата z тайла;
- x – координата x тайла;
- y – координата y тайла.

Библиотека mercantile предоставляет функцию Tile для генерации тайла на основе заданных координат x, y, z. С помощью данной функции формируется исходный тайл. Затем с помощью функции children формируется список дочерних тайлов уровня z + level. В переменную size помещается значение стороны одного тайла. Далее высчитываются минимальные и максимальные координаты x и y дочерних тайлов, на основе которых создается кластер нужного размера (Image.new) для результирующего тайла. После этого в цикле выполняется подгрузка с сервера каждого дочернего тайла и его вставка в соответствующее место кластера. Получившееся в результате изображение – это новый тайл, являющийся возвращаемым значением метода.

Возможный результат работы функции __zoom_in представлен на рис. 3 (размеры исходного и результирующего тайла для наглядности приведены в одном размере).

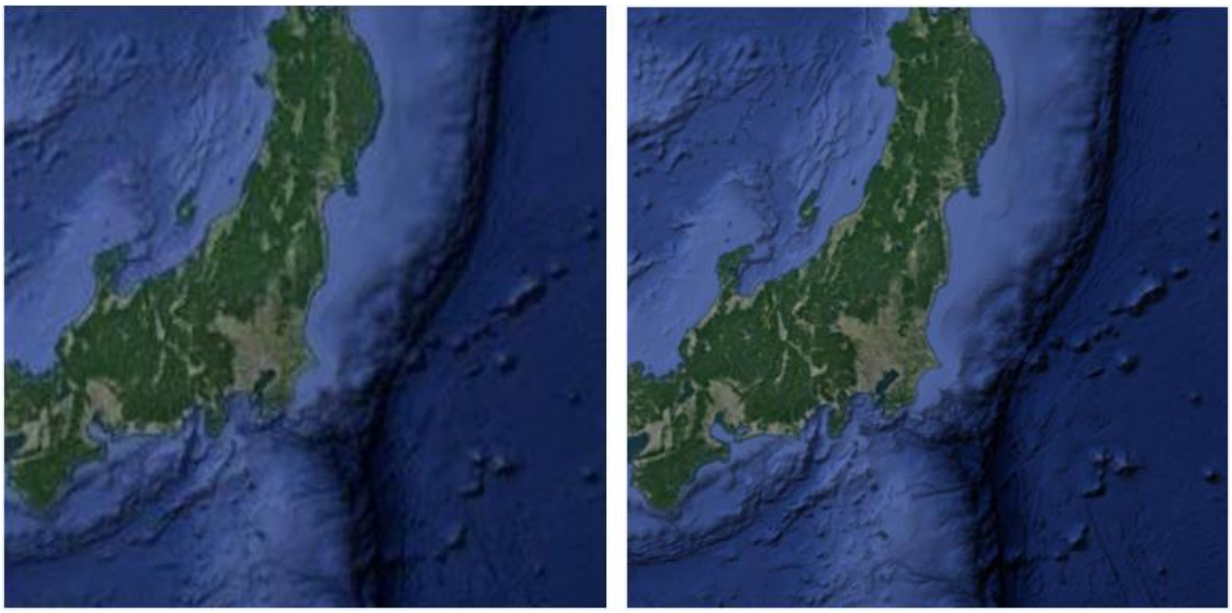

Рис. 3. – Исходный и результирующий тайлы функции __zoom_in с параметрами provider = google, level = 2, z = 5, x = 28, y = 12

Метод __zoom_out отвечает за генерацию нового тайла на основе тайлов более низкого уровня при отрицательном смещении зума. Реализация данного метода приведена в листинге 2.

