Рендеринг векторных данных в Mapnik без предоставления прямого доступа к хранилищу

Обсудить в форуме Комментариев — 2

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

Содержание

Введение

Mapnik содержит широкий список плагинов, обеспечивающих поддержку различных источников векторных данных, в том числе PostGIS и OGR. Однако в ряде ситуаций использование прямого подключения Mapnik к хранилищу оказывается не совсем удачным решением: например, нам требуется произвести какую-либо предварительную обработку объектов перед рендерингом или вообще может оказаться так, что наш источник данных не поддерживается Mapnik. В этом случае универсальным решением является следующий подход: запросить необходимые объекты из хранилища средствами того языка программирования, на котором ведётся работа (в нашем случае это будет Python) и передать их рендеру.

Данную задачу можно решить путём использования класса MemoryDatasource или плагина Python.

MemoryDatasource

Этот способ более простой, но имеет существенный недостаток - размер векторных данных, помещённых в MemoryDatasource ограничен размером доступной оперативной памяти.

Допустим, что у нас есть некоторый массив векторных данных в формате WKT, которые мы хотим отрендерить. Для представления векторных данных в Mapnik существует класс Feature, конструктор которого принимает два аргумента: объект Context (предназначен для оптимизации структуры хранения атрибутивных данных) и уникальный идентификатор векторного объекта. Создаём объекты класса *Feature* и помещаем их в *MemoryDatasource*:

```
import mapnik

# Массив векторных данных
geometries = [
    (0, "POLYGON ((0 0, 5 0, 5 5, 0 5, 0 0))"),
    (1, "POLYGON((1 5, 4 5, 4 6, 1 6, 1 5))")
]

# Создаём локальный источник данных
ds = mapnik.MemoryDatasource()

# Объект контекста
ctx = mapnik.Context()

# Заполняем локальный источник данных
```

```
for id, wkt in geometries:
    f = mapnik.Feature(ctx, id)
    f.add_geometries_from_wkt(wkt)
    ds.add_feature(f)
```

После чего можно использовать этот источник данных для любого слоя Mapnik:

```
layer = mapnik.Layer('memory')
layer.datasource = ds
```

Отметим, что объект класса Feature имеет не только метод *add_geometries_from_wkt* , но и *add_geometries_from_wkb* , что позволяет добавлять геометрии в формате WKB.

Плагин Python

Как было отмечено выше - подход к рендерингу векторных данных с использованием *MemoryDatasource* обладает определёнными недостатками. Во первых, все данные должны одновременно находиться в памяти, во-вторых - на уровне локального источника данных *datasource* нет возможности запрашивать данные на определённый охват, то есть мы, конечно, можем каждый раз создавать новый *datasource*, подавая на вход набор векторных объектов, осуществив предварительно необходимый запрос к хранилищу средствами Python, но использование плагина *Python* предполагает более элегантное решение приведённых проблем.

Плагин *Python* позволяет создать *datasource*, который будет возвращать объекты, попавшие в охват, а также эффективно использовать память. Выглядит это следующим образом - при рендеринге данных Mapnik обращается в *datasource*, передавая тому в качестве параметра некоторый охват. *datasource* в свою очередь возвращает итератор. Mapnik получает данный итератор, извлекает из него первый объект класса Feature, рендерит его, после чего объект удаляется из памяти и осуществляется переход к следующему, до тех пор пока итератор не вернёт исключение *StopIteration*, являющееся признаком окончания процесса итерации.Таким образом, при использовании плагина *Python* объём оперативной памяти не должен превышать размера самого "тяжёлого" векторного объекта.

Инициализация *datasource* в этом случае осуществляется следующим образом:

```
layer = mapnik.Layer('python')
ds = mapnik.Python(factory='DatasourceClass')
layer.datasource = ds
```

Для инициализации плагина требуется указать один единственный именованный аргумент *factory*. В качестве *factory* может выступать любая вызываемая сущность (callable), которая должна возвращать объект, имеющий следующие атрибуты:

1. *envelope* - объект класса mapnik.Box2d, охват *datasource*;
2. *data_type* - объект класса mapnik.DataType, определяет тип *datasource* (растровый или векторный);
3. *geometry_type* (опционально) - объект класса DataGeometryType, определяет тип геометрий (точка, линия, полигон, коллекция).

