Основы работы динамических TMS-сервисов

Обсудить в форуме Комментариев — 13

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

Содержание

Введение

TMS (Tile Map Service) - сервис доступа к хранилищам тайлов. TMS-сервис может функционировать в одном из двух режимов - статическом и динамическом. В первом случае это ни что иное, как просто набор файлов, организованных определённым образом в файловой системе, так называемый "тайловый кэш" (здесь и далее в статье мы предполагаем, что тайловый сервис отвечает лишь за передачу самих тайлов (Tile Resources), остальные ресурсы, описываемые в спецификации, мы здесь не рассматриваем, в виду их тривиальности). При запросе конкретного тайла HTTP-сервер (Apache, nginx) самостоятельно переводит запрашиваемый URL в физическое расположение тайла в файловой системе и возвращает клиенту нужный тайл. Пример организации тайлового кэша:

```
└── admin_EPSG3857
    ├── 1
    │   ├── 0
    │   │   ├── 0.png
    │   │   └── 1.png
    │   └── 1
    │       ├── 0.png
    │       └── 1.png
    └── 2
        ├── 0
        │   ├── 0.png
        │   ├── 1.png
        │   ├── 2.png
        │   └── 3.png
        ├── 1
        │   ├── 0.png
        │   ├── 1.png
        │   ├── 2.png
        │   └── 3.png
        ├── 2
        │   ├── 0.png
        │   ├── 1.png
```

```
│   ├── 2.png
│   └── 3.png
└── 3
    ├── 0.png
    ├── 1.png
    ├── 2.png
    └── 3.png
```

В данном примере родительская директория admin_EPSG3857 содержит две директории 1 и 2, названия которых соответствуют номерам масштабных уровней. Имена каталогов на следующем уровне вложенности (0, 1 и 0, 1, 2, 3) определяют номер столбца тайловой сетки (x-координата), имя же самого тайла (с расширением .png) определяет номер строки (y-координата). В данном случае, чтобы получить, тайл с координатами (0, 0) на втором масштабном уровне, необходимо выполнить запрос, который может выглядеть следующим образом:

`http://10.22.0.9/tms/1.0.0/admin_EPSG3857/2/0/0.png`

Для создания тайлового кэша существует различное программное обеспечение, например, утилиты gdal2tiles и mapproxy-seed, TileMill, QTiles. Процесс создания тайлового кэша называется *сидированием* (seeding) и заключается в нарезке растров на каждом масштабном уровне на тайлы заданного размера. Данная процедура весьма требовательна к объёму жесткого диска и может занимать очень много времени. Например, время создания тайлового кэша на территорию Казахстана вплоть до 16 масштабного уровня (профиль global-mercator) при средней наполненности слоёв составляет 2-3 дня, поэтому на практике данный способы используется для подготовки кэшей карт небольших территорий или для создания кэшей определённых масштабных уровней.

Второй режим работы тайлового сервиса - динамический - предполагает, что никакого тайлового кэша нет, а запрашиваемые клиентом тайлы генерируются по запросу "на лету".

Условно, можно выделить ещё один режим работы тайлового сервиса - смешанный. В этом случае сервис, функционирующий в динамическом режиме, не только отдаёт клиенту сгенерированный тайл, но и сохраняет его в некоторый кэш, что позволяет при следующем запросе этого тайла, не создавать его вновь, а брать из кэша, что значительно повышает скорость работы сервиса в целом.

Чтобы разобраться с тем, как организована работа современных тайловых серверов (TileCache, MapProxy, tWMS) (фактически работающих в смешанном режиме), попытаемся написать собственный динамический TMS-сервер.

В качестве языка программирования будем использовать Python, операционная система - Debian GNU/Linux 7.0.

Архитектура динамического TMS-сервиса

Логику любого динамического TMS-сервиса можно упрощённо представить в виде следующей схемы:

