Создание простейшего HTTP-сервиса для публикации векторных данных

Обсудить в форуме Комментариев — 15

В статье представлен простейший вариант реализации HTTP-сервиса, позволяющий клиенту запрашивать данные на определенный охват в указанной системе координат, некий упрощенный вариант запроса GetFeature, определенного в рамках OGC Web Feature Service.

Содержание

Введение

При разработке Веб-ГИС зачастую возникает необходимость в передаче векторных данных с сервера на клиентскую сторону. При этом, как правило, векторные данные имеют большой объем или находятся в различных хранилищах и поэтому требуется некоторое промежуточное звено, которое бы могло получать от клиентского приложения запрос данных, удовлетворяющих определенному критерию (например, попадающие в некий охват), подключаться к хранилищу данных, извлекать нужный набор и возвращать его клиенту.

Для решения данной задачи существует специальный протокол OGC Web Feature Service, о котором мы рассказывали в одной из предыдущих статей. С одной стороны, опубликовав свои данные по WFS, мы получаем очень богатые возможности:

- отображение данных в настольных ГИС;
- гибкий язык описания фильтров;
- возможность редактирования данных.

Если бы перед нами стояла задача просто опубликовать некоторый набор данных по WFS, то никаких проблем - берем TinyOWS, MapServer или GeoServer, настраиваем подключение к хранилищу - всё. Но это не наш случай, нам нужно добавить WFS-функционал непосредственно в наше приложение. Как поступить в этом случае? Вариантов на самом деле не много: либо найти необходимую библиотеку на том языке программирования, на котором ведется разработка, либо каким-то образом "прикрутить" имеющийся WFS-

сервер к нашему приложению. Не беремся утверждать за другие языки программирования, но, например, в случае Python не существует нативных библиотек, реализующих функционал WFS-сервиса, только лишь обертки к mapscript, а это по сути сводится к установке библиотек MapServer (который написан на C/C++). Завязывание же своего приложения на внешний по отношению к нему WFS-сервер - задача непростая и тянущая за собой ряд очень серьезных проблем:

- необходимость подготовки дистрибутива WFS-сервера под целевую платформу;
- язык программирования, используемый при разработке основного приложения и WFS-сервера зачастую не совпадают, следовательно в случае возникновение проблем с последним, необходимо привлечение дополнительных ресурсов;
- сама задача по интеграции приложение с WFS-сервером далеко не тривиальная.

Поэтому, если вы все-таки хотите использовать возможности WFS в своем приложении, то наиболее правильным видится подход в разработке соответствующего ПО на том же языке программирования, что используется в вашем приложении. В этом случае вы имеете полный контроль над своей системой.

Как показывает практика, для разработки небольших картографических Веб-приложений функционал, предлагаемый спецификацией WFS, крайне избыточен. Зачастую оказывается ненужным ни отображение данных в настольных ГИС, ни гибкий язык описания фильтров, поэтому в большинстве случаев достаточно написания собственного простейшего HTTP-сервиса, возвращающего данные в каком-нибудь стандартном формате (например,GeoJSON). Именно решению данной задачи и будет посвящена оставшаяся часть статьи. В качестве примера использования подобного сервиса может служить проект Найди участкового, откройте консоль вашего браузера и посмотрите, какие запросы уходят на сервер и какие ответы приходят от него при сдвиге карты. По виду URL мы можем однозначно сказать, что клиент и сервер взаимодействуют друг с другом, не используя протокол WFS, вместо этого используется некий самописный HTTP-сервис со своим API.

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

Выбор API

Если вам требуется разработать полнофункциональный HTTP-сервис, то, чтобы не изобретать велосипед, можно взять готовое описание какого-нибудь API (в зависимости от задач) и реализовать его. Например, MapFish Protocol (реализован в Papyrus) или ArcGIS Server REST API (открытая реализация HTTP-сервиса, реализующая данный API уже существует).

