Встраивание кэширующего TMS-сервиса в собственное приложение

Обсудить в форуме Комментариев — 13

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

Рассмотрена процедура встраивания TileCache в собственное приложение, а также приведен пример создания кэша на базе PostgreSQL.

Содержание

Введение

Предположим, что вы разрабатываете клиент-серверную Веб-ГИС. Вам требуется отображать на клиенте некоторую растровую подложку, которая создается на базе данных хранилища (БД), к которому имеет доступ серверная компонента. Стандартным решением подобной задачи является создание на сервере Tile Map Service (TMS, спецификация) и подключение его на клиенте.

Виды TMS-сервисов

TMS-сервис может функционировать в одном из двух режимов: статическом и динамическом. Статический TMS - это предварительно созданный набор тайлов, он удобен если данные, на основе которых будет создаваться TMS-слой, одинаковы для всех установок приложения (например, данные о государственных границах) и распространяются вместе с приложением. Разумеется, это может привести к недопустимому росту размера приложения и поэтому статический TMS может также предоставлять пользователю возможность генерирования тайлов "на месте", чтобы не распространять их с приложением.

Недостаток этого варианта заключается в том, что пользователю, для того чтобы начать работать с приложением, потребуется время на создание кэша (может занимать от нескольких часов до нескольких дней). Динамической TMS решает эту проблему и генерирует тайлы по запросу.

Почему могут не подойти готовые решения

Существующее ПО для создания динамических TMS-сервисов (TileCache, MapProxy) является самостоятельным программным обеспечением и изначально не предназначено для использования в качестве встраиваемых решений. Использовать MapProxy или TileCache по их прямому назначению может быть невозможно, если:

1. Пользователь по той или иной причине не может отдельно установить и настроить тайловый сервер;
2. Доступ к тайлам должен предоставляться не всем пользователям, а только тем, которые имеют на это права согласно подсистеме разделения прав, используемой в приложении (если таковая имеется).

Решение

Поэтому решение данной задачи сводится к написанию собственного TMS-сервиса и интеграции его в приложение. Пример решения подобной задачи был рассмотрен в статье Основы работы динамических TMS-сервисов, но основным недостатком получившегося там сервиса является то, что он не поддерживает процедуру кэширования, что очень важно при разработке реального приложения.

В рамках данной статьи рассмотрим создание кэширующего TMS-сервиса на базе классов, предоставляемых TileCache. TileCache был выбран в качестве базовой системы в виду того, что он имеет довольно простую и понятную архитектуру в отличие от того же MapProxy, хотя по функционалу в целом значительно уступает последнему. В качестве рендерера будем использовать Mapnik.

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

Создание каркаса приложения

Во избежание написания большого количества служебного кода в качестве каркаса нашего приложения будем использовать Веб-фреймворк Pyramid.

Создание каталога будущего проекта

```
mkdir ~/cache
cd ~/cache
```

Создание виртуального окружения

```
virtualenv --no-site-packages env
```

Установка Pyramid

```
source env/bin/activate
pip install pyramid
```

Генерирование структуры проекта

```
pcreate -s alchemy cache
```

В файл ~/cache/cache/setup.py в массив *requires* к имеющимся пакетам добавляем имена пакетов, которые будут использоваться в нашем проекте: *psycopg2* и *TileCache*.

Установка проекта в режиме разработки

```
cd cache
python setup.py develop
```

Установка Mapnik

Идейно верное решение - это указать Mapnik как зависимость в файле setup.py для его автоматической установки, но на практике установка Mapnik в виртуальное окружение представляет собой довольно сложную задачу, поэтому для перехода к следующему шагу установите Mapnik в систему, после чего сделайте симлинк на директорию с Python-пакетами виртуального окружения. Для того, чтобы узнать, куда был установлен Mapnik, запустите системный Python и выполните следующие команды:

```
import mapnik
mapnik.__file__
'/usr/lib/python2.7/dist-packages/mapnik/__init__.py'
```

Как можно видеть, в нашем случае Mapnik был установлен в каталог /usr/lib/python2.7/dist-packages/mapnik. Находясь в активном виртуальном окружении, создаем симлинк:

```
ln -s /usr/lib/python2.7/dist-packages/mapnik $VIRTUAL_ENV/lib/python2.7/site-packages/mapnik
```

Теперь, если запустить Python в виртуальном окружении и дать команду:

```
import mapnik
```

то не должно появляться никаких сообщений об ошибках, что свидетельствует о том, что системный Mapnik виден из виртуального окружения.