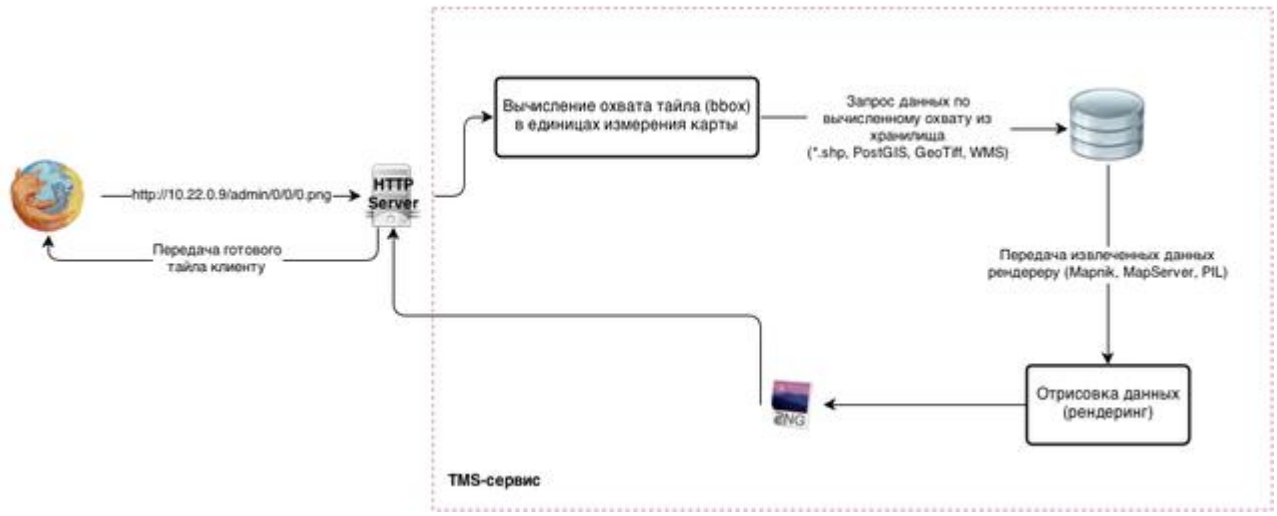


Схема работы динамического TMS-сервиса

Из данной схемы следует, что для создания сервиса необходимы инструменты, позволяющие следующее:

1. Извлекать информацию о координатах тайла из URL
2. По вычисленным координатам тайла получать его охват в единицах измерения карты
3. Извлекать данные из хранилища по переданному охвату
4. Отрисовывать (рендерить) извлечённые данные
5. Формировать HTTP ответ, содержащий готовый тайл

Каждая из этих задач может быть решена с помощью широкого спектра инструментов (библиотек), не связанных друг с другом. Например, процедуру извлечения данных из хранилища можно выполнить с помощью таких библиотек как GDAL/OGR или OWSLib, рендеринга - Mapnik или MapScript и т.д.

Выбор инструментов

Для решения 1 и 5 задачи подходит любой Веб-фреймворк. Поскольку наши требования к его функциональности достаточно скромные, то нет нужды использовать что-то вроде Django или Pyramid, вполне можно обойтись любым микрофреймворком, таким как Flask или Bottle. Остановимся на последнем.

Логотип микрофреймворка Bottle

2-я задача довольно простая, решим её самостоятельно без привлечения сторонних библиотек.

Для решения 3-й и 4-й задачи воспользуемся Mapnik-ом. Заметьте, что Mapnik в данном случае выступает не только в роли рендерера, но и в роли библиотеки, умеющей извлекать данные из хранилища. Полный список форматов, которые умеет читать Mapnik, доступен здесь. Mapnik вообще очень универсальный инструмент, позволяющий, в частности, отрисовывать объекты и без предоставления прямого доступа к хранилищу.

Логотип Mapnik

Установка и настройка Mapnik

Mapnik присутствует в репозиториях Debian GNU/Linux 7.0 и поэтому его установка сводится к выполнению следующей команды:

```
sudo aptitude install libmapnik2-2.0 mapnik-utils
```

Информацию об установке Mapnik на другие платформы можно найти на странице Mapnik Installation.

Чтобы убедиться в том, что Mapnik установился успешно, выполните команду (в используемой версии Debian модуль называется mapnik2, в других версиях или на других платформах может быть использовано имя mapnik):

```
python -c "import mapnik2"
```

Если в ответ на эту команду не последует никаких сообщение об ошибках, значит Mapnik установился корректно.

Mapnik поддерживает 2 способа конфигурирования - с помощью команд Python и с помощью специального *.xml файла. Основы конфигурирования Mapnik с помощью XML можно найти на странице Mapnik Getting

Started In XML. В рамках нашей задачи будем использовать именно этот подход.