В нашем случае мы разработаем HTTP-сервис, поддерживающий 3 GET-параметра:

- bbox={xmin,ymin,xmax,ymax} - запрашиваемый bbox (по умолчанию считается, что координаты указаны в системе координат EPSG:4326);
- epsg={num} - система координат в которой должны быть возвращены данные, также используется как система координат для bbox;
- attrs={field1},{field2},... - имена запрашиваемых атрибутивных полей.

Хранилище и формат выходных данных

В качестве данных возьмем БД PostGIS, созданную в рамках проекта по созданию слоя детских учреждений. Выходной формат - GeoJSON.

При разработке Веб-ГИС стандартная практика заключается в передаче данных клиенту в формате GeoJSON, при этом объект GeoJSON - это объект типа "FeatureCollection" - коллекция элементарных объектов. Объект типа "FeatureCollection" содержит одно свойство "features", значение данного свойства – массив, каждый элемент которого представляет собой элементарный объект.

В PostgreSQL, начиная с версии 9.2, появились специальные функции для работы с JSON-данными, в частности row_to_json, позволяющие, используя SQL-запрос к базе данных PostGIS, получить "FeatureCollection". О том как это сделать подробно описано в статье Creating GeoJSON Feature Collections with JSON and PostGIS functions, мы же, ввиду того что используемое нами хранилище развернуто на PostgreSQL 9.1, будем формировать "FeatureCollection" самостоятельно в коде нашего сервиса.

Веб-фреймворк

Для того, чтобы облегчить себе жизнь и не писать служебный код, в качестве инструмента для работы с HTTP-запросами выберем какой-нибудь Веб-фреймворк. Остановимся на Bottle. Данный фреймворк уже использовался нами в статье Основы работы динамических TMS-сервисов.

Установка программного обеспечения

Установка Bottle

Установим Bottle в виртуальное окружение:

```
sudo aptitude install python-virtualenv
cd ~
virtualenv --no-site-packages geoservice
source geoservice/bin/activate
pip install bottle
pip install waitress
```

Установка psycopg2

```
pip install psycopg2
```

В случае возникновения ошибок при установке требуется установка необходимых библиотек на уровне операционной системы, подробнее.

Установка bottle-pgsql

```
pip install bottle-pgsql
```

bottle-pgsql - это плагин, автоматизирующий работу с PostgreSQL в Bottle-приложение. При каждом запросе он автоматически подключается к базе данных, передаёт в функцию обработки запроса ее дескриптор и в самом конце закрывает соединение. Результаты выполнения запросов представляются в виде словарей, а не в виде кортежей, как это происходит по умолчанию в psycopg2.

Установка библиотеки для парсинга конфигурационного файла

```
pip install pyyaml
```

Создание сервиса

Переходим в директорию нашего виртуального окружения geoservice:

```
cd geoservice
```

и создаём в ней файл geoservice.py, в котором и будет располагаться код будущего HTTP-сервиса, также здесь поместим файл, содержащий описание подключения к нашей базе данных config.yaml:

```
touch geoservice.py
touch config.yaml
```

Открываем файл config.yaml и помещаем в него следующее содержимое:

```
db:
  host: gis-lab.info
  port: 5432
  name: geodetdom
  table: data
  user: guest
  password: guest
  geometry_column: geometry
```

Затем открываем файл geoservice.py и пишем в него:

```
# -*- encoding: utf-8 -*-
import json
import bottle_pgsql
from bottle import install, route, response, request, run, static_file
from yaml import load

# Остальной код будет располагаться тут

if __name__ == "__main__":
    run(host='0.0.0.0', port=8087, server='waitress')
```

Подключение к базе данных

Извлекаем информацию о подключении из конфигурационного файла:

```
db_config_file = open('config.yaml', 'rb')
db_config = load(db_config_file).get('db')
conn_string = 'host=%(host)s dbname=%(name)s user=%(user)s password=%(password)s'
conn_string %= db_config
```