Хранение кэшированных данных в PostgreSQL

Среди списка поддерживаемых кэшей в Tilecache отсутствуют те или иные СУБД. Конечно, в качестве учебного примера мы могли бы остановиться на использовании какого-нибудь стандартного кэша, например, файлового, но мы немного усложним задачу, реализовав собственный класс, отвечающий за хранение кэшированных данных в СУБД PostgreSQL. Будем считать, что PostgreSQL установлен на той же машине, что и разрабатываемое приложение.

Создание базы данных

Создадим на уровне БД отдельного пользователя, назовём его *cacheuser* (пароль *secret*):

```
sudo su postgres -c "createuser -P -e cacheuser"
Введите пароль для новой роли:
Повторите его:
Должна ли новая роль иметь полномочия суперпользователя? (y - да/n - нет) n
Новая роль должна иметь право создавать базы данных? (y - да/n - нет) n
Новая роль должна иметь право создавать другие роли? (y - да/n - нет) n
CREATE ROLE cacheuser PASSWORD 'md57ea40027c2db9dc1b1b85ad7bf0f5314' NOSUPERUSER
NOCREATEDB NOCREATEROLE INHERIT LOGIN;
```

Мы создали пользователя, который не обладает правами суперпользователя, не может создавать базы данных и другие роли. От имени пользователя *postgres* создадим базу данных *tilecache* и сделаем пользователя *cacheuser* её владельцем:

```
sudo su postgres -c "createdb -O cacheuser --encoding=UTF8 tilecache"
```

Подключение базы данных в приложение

Подключаем созданную БД в наше приложение, для этого в файле ~/cache/cache/development.ini редактируем строку *sqlalchemy.url*:

```
sqlalchemy.url = postgresql+psycopg2://cacheuser:secret@localhost/tilecache
```

Описание модели кэша

Для взаимодействия нашего приложения с базой данных будем использовать ORM SQLAlchemy. Для описания структуры таблицы базы данных, в которой будут храниться кэшированные данные, воспользуемся декларативным синтаксисом SQLAlchemy. Для этого открываем файл ~/cache/cache/cache/models.py и помещаем в него следующее содержимое:

```
# -*- coding: utf-8 -*-
from sqlalchemy import Column
from sqlalchemy import Integer, Unicode, LargeBinary

from sqlalchemy.orm import scoped_session
from sqlalchemy.orm import sessionmaker

from sqlalchemy.ext.declarative import declarative_base

from zope.sqlalchemy import ZopeTransactionExtension
```

```
DBSession = scoped_session(sessionmaker(extension=ZopeTransactionExtension()))
Base = declarative_base()

# Хранилище тайлов TMS-слоя
class TileCache(Base):
    __tablename__ = 'tilecache'
    layer = Column(Unicode(50), primary_key=True)
    z = Column(Integer, primary_key=True)
    x = Column(Integer, primary_key=True)
    y = Column(Integer, primary_key=True)
    data = Column(LargeBinary)
```

Из представленного кода видно, что кэшированные данные будут храниться в таблице *tilecache*, содержащей 5 полей: *layer* - поле, содержащее имя слой, тайл которого представлен в записи, *x*, *y*, *z* - координаты тайла, *data* - собственно сам тайл. Если в вашей базе данных установлен PostGIS, то имеет смысл добавить в эту таблицу дополнительное поле геометрии, которое будет содержать в себе значение охвата тайла. Это поле в дальнейшем очень удобно использовать, например, для очистки фрагмента кэша по какому-либо пространственному условию.