Создадим файл mapnik-config.xml и поместим в него следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<Map srs="+init=epsg:32644" background-color="#b8dee6">
    <Style name="Admin style">
        <Rule>
            <PolygonSymbolizer fill="#ffffff"/>
            <LineSymbolizer stroke="#85c5d3" stroke-width="2" stroke-linejoin="round"
/>
        </Rule>
    </Style>
    <Style name="Ecoregions style">
        <Rule>
            <LineSymbolizer stroke="#85c5d3" stroke-width="0.5" stroke-linejoin="round"
/>
        </Rule>
    </Style>
    <Layer name="admin" srs="+init=epsg:4326">
        <StyleName>Admin style</StyleName>
        <Datasource>
            <Parameter name="dbname">geosample</Parameter>
            <Parameter name="geometry_field">the_geom</Parameter>
            <Parameter name="host">gis-lab.info</Parameter>
            <Parameter name="password">guest</Parameter>
            <Parameter name="table">(select the_geom from admin where id=2) as
admin</Parameter>
            <Parameter name="type">postgis</Parameter>
            <Parameter name="user">guest</Parameter>
        </Datasource>
    </Layer>
    <Layer name="ecoregions" srs="+init=epsg:4326">
        <StyleName>Ecoregions style</StyleName>
        <Datasource>
            <Parameter name="dbname">geosample</Parameter>
            <Parameter name="geometry_field">the_geom</Parameter>
            <Parameter name="host">gis-lab.info</Parameter>
            <Parameter name="password">guest</Parameter>
            <Parameter name="table">(select the_geom from ecoregions where
name='Алтайский край') as ecoregions</Parameter>
            <Parameter name="type">postgis</Parameter>
            <Parameter name="user">guest</Parameter>
        </Datasource>
    </Layer>
</Map>
```

Данный файл сообщает Mapnik-у, что наша карта будет состоять из двух слоёв набора Geosample - admin и ecoregiogions (только Алтайский край), расположенных в базе данных PostGIS, и отрисованных стилями Admin style и Ecoregions style, описанных здесь же в *.xml файле. Также обратите внимание, что мы перепроецируем исходные данные из географической системы координат в проекцию UTM зона 44 EPSG:32644 опять же средствами самого Mapnik-а.

Проверим, что наш *.xml файл составлен корректно. Для этого в той же директории создадим файл test.py и поместим в него следующий код:

```
import mapnik2 as mapnik
m = mapnik.Map(256,256)
mapnik.load_map(m, 'mapnik-config.xml')
m.zoom_all()
mapnik.render_to_file(m,'altay.png', 'png')
```

После чего выполним его:

```
python test.py
```

Представленный код считывает настройки Mapnik-а из файла mapnik-config.xml и отрисовывает карту в файл altay.png, расположенный в этой же директории. Откройте его (если операция выполняется на сервере без графического интерфейса, то просто сделайте симлинк в директорию /var/www/ и откройте изображение в браузере), он должен выглядеть следующим образом:

Пример карты, отрисованной с помощью Mapnik

Установка Bottle

Установим Bottle в виртуальное окружение:

```
sudo aptitude install python-virtualenv
cd ~
virtualenv --system-site-packages tms
source tms/bin/activate
pip install bottle
```

Подробнее об установки Bottle здесь.

Мы создали директорию tms с ключом *--system-site-packages*. Это сделано для того, чтобы в виртуальном окружении были доступны системные библиотеки, в частности Mapnik.

Установим HTTP-сервер Waitress:

```
pip install waitress
```

Создание собственного TMS-сервера

Переходим в директорию нашего виртуального окружения tms:

```
cd ~/tms
```

и создаём в ней файл tms.py, в котором и будет располагаться код будущего TMS-сервера.

Извлечение координат тайла из URL

В Bottle, как и в большинстве подобных фреймворков, присутствует удобный механизм диспетчеризации URL (роутинг), основанный на принципе сопоставления. Более подробную информацию можно найти в документации. Данная возможность позволяет по виду URL выбрать необходимую функцию, которая будет обрабатывать поступивший запрос, поэтому прежде всего нам нужно определится каким образом будет выглядеть URL запрашиваемого тайла. Согласно спецификации URL тайла должен выглядеть следующим образом:

```
/tms/1.0.0/mapname/z/x/y.png
```

где mapname - это произвольное имя карты, z - номер масштабного уровня, x - номер столбца в тайловой сетке, y - номер строки.

Открываем файл tms.py и пишем в него:

```
# -*- encoding: utf-8 -*-
from bottle import route, response, run
import mapnik2 as mapnik

@route('/<service>/1.0.0/altay/<z:int>/<x:int>/<y:int>.png')
def tms(z, x, y, service):