И два обязательных метода:

1. *features(query)* - функция, принимающая на вход объект класса mapnik.Query и возвращающая итератор, выдающиё объекты класса mapnik.Feature;
2. *features_at_point(point)* - функция, принимающая на вход объект класса mapnik.Coord и возвращающая итератор объектов mapnik.Feature внутрь которых попала указанная точка.

В качестве *factory* обычно используется отдельный класс, для удобства создания которого существует PythonDatasource, от которого можно наследоваться. *PythonDatasource* удовлетворяет всем необходимым требованиям к объекту *factory*, более того, он принимает аргументы *envelope*, *geometry_type* и *data_type* в качестве аргументов конструктора (envelope учитывается при рендеринге, а geometry_type и data_type - имеют справочное значение).

Метод *features_at_point* используется довольно редко, основное взаимодействие рендера с *datastore*

осуществляется посредством метода *features*. Очевидно, что конкретная реализация *features* в каждом случае будет своей, то есть её нужно писать отдельно.

Рассмотрим небольшой пример:

```
import mapnik

class TestDatasource(mapnik.PythonDatasource):
    def __init__(self):
        super(TestDatasource, self).__init__(envelope=mapnik.Box2d(0,0,6,6))

    def features(self, query):
        features = (
            (1, "POLYGON((0 0, 5 0, 5 5, 0 5, 0 0))"),
            (2, "POLYGON((1 5, 4 5, 4 6, 1 6, 1 5))"),
        )
        ctx = mapnik.Context()
        for id, wkt in features:
            f = mapnik.Feature(ctx, id)
            f.add_geometries_from_wkt(wkt)
            yield f
...
ds = mapnik.Python(factory='TestDatasource')
layer = mapnik.Layer('python')
layer.datasource = ds
```

Данный пример иллюстрирует идею плагина *Python* - метод *features* должен возвращать итерируемый объект (в данном примере метод *features* реализован в виде генератора).

Чтобы не формировать объекты класса mapnik.Feature вручную в PythonDatasource существует 2 удобных метода класса wkb_features и wkt_features.

Перепишем наш метод features, используя *mapnik.PythonDatasource.wkb_features* :

```
def features(self, query):
    return mapnik.PythonDatasource.wkb_features(
        keys = (),
        features = (
            ("POLYGON((0 0, 5 0, 5 5, 0 5, 0 0))", {}),
            ("POLYGON((1 5, 4 5, 4 6, 1 6, 1 5))", {}),
        )
    )
```

mapnik.PythonDatasource.wkt_features - представляет собой обёртку, возвращающую итератор, который возвращает объекты класса *mapnik.Features*. В данном примере векторные объекты жестко прописаны в коде, однако в качестве аргумента *features* метода *mapnik.PythonDatasource.wkt_features* может выступать любой итерируемый объект или генератор, например:

```
def get_features(self, query):
    features = (
        ("POLYGON((0 0, 5 0, 5 5, 0 5, 0 0))", {}),
        ("POLYGON((1 5, 4 5, 4 6, 1 6, 1 5))", {}),
    )
    for f in features:
        yield f

def features(self, query):
    return mapnik.PythonDatasource.wkb_features(
        keys = (),
        features = self.get_features(query)
    )
```

Заметим, что мы никак не учитывали значение аргумента *query*, что в реальных задачах недопустимо. Также в

реальных условиях никаких объектов, заданных вручную нет - функционал запроса данных из хранилищ выполняется в виде отдельных функций, возвращающих итераторы.

Ссылки

1. Python plugin

Обсудить в форуме Комментариев — 2

Последнее обновление: 2014-05-15 01:45

Дата создания: 27.12.2012
Автор(ы): Денис Рыков