Создание WMS сервера на базе GRASS GIS и Pyramid

Обсудить в форуме Комментариев — 8

В данной статье описывается простой пример создания WMS сервера на базе геоинформационной системы GRASS. Доступ к данным организован при помощи фреймворка Pyramid.

Данная статья является расширенным переводом статьи GRASS GIS Web Map Service with Pyramid.

Содержание

Общие сведения

Статья расчитана на читателя, который на базовом уровне знаком с геоинформационной системой GRASS и основами веб-разработки на языке программирования Python. В частности, читатель должен представлять, как происходит вызов команд GRASS на языке Python и быть знакомым с принципами использования фреймворка Pyramid.

В статье используется GRASS версии 6.4. Приведенный код применим и к более новым версиям GRASS, но, возможно, потребуется его незначительная правка.

Исходный код модуля описанный в статье доступен по ссылке: Файл:Grass wms.zip.

Архитектура системы

В разрабатываемой системе вся обработка геоданных будет возложена на GRASS, а обработка веб-запросов --- на Pyramid. Основная часть WMS сервера, отвечающая за отрисовку геоданных --- модуль d.mon, который будет использоваться совместно с графическим драйвером Cairo, позволяющим генерировать изображения в форматах PNG, BMP, PPM, PS, PDF и SVG.

Для реализации сервера потребуется создать функции, служащие для:

1. Настройки путей к GRASS.
2. Получения и анализа параметров запроса WMS.
3. Отрисовки слоев по запросу клиента и сохранения результата в графический файл.
4. Возвращения результата обработки запроса клиенту.

Собственно к GRASS относятся первый и третий пункты, остальные --- обычные действия по обработке веб-запросов, которые реализуются при помощи Pyramid. Поэтому для простоты изложения и реализации объединим первый и третий пункты в виде одной функции, а второй и четвертый --- в виде другой.

Зависимости и их установка

Для того, чтобы WMS сервер мог функционировать, требуется, чтобы на компьютере была установлена геоинформационная система GRASS и Pyramid (Pyramid рекомендуется устанавливать в виртуальное окружение, чтобы не "мусорить" в основной системе).

Установка этих систем обеспечит установку и настройку требуемых зависимостей.

Реализация

Настройка путей GRASS и обработка запроса клиента

Ниже приводится код функции, которая инициализирует GRASS, отрисовывает запрошенные пользователем слои и сохраняет полученное изображение во временном файле.

На вход функция принимает

- список слоев (layers), которые требуется отрисовать;
- охват интересующего пользователя региона (bbox);
- ширину/высоту (width/height) изображения.

На выходе функция возвращает имя файла, в котором будет сохранено запрошенное изображение.

```
def _grass_wms(layers=, bbox=, width=256, height=256):
   # Прописываем пути к GRASS. Ис под свои нужды.
   gisdbase = "/home/useranme/GRASSDATA"
   location = "MSK_28407"
   mapset = "PERMANENT"
   gisbase = os.environ"GISBASE" = "/usr/lib/grass64"

   # Добавляем путь модулям GRASS в системные пути
   sys.path.append("%s/etc/python" % gisbase)

   # Импортируем библиотеки GRASS
   from grass.script import core as grass
   from grass.script import setup as gsetup

   # Инициализируем GRASS
   gsetup.init(gisbase, gisdbase,
               location, mapset)

   # Получаем список известных растровых и векторных слоев
   vector_layers = grass.list_strings(type = "vect")
   raster_layers = grass.list_strings(type = "rast")

   grass.run_command("g.region", w=bbox0, s=bbox1, e=bbox2, n=bbox3)

   # Создание временного файла
   tempfile = grass.tempfile()
   filename = "%s.png" % tempfile

   # Настраиваем параметры вывода
   os.environ"GRASS_CAIROFILE" = filename
   os.environ"GRASS_WIDTH" = width
   os.environ"GRASS_HEIGHT" = height

   # Отрисовка
   grass.run_command("d.mon", start="cairo")

   for layer in layers:
```

```
        if layer in raster_layers:
            grass.run_command("d.rast", map=layer, quiet=1)
        elif layer in vector_layers:
            grass.run_command("d.vect", map=layer, quiet=1, fcolor="0:0:255", color=None)

    grass.run_command("d.mon", stop="cairo")

    return filename
```

Как видим, большая часть кода связана с первоначальной настройкой GRASS. Основной код по отрисовке запрошенных слоев использует команды GRASS и занимает всего пять строк:

```
 for layer in layers:
 ...
```

При этом все векторные слои будут отрисованы синим цветом, (см. параметры, передаваемые в `grass.run_command("d.vect", map=layer, quiet=1, fcolor="0:0:255", color=None)`). Внешний вид слоя может быть изменен настройкой параметров команды d.vect. Например, данной командой можно назначить стиль отображения объекта в зависимости от его атрибутов или геометрических характеристик.

Аналогично, растровые слои отображаются в цветовой схеме по умолчанию (или заранее назначенной пользователем вручную). При необходимости изменить стиль слоя при отрисовке следует воспользоваться командами, обрабатывающими цветовые схемы растровых карт, например r.colors или аналогичными модулями.

Временный файл, в который сохраняется отрисованное изображение, в GRASS 6.4 задается следующими строками:

```
# Настраиваем параметры вывода
os.environ"GRASS_CAIROFILE" = filename
os.environ"GRASS_WIDTH" = width
os.environ"GRASS_HEIGHT" = height
```

Для GRASS седьмой версии можно немного упростить этот код, передавая параметры напрямую модулю `d.mon`. Пример можно посмотреть в исходной статье.