    # Остальной код будет располагаться здесь

run(host='0.0.0.0', port=8080, server='waitress')
```

Согласно представленному коду любой URL, имеющий шаблон:

```
/<service>/1.0.0/admin/<z:int>/<x:int>/<y:int>.png
```

будет обрабатываться функцией tms. В данном шаблоне угловыми скобками обозначены участки URL, которые могут иметь произвольное значение. Значения, указанные в угловых строках - это имена переменных, в которые будут попадать соответствующие фрагменты URL: service, z, x, y. То есть если на сервер придёт запрос вида:

```
http://localhost:8080/tms/1.0.0/altay/0/0/0.png
```

то в переменную service попадёт значение tms, в z, x и y - 0, причём координаты тайлов и номер масштабного уровня автоматически будут приведены к целочисленному типу (за что отвечает инструкция *:int*). Мы специально не стали жёстко прописывать тип сервиса tms в шаблон URL, для чего это было сделано станет понятно чуть позже.

Таким образом на данном этапе мы имеем функцию tms, внутри которой доступны значения масштабного уровня, координат тайла и типа сервиса.

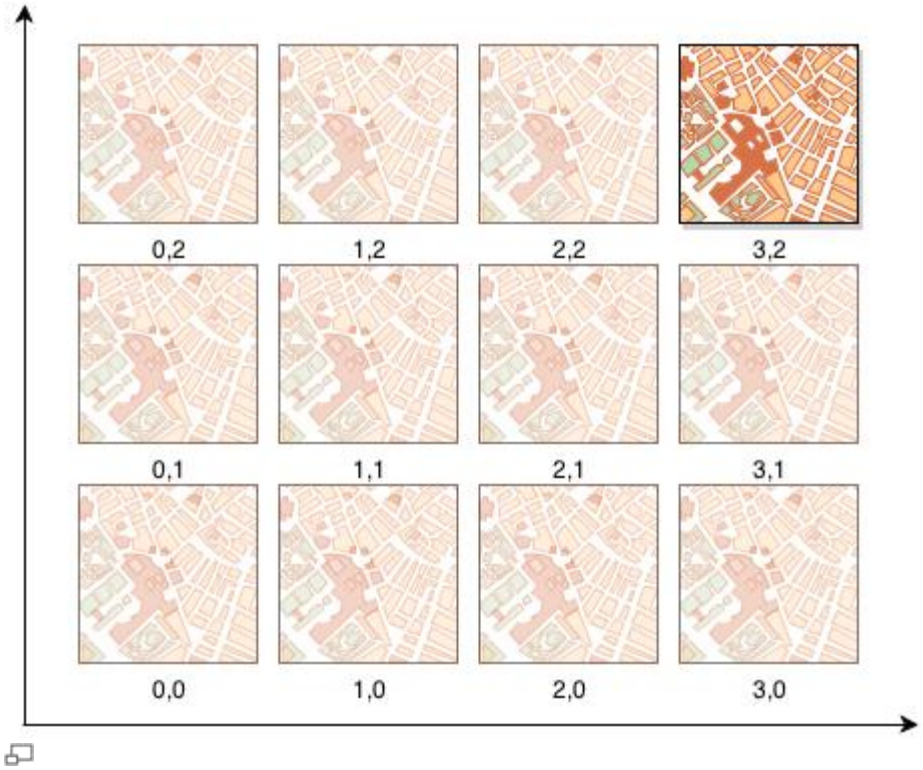
Получение охвата тайла в единицах измерения карты

Чтобы вычислить охват тайла в единицах измерения карты, вначале нужно определить размеры самого тайла. Согласно статье Основы конфигурирования тайловых сеток размер одного тайла на z-м масштабном уровне равен:

```
step = max(W,H)/2^z
```

где W и H - ширина и высота карты соответственно (в единицах измерения используемой проекции).

Зная информацию о координатах начала отсчёта тайловой сетки (minx, miny) легко посчитать охват любого тайла в единицах измерения карты. Предположим, мы хотим определить охват тайла (3,2). Для наглядности воспользуемся следующим рисунком:



Запрашиваемый тайл в TMS-совместимой сетке

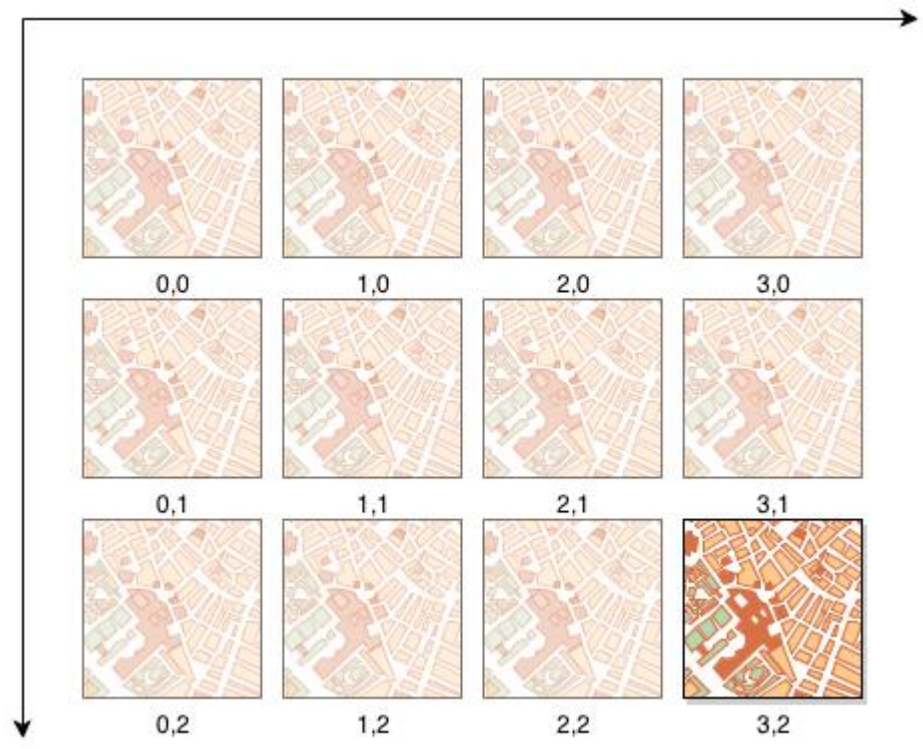
Тогда искомый охват запишется следующим образом:

```
(minx + step*3, miny + step*2, minx + step*(3+1), miny + step*(3+1))
```

Или в общем виде для тайла (i,j):

```
(minx + step*i, miny + step*j, minx + step*(i+1), miny + step*(j+1))
```

Для тайловой сетки с перевёрнутой осью y, имеющей начало отсчёта в точке (minx, maxy) охват тайла будет вычисляться несколько иначе:



Запрашиваемый тайл в тайловой сетке с перевёрнутой осью y (как в OpenStreetMap)

```
(minx + i*step, maxy - (j+1)*step, minx + (i+1)*step, maxy - j*step)
```

Охват карты, которую мы будем рендерить нашим сервисом (можно подсмотреть в статье Основы конфигурирования тайловых сеток):

```
(287157, 5613155, 920220, 6045880)
```

Отразим всё выше сказанное в виде Python-кода, поместив его внутрь функции tms:

```
bbox = dict(minx=287157, miny=5613155, maxx=920220, maxy=6045880)

step = max(bbox['maxx']-bbox['minx'], bbox['maxy']-bbox['miny']) / 2**z

extents = dict()

extents['tms'] = (
    bbox['minx'] + x*step, bbox['miny'] + y*step,
    bbox['minx'] + (x+1)*step, bbox['miny'] + (y+1)*step
)

extents['xyz'] = (
    bbox['minx'] + x*step, bbox['maxy'] - (y+1)*step,
    bbox['minx'] + (x+1)*step, bbox['maxy'] - y*step
)
```

Как видно из представленного кода, мы сформировали словарь extents, который содержит значение охвата тайла в зависимости от типа тайловой сетки (tms - TMS-совместимой, xyz - как в OpenStreetMap).

Отрисовка тайла

После того как мы вычислили интересующий нас охват, мы должны извлечь интересующие данные из хранилища. Но поскольку мы решили использовать Mapnik, который умеет напрямую работать с хранилищем PostGIS, то этот этап будет выполняться автоматически "за кадром". Добавляем следующий фрагмент кода в нашу функцию tms:

```
tile = dict(width=256, height=256)

m = mapnik.Map(tile['width'], tile['height'])

# Указываем путь, где находится файл с настройками Mapnik
mapnik.load_map(m, '/home/rykovd/mapnik-config.xml')

# Выбираем охват тайла из словаря extents согласно типу запрашиваемого сервиса (tms или xyz)
box = mapnik.Box2d(*extents.get(service))