Инициализация базы данных

По описанному классу создадим таблицу в базе данных. Для этого откройте файл ~/cache/cache/cache/scripts/initializedb.py и приведите его в соответствие со следующим фрагментом:

```
import os
import sys
import transaction

from sqlalchemy import engine_from_config

from pyramid.paster import (
    get_appsettings,
    setup_logging,
    )

from ..models import (
    DBSession,
    Base
    )


def usage(argv):
    cmd = os.path.basename(argv[0])
    print('usage: %s <config_uri>\n'
          '(example: "%s development.ini")' % (cmd, cmd))
    sys.exit(1)

def main(argv=sys.argv):
    if len(argv) != 2:
        usage(argv)
    config_uri = argv[1]
    setup_logging(config_uri)
    settings = get_appsettings(config_uri)
    engine = engine_from_config(settings, 'sqlalchemy.')
    DBSession.configure(bind=engine)
    Base.metadata.create_all(engine)
```

После этого выполните команду:

```
initialize_cache_db ~/cache/cache/development.ini
```

Должно появиться следующее сообщение, свидетельствующее о том, что таблица успешно создана:

```
2013-07-17 12:05:08,945 INFO  [sqlalchemy.engine.base.Engine][MainThread] select
```

```
version()
2013-07-17 12:05:08,945 INFO  [sqlalchemy.engine.base.Engine][MainThread] {}
2013-07-17 12:05:08,961 INFO  [sqlalchemy.engine.base.Engine][MainThread] select
current_schema()
2013-07-17 12:05:08,962 INFO  [sqlalchemy.engine.base.Engine][MainThread] {}
2013-07-17 12:05:08,980 INFO  [sqlalchemy.engine.base.Engine][MainThread] select
relname from pg_class c join pg_namespace n on n.oid=c.relnamespace where
n.nspname=current_schema() and relname=%(name)s
2013-07-17 12:05:08,980 INFO  [sqlalchemy.engine.base.Engine][MainThread] {'name':
u'tilecache'}
2013-07-17 12:05:08,993 INFO  [sqlalchemy.engine.base.Engine][MainThread]
CREATE TABLE tilecache (
        layer VARCHAR(50) NOT NULL,
        z SERIAL NOT NULL,
        x INTEGER NOT NULL,
        y INTEGER NOT NULL,
        data BYTEA,
        PRIMARY KEY (layer, z, x, y)
)


2013-07-17 12:05:08,998 INFO  [sqlalchemy.engine.base.Engine][MainThread] {}
2013-07-17 12:05:09,286 INFO  [sqlalchemy.engine.base.Engine][MainThread] COMMIT
```

Создание кэширующего TMS-сервиса

Итак, все подготовительные работы закончены, переходим непосредственно к созданию сервиса. Откройте файл ~/cache/cache/cache/__init__.py и приведите его в соответствие со следующим фрагментом:

```
from pyramid.config import Configurator
from sqlalchemy import engine_from_config

from cache.models import DBSession
from cache.models import Base


def main(global_config, **settings):
    """ This function returns a Pyramid WSGI application.
    """
    engine = engine_from_config(settings, 'sqlalchemy.')
    DBSession.configure(bind=engine)
    Base.metadata.bind = engine
    config = Configurator(settings=settings)

    config.add_route('tilecache', '/tilecache/{path:.*}')
    config.add_view('cache.tilecache.view_tilecache', route_name='tilecache')

    return config.make_wsgi_app()
```

Данная функция обеспечивает следующую функциональность: любой запрос, имеющий URL вида */tilecache/** (где * - произвольная последовательность символов), будет обработан функцией *cache.tilecache.view_tilecache* .