И подключаем плагин bottle-pgsql:

```
plugin = bottle_pgsql.Plugin(conn_string)
install(plugin)
```

Извлечение данных из базы

Опишем функцию, которая будет извлекать значения GET-параметров из URL, осуществлять запрос к базе данных и передавать данные обратно клиенту:

```
@route('/features')
def geoservice(db, bbox='-180,-90,180,90', epsg='4326', limit='100'):
        # Параметры запроса
        bbox = (request.GET.get('bbox') or bbox).split(',')
        epsg = request.GET.get('epsg') or epsg
        attrs = request.GET.get('attrs', [])
        coordinates = dict(xmin=bbox[0], ymin=bbox[1], xmax=bbox[2], ymax=bbox[3])
        geometry_column = db_config.get('geometry_column')
        table = db_config.get('table')

        # Строка запроса
        query = """select st_asgeojson(st_transform({gc}, {epsg})) as g, *
              from {table}
              where st_intersects(st_transform({gc}, {epsg}),
st_makeenvelope({xmin},{ymin},{xmax},{ymax},{epsg}))
              limit {limit};
              """.format(gc=geometry_column, epsg=epsg, table=table, limit=limit,
**coordinates)

        # Выполнение запроса
        db.execute(query)
        records = db.fetchall()

        # Формируем GeoJSON вручную
        collection = {'type': 'FeatureCollection', 'features': []}
        for rec in records:
            feature = dict()
            feature['type'] = 'Feature'
            feature['properties'] = dict()

            # Итерация по именам полей
            for colname in rec.keys():
                if colname == 'g':
                    feature['geometry'] = json.loads(rec[colname])
```

```
                    continue
                if colname in attrs:
                    feature['properties'][colname] = rec[colname]
            collection['features'].append(feature)

        response.content_type = 'application/json'
        return json.dumps(collection)
```

Запуск сервиса

```
$ python geoservice.py
Bottle v0.11.6 server starting up (using WaitressServer())...
Listening on http://0.0.0.0:8087/
Hit Ctrl-C to quit.

serving on http://0.0.0.0:8087
```

Результаты

Пришло время проверить, что же получилось. Откроем браузер и протестируем следующие URL.

```
http://localhost:8087/features?bbox=4180824.1850282,7488851.5571727,4187192.3449474,749
0776.8148227&epsg=3857&attrs=post,name
```

Результат:


```
{
    "type": "FeatureCollection",
    "features": [
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    4184578.0278406707,
                    7489606.364854654
                ]
            },
            "type": "Feature",
            "properties": {
                "post": "117303, Москва, ул. Каховка, 2",
                "name": "Школа-интернат № 24 дя детей-сирот"
            }
        },
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    4186044.5508123846,
                    7488978.3756106505
                ]
            },
            "type": "Feature",
            "properties": {
                "post": "117452, Москва, Симферопольский б-р, 20",
                "name": "Школа-интернат № 95 общеобразовательная"
            }
        }
    ]
}
```


Как можно увидеть получены данные, удовлетворяющие заданному охвату в системе координат EPSG:3857.

```
http://localhost:8087/features?bbox=82,53,83,54&attrs=post,name
```

Результат:

```
{
    "type": "FeatureCollection",
    "features": [
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    82.995135,
                    53.320683
                ]
            },
            "type": "Feature",
            "properties": {
                "post": "659000, Алтайский край, с.Павловск, ул. Шумилова, 1",
                "name": "Павловский детский дом"
            }
        },
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    82.984438,
                    53.311395
                ]
            },
            "type": "Feature",
            "properties": {
                "post": "659000, Алтайский край, с.Павловск, ул. Коминтерна, 2",
                "name": "Павловская школа-интернат спец.корр."
            }
        },
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    82.470054,
                    53.443182
                ]
            },
            "type": "Feature",
            "properties": {
                "post": "659053, Алтайский край, Шелаболихинский р-н, с.Кучук, ул.
Новая, 15",
                "name": "Кучукский детский дом"
            }
        },
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    82.316406,
                    53.784103
                ]
            },
            "type": "Feature",
            "properties": {
                "post": "633623, Новосибирская обл., п. Сузун, ул. Толстого, 11",
                "name": "Социальный приют для детей и подростков \"Лесовичек\""
            }
        },
        {
            "geometry": {
                "type": "Point",
                "coordinates": [
                    82.316406,
```

```
                    53.784103
                ]
            },
            "type": "Feature",
            "properties": {
                "post": "633610, Новосибирская обл., п. Сузун, ул. Партизанская, 19",
                "name": "Сузунская школа-интернат для детей-сирот 8ого вида"
            }
        }
    ]
}
```