# Отрисовываем тайл
m.zoom_to_box(box)
im = mapnik.Image(m.width, m.height)
mapnik.render(m, im)
output = im.tostring('png')
response.content_type = 'image/png'

# Передаём ответ клиенту
return output
```

Запуск сервера

Запускаем наш сервер:

```
(tms)rykov@extensa:~/tms$ python tms.py
Bottle v0.11.6 server starting up (using WaitressServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.
```

```
serving on http://0.0.0.0:8080
```

Результаты работы

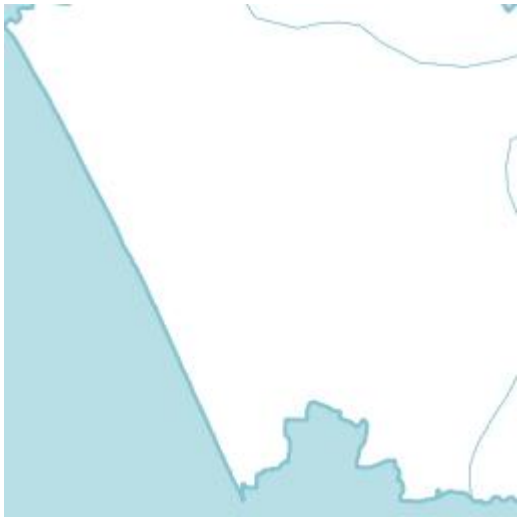
Запрашиваем тайлы по различным URL и смотрим, что получилось.

TMS-сетка

- 

  http://localhost:8080/tms/1.0.0/altay/1/0/1.png

http://localhost:8080/tms/1.0.0/altay/1/1/1.png

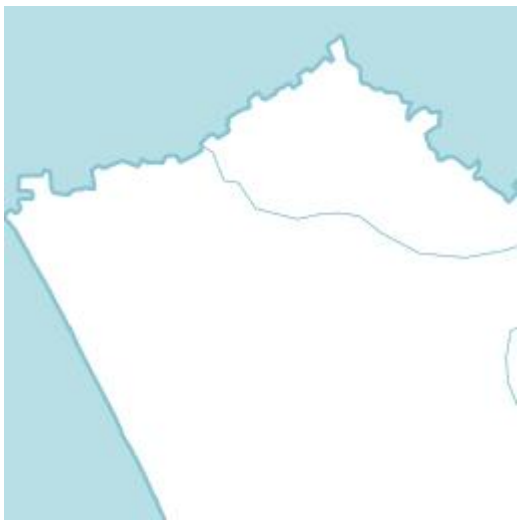


http://localhost:8080/tms/1.0.0/altay/1/0/0.png

- 

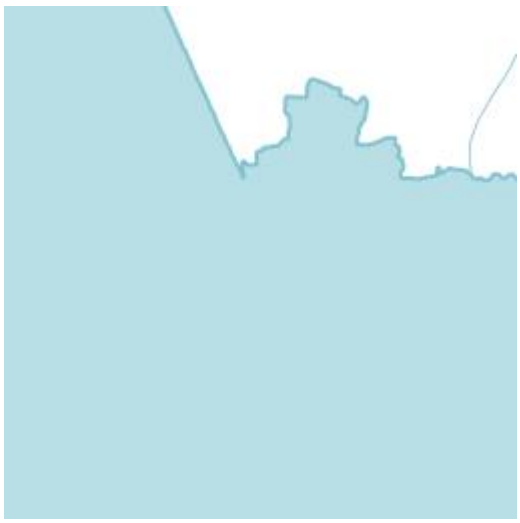
http://localhost:8080/tms/1.0.0/altay/1/1/0.png

OpenStreetMap-сетка

- 

http://localhost:8080/xyz/1.0.0/altay/1/0/0.png

http://localhost:8080/xyz/1.0.0/altay/1/1/0.png

- 

http://localhost:8080/xyz/1.0.0/altay/1/0/1.png

- 

http://localhost:8080/xyz/1.0.0/altay/1/1/1.png

Заключение

Таким образом, мы написали свой собственный примитивный TMS-сервер в котором пока нет ни возможности кэширования тайлов, ни многих других важных функций, но который наглядно отражает подходы, используемые современным ПО данного класса.

Обсудить в форуме Комментариев — 13

Последнее обновление: 2014-05-15 01:46

Дата создания: 06.04.2013
Автор(ы): Denis Rykov