Переходим к написанию этой функции. Создайте файл ~/cache/cache/cache/tilecache.py:

```
# -*- coding: utf-8 -*-
import datetime
import os

from TileCache.Cache import Cache
from TileCache.Service import Service
from TileCache.Layers.Mapnik import Mapnik

from pyramid.response import Response
```

```
from cache.models import DBSession
from cache.models import TileCache as TileCacheModel

_tilecache_service = None

# Весь дальнейший код будет помещен сюда
```

Описание класса кэша PostgreSQL

В TileCache в роли кэша может выступать любой объект класса, имеющий 2 обязательных метода: *get* и *set*. Первый отвечает за извлечение данных из кэша, второй - за помещение их туда (в случае если запрошенный клиентом тайл отстутствует в кэше). Здесь же, в файле tilecache.py, описываем соответствующий класс, унаследовавшись от базового класса TileCache.Cache.Cache:

```
class PostgresCache(Cache):
    def __init__(self, **kwargs):
        Cache.__init__(self, **kwargs)

    def get(self, tile):

        dbsession = DBSession()
        obj = dbsession.query(TileCacheModel).\
            filter_by(
                layer=tile.layer.name,
                z=tile.z,
                x=tile.x,
                y=tile.y
            ).first()
        if obj:
            return obj.data
        else:
            return None

    def set(self, tile, data):

        dbsession = DBSession()

        obj = TileCacheModel(
            layer=tile.layer.name,
            z=tile.z,
            x=tile.x,
            y=tile.y,
            data=data
        )
        dbsession.add(obj)
        dbsession.flush()

        return data
```

Подготовка конфигурационного XML-файла Mapnik

При запросе клиентом тайла в случае если его нет в кэше, его нужно создать (отрендерить). Как уже было сказано выше для этой цели мы будем использовать Mapnik.

В качестве источника данных воспользуемся результатами проекта, в рамках которого был создан открытый точечный слой геоданных по детским учреждениям (детским домам). Этот слой также доступен в БД PostGIS.

Создадим файл ~/cache/cache/cache/static/mapnik.xml следующего содержания:

```
<?xml version="1.0" encoding="utf-8"?>
<Map srs="+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0.0
+k=1.0 +units=m +nadgrids=@null +wktext +no_defs +over" background-color="#00000000"
buffer-size="128">
```

```
<Style name="geodetdom" filter-mode="first" >
  <Rule>
    <PointSymbolizer file="icon.png" allow-overlap="true"/>
  </Rule>
</Style>

<Layer name="geodetdom" srs="+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs">
    <StyleName>geodetdom</StyleName>
    <Datasource>
       <Parameter name="type">postgis</Parameter>
       <Parameter name="host">gis-lab.info</Parameter>
       <Parameter name="dbname">geodetdom</Parameter>
       <Parameter name="table">data</Parameter>
       <Parameter name="geometry_field">geometry</Parameter>
       <Parameter name="user">guest</Parameter>
       <Parameter name="password">guest</Parameter>
    </Datasource>
  </Layer>

</Map>
```

В эту же директорию поместим файл icon.png, которым будут отрисовываться наши данные (любая растровая иконка):

Функция обработки входящих запросов

Представим код функции и в комментариях опишем, что в ней происходит:

```
def view_tilecache(request):

    global _tilecache_service

    # Настройки слоя Tilecache
    layer_args = dict(
        srs='EPSG:3857',
        spherical_mercator='yes',
        tms_type='google',
        debug='no'
    )

    # Создаем сервис (объект класса TileCache.Service.Service),
    # указывая объект кэша, конфигурационный файл Mapnik-а и словарь
    # слоев, в котором в качестве ключа используется имя слоя, которое
    # будет передаваться в URL (согласно спецификации TMS)
    if not _tilecache_service:

        # Определяем путь до конфигурационного файла Mapnik
        mapfile = os.path.join(os.path.split(__file__)[0], 'static', 'mapnik.xml')

        # Здесь мы могли создать объект какого-нибудь стандартного
        # кэша TileCache, например TileCache.Caches.Disk
        cache = PostgresCache()

         # Сохраняем объект сервиса в глобальной переменной
        _tilecache_service = Service(
            cache,
            # Словарь доступных слоев
            dict(
                geodetdom=Mapnik('domiki', mapfile, cache=cache, **layer_args)
            )
```

```
    )

    # Так как шаблон используемого URL имеет вид '/tilecache/{path:.*}',
    # то в переменную path извлекается та часть URL, которая следует
    # за '/tilecache/'.
    path = request.matchdict['path']

    # Вызываем метод dispatchRequest сервиса _tilecache_service,
    # который в по виду URL определяет какой запрос пришел
    # от пользователя (TileCache поддерживает не только TMS, но и
    # WMS-C и WorldWind, а также может возвращать документы с метаданными) и
    # формирует ответ соответствующего типа
    format, body = _tilecache_service.dispatchRequest(
        request.params,
        path,
        request.method
    )

    # Выставляем значение заголовка Expires для того, чтобы
    # браузер смог кэшировать тайлы на стороне клиента
    expires = datetime.datetime.utcnow() + datetime.timedelta(seconds=3600)

    # Возвращаем результат клиенту
    return Response(body, content_type=format, expires=expires)
```