Для представления JSON-данных в удобочитаемом варианте можно воспользоватся сервисом JSONLint, что мы собственно и сделали.

Визуализация данных в OpenLayers

По умолчанию используемый нами HTTP-сервер не умеет раздавать статику. Чтобы это ис, откроем файл geoservice.py и добавим следующую функцию (в тот же уровень вложенности, на котором располагается функция geoservice):

```
@route('/<filename:re:.*\.html>')
def htmls(filename):
    return static_file(filename, root='/home/rykovd/projects/geoservice/static')
```

Данная функция будет возвращать любой документ с расширением *.html, находящийся в каталоге root.

В каталоге нашего виртуального окружения geoservice создадим директорию static и поместим в нее файл client.html, в котором будет располагаться код нашего Веб-клиента:

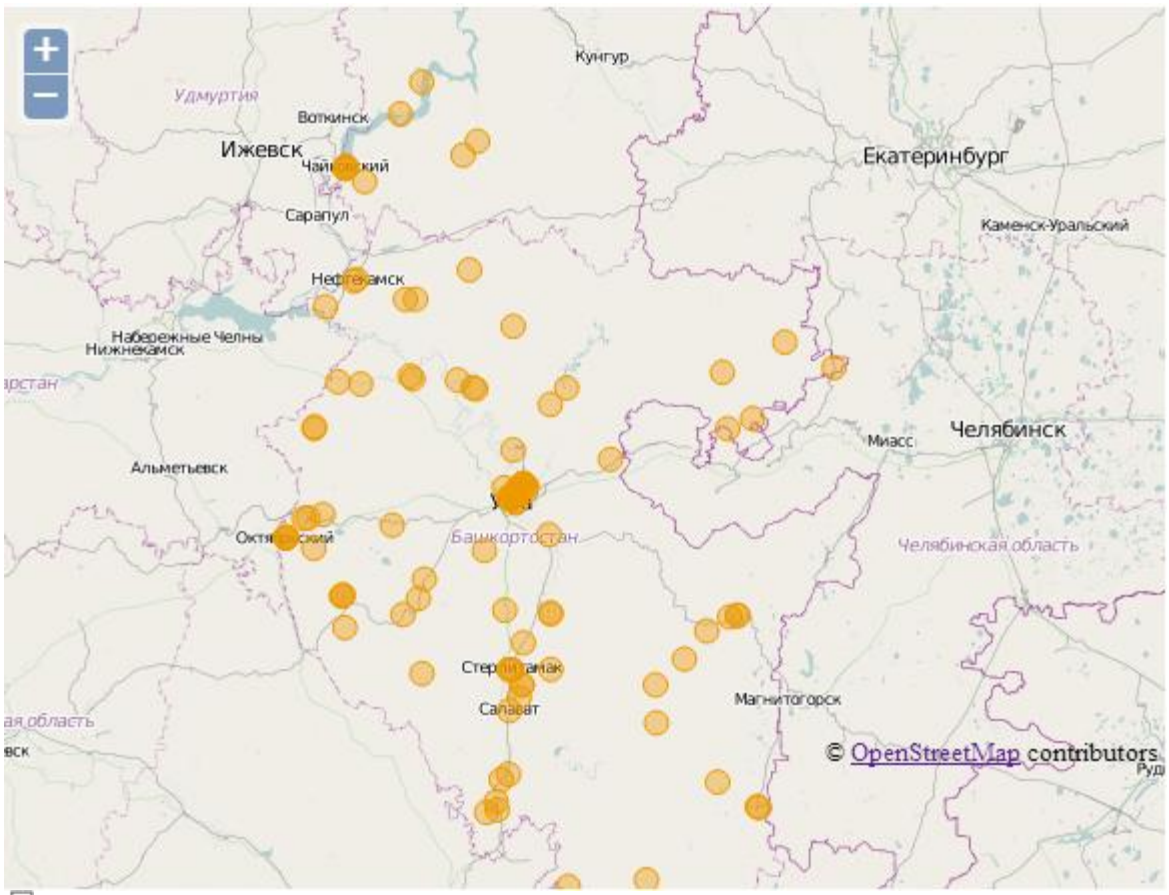
```
mkdir static
cd static
touch client.html
```