Листинг 2.

```
    def __zoom_out(self, provider, level, z, x, y):
            size = provider.tile_size // (2 ** abs(level))

            tile = mercantile.Tile(x=x, y=y, z=z)
            tile_parent = mercantile.parent(tile, zoom=z +
level)
            tile_children =
list(mercantile.children(tile_parent, zoom=z))
```

```
            min_x = min([t.x for t in tile_children])
            min_y = min([t.y for t in tile_children])

            map_image =
provider.download_tile(x=tile_parent.x, y=tile_parent.y,
z=tile_parent.z)
            map_image = map_image.crop(((tile.x - min_x) *
size, (tile.y - min_y) * size, (tile.x - min_x + 1) * size,
(tile.y - min_y + 1) * size))
            return map_image
```

У функции __zoom_out входные параметры совпадают с параметрами метода __zoom_in. Сначала в зависимости от уровня смещения зума вычисляется size – размер стороны результирующего тайла. Затем создается исходный тайл по его координатам, а на его основе с помощью функции parent вычисляется родительский тайл уровня z + level (level < 0). После чего необходимо получить список дочерних тайлов родительского тайла и найти их минимальные координаты x и y. Эти координаты будут указывать на необходимую для вырезания (crop) из родительского тайла область, которая и будет являться нужным тайлом. Предварительно родительский тайл следует загрузить с сервера.

Возможный результат работы метода __zoom_out представлен на рис. 4 (размеры исходного и результирующего тайла для наглядности приведены в одном размере).

Рис. 4. – Исходный и результирующий тайлы функции __zoom_out с параметрами provider = google, level = -2, z = 5, x = 28, y = 12

В случае нулевого смещения зума метод __zoom_not_change вернет исходный тайл (листинг 3).

Листинг 3.

```
def __zoom_not_change(self, provider, z, x, y):
    map_image = provider.download_tile(x=x, y=y, z=z)
    return map_image
```

Анализируя полученные результаты, можно заметить, что при увеличении зума сгенерированный тайл имеет более высокое качество, а при уменьшении зума, наоборот, более низкое качество.

После написания основной логики, необходимо позаботиться о взаимодейтсвии микросервиса с поставщиками данных, то есть требуется разработать API.

Flask – это упрощенная платформа Python для веб-приложений, которая обеспечивает основные возможности маршрутизации URL-адресов и визуализации страниц. Flask-RESTX – это расширение для Flask, которое добавляет поддержку для быстрого создания REST API.

Для микросервиса ретайлирования реализация REST API приведена в листинге 4.

Листинг 4.

```
from flask import Flask
from flask_restx import Api, Resource
...
app = Flask(__name__)
api = Api(app)
...
@api.route('/retile/<string:provider_name>/<int(signed=True)
:level>/<int(signed=True):resolution>/<int(signed=True):z>/<
int(signed=True):x>/<int(signed=True):y>.png')
class GetRetile(Resource):
    def get(self, provider_name, level, resolution, z, x,
y):
        res_img = retiler.retile(provider_name, level,
resolution, z, x, y)
        buf = io.BytesIO()
        res_img.save(buf, format='PNG')
        buf.seek(0)
        byte_im = buf.getvalue()
        return send_file(buf, mimetype="image/png")
if __name__ == '__main__':
    ...
    app.run(debug=False)
```

Необходимо инициализировать основную точку входа для приложения (api = Api(app)) с помощью приложения Flask.

Прежде чем описывать HTTP-методы приложения, нужно настроить маршрутизацию при помощи метода route. При необходимости можно указать параметры запроса, например, <int(signed=True): level>, где атрибут signed указывает на то, что level – целое число со знаком.

Основным строительным блоком, предоставляемым Flask-RESTX, являются ресурсы (Resource). Ресурсы предоставляют более легкий доступ к нескольким HTTP-методам, просто определяя методы на вашем ресурсе.

Приведенный в листинге метод get соответствует HTTP-методу GET. В нем вызывается метод retile пользовательского класса Retiler, который отвечает за вызов соответствующих входным параметрам методов __zoom_in, __zoom_out и __zoom_not_change. Ответом метода GET будет файл с расширением PNG. Также сервис способен возвращать следующие коды ответа:

- 200 – когда тайл успешно сгенерирован;
- 422 – переданы неверные параметры запроса;
- 500 – произошла внутренняя ошибка сервиса.