Запуск сервиса

Запускаем наш сервис:

```
pserve ~/cache/cache/development.ini
```

По умолчанию сервис будет запущен на порту с номером 6543.

Результаты

Открываем браузер и в адресной строке вводим URL следующего вида (подразумевается, что клиент и сервер физически находятся на одной машине, если у вас не так, то вместо localhost введите имя хоста, на котором запущен сервис):

```
http://localhost:6543/tilecache/1.0.0/geodetdom/0/0/0.png
```

В ответ вы должны получить следующее изображение:


Пример запрошенного тайла

Кроме того, созданный тайл будет помещен в базу данных:

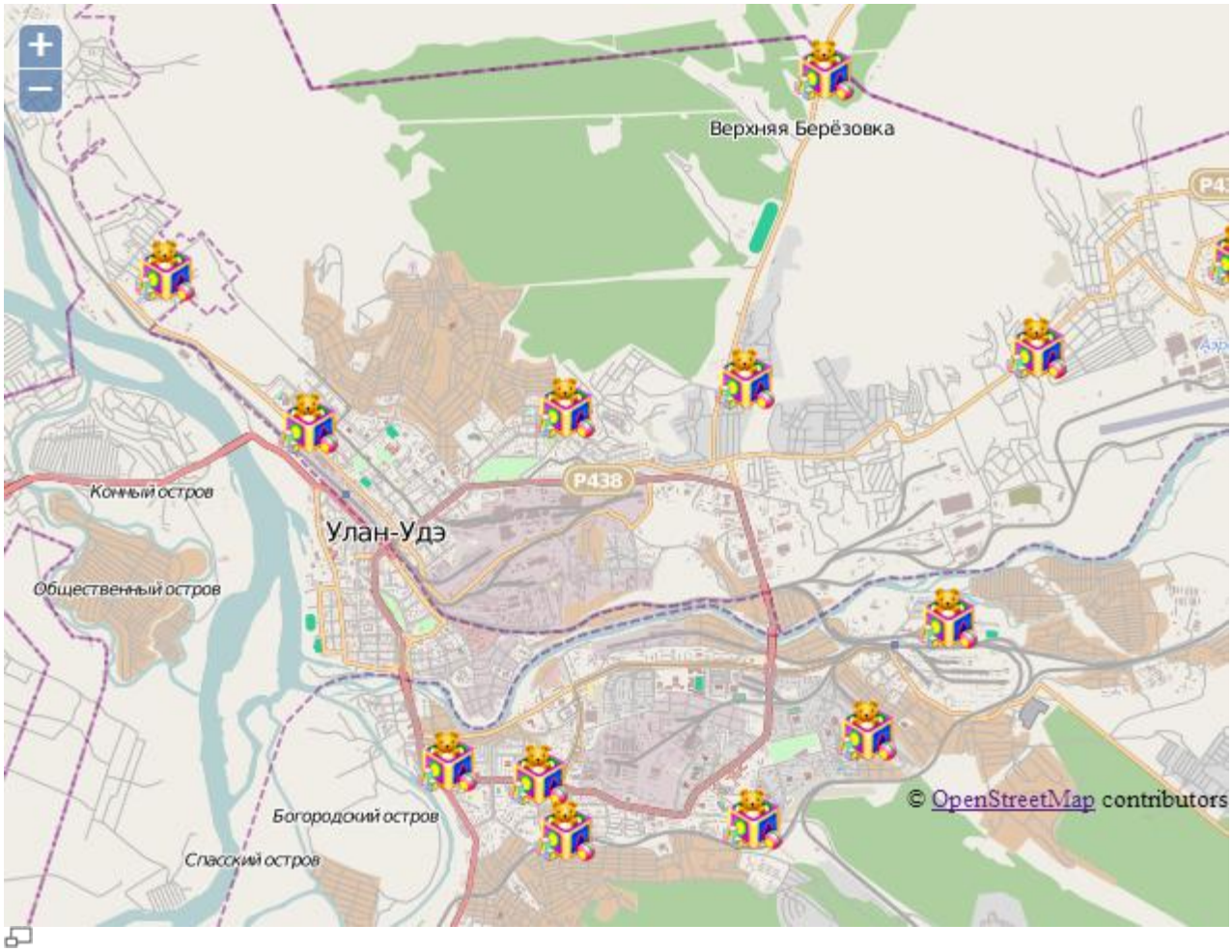
	layer [PK] characte	z [PK] serial	x [PK] integer	y [PK] integer	data bytea
1	domiki	0	0	0	<binary data>
*					