Открываем файл client.html и вставляем в него следующий код:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Geodetdom</title>
<script src="http://openlayers.org/dev/OpenLayers.js"></script>

<script type="text/javascript">
function init(){

    var options = {
        div: "map",
        layers: [new OpenLayers.Layer.OSM()]
    };
    var  map = new OpenLayers.Map(options);

    geodetdom = new OpenLayers.Layer.Vector("Детские дома", {
        projection: new OpenLayers.Projection('EPSG:3857'),
        strategies: [new OpenLayers.Strategy.BBOX({ratio: 1, resFactor: 1})],
        protocol: new OpenLayers.Protocol.HTTP({
            url: "http://localhost:8087/features",
            params: {
                attrs: 'id,name',
                epsg: '3857'
            },
            format: new OpenLayers.Format.GeoJSON()
        })
```

```
    });
    map.addLayer(geodetdom);
    map.zoomToExtent([3062373.1007906, 8962088.6911328, 18560133.457509,
9930698.7134278]);
}
</script>
</head>
<body onload="init()">
    <div id="map" style="width:100%; height:100%;"></div>
</body>
</html>
```

В данном примере использована стратегия OpenLayers.Strategy.BBOX, что означает, что данные будут запрашиваться при каждом изменении охвата карты. Опция resFactor, выставленная в значение 1, позволяет нам запрашивать данные даже в том случае, если новый охват попадает в предыдущий (например, при изменении масштаба карты), но с учетом того, что изменилось разрешение базового слоя. Стратегия BBOX автоматически создает фильтр слоя OpenLayers.Filter.Spatial.BBOX, что приводит к тому, что при запросе данных к URL автоматически добавляется GET-параметр bbox с указанием значения текущего охвата, что совпадает с именем GET-параметра, ответственного за охват, используемого в нашем сервисе.



Результат визуализации данных HTTP-сервиса в OpenLayers

Если открыть консоль Веб-браузера и ввести следующие команды:

```
geodetdom.features.length
100
geodetdom.features1.attributes
Object {id: 8547, name: "Школа-интернат № 1 "}
```

то можно увидеть, что количество объектов слоя не превосходит лимита, заданного на уровне сервиса (100 объектов), а также что, объекты содержат только запрошенные атрибуты.

Заключение

Таким образом, мы создали простейший HTTP-сервис, который на входе принимает определенный набор параметров и возвращает результат в формате GeoJSON. И хотя это всего-лишь учебный вариант, в котором не предусмотрена никакая обработка ошибок, но он наглядно показывает как может быть устроен простейший сервис подобного типа. Данные, предоставляемые этим сервисом, легко могут быть визуализированы с помощью картографического JavaScript-клиента, например, OpenLayers.

Обсудить в форуме Комментариев — 15

Последнее обновление: 2014-05-15 01:47

Дата создания: 13.06.2013
Автор(ы): Denis Rykov