Анализ запроса пользователя и возвращение результата

Анализ запроса пользователя производится в следующей функции:

```
@view_config(route_name="wms")
def wms_view(request):

    layers = request.params.get("LAYERS", "").split(",")
    bbox = request.params.get("BBOX", "").split(",")
    width = request.params.get("WIDTH")
    height = request.params.get("HEIGHT")

    try:
        filename = _grass_wms(layers=layers,
                              bbox=bbox,
                              width=width,
                              height=height)
        f = open(filename, "r+")
        return Response(body=f.read(), content_type="image/png")
    except:
        pass
```

Данная функция принимает запрос (request), в котором могут определяться:

- список слоев в запрашиваемом изображении (параметр LAYERS);
- охват области запрашиваемого изображения (параметр BBOX);
- высота и ширина запрашиваемого изображения (параметры HEIGHT и WIDTH).

Полученные параметры анализируются и передаются в определенную выше функцию _grass_wms. Результат работы функции --- имя файла изображения --- считывается с диска и возвращается в качестве ответа на запрос.

Создание веб-приложения

Еще одно действие --- создание простейшего веб-приложения, которое использует построенный WMS сервер. Воспользуемся библиотекой OpenLayers для отображения карты, возвращаемой сервером.

Данный шаг является необязательным, но его удобно реализовать в целях отладки.

Следующая функция создает веб-страницу с картой на базе слоев векторных слоев mo и io_15 из набора PERMANENT:

```
@view_config(route_name="index")
def index_view(request):
   body = """
<html xml:lang="en"
xmlns:tal="http://xml.zope.org/namespaces/tal"
xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>GRASS GIS Web Map Service</title>
 <script src="http://openlayers.org/api/OpenLayers.js" type="text/javascript"></script>
 <script type="text/javascript">
   var map, wms_layer;
   function init(){
       var map = new OpenLayers.Map("map-div",{
           projection: "EPSG:28407",
           maxExtent: new OpenLayers.Bounds(7361000, 6115000, 7435000, 6212000),
           numZoomLevels: 6,
       });
       var wms_layer = new OpenLayers.Layer.WMS("GRASS WMS", '/wms', {
           layers: "mo@PERMANENT,io_15@PERMANENT"
       },{
           singleTile: true,
       });
       map.addLayer(wms_layer);
       map.zoomToMaxExtent();
   }
 </script>
</head>
<body onload="init()">
   < div id="map-div" style="height: 600px; width: 800px;"></div>
</body>
</html>"""
   return Response(body=body, content_type="text/html", status=200)
```

Понятно, что список доступных слоев (`layers: "mo@PERMANENT,io_15@PERMANENT"`) и параметры области (`{projection: "EPSG:28407", maxExtent: new OpenLayers.Bounds(7361000, 6115000, 7435000, 6212000), numZoomLevels: 6}`) следует отредактировать под конкретный проект.

Конфигурирование Pyramid

Для того, чтобы описанные выше функции можно было использовать, необходимо сконфигурировать Pyramid. Пример такой функции приведен ниже:

```
def main():
   config = Configurator()
   config.add_static_view('static', 'static', cache_max_age=3600)
```

```
    config.add_route("index", '/')
    config.add_route("wms", '/wms')
    config.scan()
    app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 8080, app)
    server.serve_forever()
```

Эта функция создает сервер, который будет функционировать по адресу 0.0.0.0:8080

Использование

После того, как WMS сервер будет настроен, можно его запустить командой:

```
python grass_wms.py
```

(Здесь предполагается, что программный код хранится в файле grass_wms.py).

Для проверки того, все ли было настроено верно, можно открыть браузер и в адресную строку внести параметры WMS-запроса, например,

http://0.0.0.0:8080/wms?BBOX=7361000,6115000,7435000,6212000&LAYERS=mo@PERMANENT&WIDTH=300&HEIGHT=700

В результате в окне браузера будет показана запрошенная карта, построенная на базе слоя mo@PERMANENT. Также можно загрузить сразу веб-странцу по адресу

http://0.0.0.0:8080/

на которой при помощи OpenLayers будет отображена веб-карта, поддерживающая возможности прокрутки и масштабирования.

Ограничения данной реализации

Рассмотренный пример демонстрирует принцип создания сервера WMS, потому в нем опущен ряд деталей, которые должны быть реализованы в реальной системе. Например, в рассмотренном варианте в целях упрощения изложения были введены следующие ограничения:

- Нет перепроецирования "на лету". Т.е. построенный сервер WMS может возвращать слои только в той проекции, в которой хранятся данные GRASS.
- Реализация не является потокобезопасной. Например, при одновременном запросе данных из разных областей GRASS могут проявляться неожиданные побочные эффекты (в коде производятся манипуляции с `g.region`).
- Не полная поддержка стандарта WMS. Например, описанная реализация не поддерживает запрос GetCapabilities, поэтому подключить ту или иную ГИС в качестве клиента к созданному серверу не удастся. Это ограничение легко обходится --- достаточно в функцию `wms_view` вставить обработку параметра REQUEST (и, в случае REQUEST=GetCapabilities, вернуть список доступных слоев).
- В функции _grass_wms жестко прописаны пути к GRASS и базе геоданных. В реальной системе нужно использовать .ini файл, в котором прописать необходимую информацию.

Ссылки по теме

- GRASS GIS Web Map Service with Pyramid
- Документация фреймворка Pyramid
- GRASS GIS: The world's leading Free GIS software

Обсудить в форуме Комментариев — 8

Последнее обновление: 2014-06-04 15:11

Дата создания: 04.05.2014
Автор(ы): KolesovDmitry