Пример закэшированного тайла

Подключим созданный нами TMS-слой в OpenLayers:

```
<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(){

    options = {
        div: "map",
        zoom: 1,
        center: 0,0,
        layers:
            new OpenLayers.Layer.OSM()

    };
    map = new OpenLayers.Map(options);

    var domiki = new OpenLayers.Layer.XYZ('domiki',
        'http://localhost:6543/tilecache/1.0.0/geodetdom/${z}/${x}/${y}.png',
        {
            sphericalMercator: true,
            isBaseLayer: false
        });
    map.addLayer(domiki);
}
</script>
</head>
<body onload="init()">
    <div id="map" style="width:640px; height:480px;"></div>
</body>
</html>
```

Выглядеть это будет следующим образом:



Подключение созданного TMS-слоя в OpenLayers

И кэш в СУБД значительно пополнится:

	layer [PK] characte	z [PK] serial	x [PK] integer	y [PK] integer	data bytea
131	domiki	11	1634	1370	<binary data>
132	domiki	11	1635	1368	<binary data>
133	domiki	11	1635	1369	<binary data>
134	domiki	11	1635	1370	<binary data>
135	domiki	11	1636	1368	<binary data>
136	domiki	11	1636	1369	<binary data>
137	domiki	11	1636	1370	<binary data>
138	domiki	11	1637	1368	<binary data>
139	domiki	11	1637	1369	<binary data>
140	domiki	11	1637	1370	<binary data>
141	domiki	12	3271	2738	<binary data>
142	domiki	12	3271	2739	<binary data>
143	domiki	12	3271	2740	<binary data>
144	domiki	12	3271	2741	<binary data>
145	domiki	12	3272	2738	<binary data>
146	domiki	12	3272	2739	<binary data>
147	domiki	12	3272	2740	<binary data>
148	domiki	12	3272	2741	<binary data>
149	domiki	12	3273	2738	<binary data>
150	domiki	12	3273	2739	<binary data>
151	domiki	12	3273	2740	<binary data>
152	domiki	12	3273	2741	<binary data>
153	domiki	12	3274	2738	<binary data>
154	domiki	12	3274	2739	<binary data>
155	domiki	12	3274	2740	<binary data>
156	domiki	12	3274	2741	<binary data>
*					

Заполненный кэш

Заключение

Представленный вариант интеграции возможностей TileCache был использован компанией NextGIS при разработке картографической подсистемы портала системы государственного информационного обеспечения в сфере сельского хозяйства (СГИО СХ).

Обсудить в форуме Комментариев — 13

Последнее обновление: 2014-05-15 01:47

Дата создания: 17.07.2013
Автор(ы): Денис Рыков