Выводы

Подводя итог вышесказанному, можно сделать вывод о значимости технологии ретайлирования, которая оказывается эффективным инструментом в плане снижения времени загрузки карт и повышения общей производительности сервисов.

Особое внимание уделено разработке микросервиса, осуществляющего логику ретайлирования. Важным аспектом является выбор технологического стека, где применение языка программирования Python с необходимыми библиотеками позволяет достичь оптимальной производительности и удобства в разработке.

Данная работа направлена на то, чтобы расширить понимание технологии ретайлирования и ее применения в современных картографических сервисах, а также предоставить практические рекомендации для разработчиков, деятельность которых непосредственно связана с географической информацией.

Литература

1. Wei L.Y., 2004. Tile-based texture mapping on graphics hardware. Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, ACM, pp: 55-63.

2. McNeill, G. and A.S. Hale, 2017. Generating tile maps. Computer graphics forum, 3(36). Date Views 19.10.2023 doi.org/10.1111/cgf.13200.

3. Матчин В.Т. Состояние и развитие инфраструктуры пространственных данных // Образовательные ресурсы и технологии. 2015. №1(9). С. 137-144.

4. Цветков В.Я. Геоинформационные системы и технологии. М.: Финансы и статистика, 1998. 288 с.

5. Цветков В.Я. Обновление банков данных пространственной информации // Информатизация образования и науки. 2015. №1. С. 128-136.

6. Цветков В.Я. Тайловое представление пространственной информации // Международный журнал прикладных и фундаментальных исследований. 2016. №10-4. С. 670-671.

7. Haklay, M. and P. Weber, 2008. Openstreetmap: User-generated street maps. IEEE Pervasive computing, 4(7): 12-18.

8. Bennett, J., 2010. OpenStreetMap. Birmingham: Packt Publishing Ltd, pp: 234.

9. Neis, P. and D. Zielstra, 2014. Recent Developments and Future Trends in Volunteered Geographic Information Research: The Case of OpenStreetMap. Future internet, 1(6). Date Views 27.08.2023 doi.org/10.3390/fi6010076.

10. Шитько А.М. Проектирование микросервисной архитектуры программного обеспечения // Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2017. №9. С. 122-125.

References

1. Wei L.Y., 2004. Tile-based texture mapping on graphics hardware. ACM, pp: 55-63.

2. McNeill, G. and A.S. Hale, 2017. Computer graphics forum, 3(36). Date Views 19.10.2023 doi.org/10.1111/cgf.13200.

3. Matchin V.T. Obrazovatel′nye resursy i tekhnologii. 2015. №1 (9). pp. 137-144.

4. Tsvetkov V.YA. Geoinformatsionnye sistemy i tekhnologii [Geoinformation systems and technologies]. M.: Finansy i statistika, 1998. 288 p.

5. Tsvetkov V.Ya. Informatizatsiya obrazovaniya i nauki. 2015. №1. pp. 128-136.

6. Tsvetkov V.Ya. Mezhdunarodnyj zhurnal prikladnykh i fundamental'nykh issledovaniy. 2016. №10-4. pp. 670-671.

7. Haklay M. and P. Weber IEEE Pervasive computing. 2008. Vol.4, No. 7. pp. 12-18.

8. Bennett, J. OpenStreetMap. Birmingham: Packt Publishing Ltd, 2010. 234 p.

9. Neis, P. and D. Zielstra, 2014. Future internet, 1(6). Date Views 27.08.2023 doi.org/10.3390/fi6010076.

10. Shit'ko A.M. Trudy BGTU. Seriya 3: Fiziko-matematicheskie nauki i informatika. 2017. №9. pp. 122-125.

Дата поступления: 18.11.2023

Дата публикации: 3.01.2024