django-geoip-redis Documentation

Release 1.2.1

gotlium

December 24, 2014

Contents

App to figure out where your visitors are from by their IP address.

Detects country, region, city, isp and provider querying in the database with geodata. Optional *high-level API* provides geo data on request object.

Note: Current version support only ipgeobase.ru. IPGeoBase provide information about Russia and Ukraine. There are plans to add other backends in future releases.

CHAPTER 1

Contents

1.1 Installation

1.1.1 Compatibility

- Python: 2.6, 2.7, 3.3
- Django: 1.3.x, 1.4.x, 1.5.x, 1.6

Recommended way to install is via pip:

```
pip install django-geoip-redis
```

1.1.2 Basic

- Add `geoip` to `INSTALLED_APPS` in settings.py:

  ```
  INSTALLED_APPS = (
      ...
      'geoip',
      ...
  )
  ```

- Create application tables on database:

  ```
  python manage.py syncdb
  ```

  If you're using South:

  ```
  python manage.py migrate
  ```

- Get latest data to perform geoip detection by *running management command*:

  ```
  python manage.py update_geo_db
  ```

1.1.3 Advanced

In order to make *user's location detection automatic* several other steps are required:

- Add `GeoMiddleware` to `MIDDLEWARE_CLASSES`:

```
MIDDLEWARE_CLASSES = (
    ...
    'geoip.middleware.GeoMiddleware',
)
```

1.2 Usage

The app provides both high and low-level APIs to work with geolocation. Low-level API works super-simple: it guesses geographic location by an IP adress. High-level API is more complex and deeply integrated in Django: it automatically detects user location in every request and makes it available as `request.geo`.

1.2.1 Low-level API usage

Low-level API allows you to guess user's location by his IP address. This function returns dictionary associated with IP's city, area, country, isp and provider.

Here is a basic example:

```
from geoip.geo import record_by_ip_as_dict

ip = '91.195.136.52'

geoip_record = record_by_ip_as_dict(ip)

if geoip_record is not None:
    print geoip_record.get('country')
    print geoip_record.get('area')
    print geoip_record.get('city')
    print geoip_record.get('isp')
    print geoip_record.get('provider')
else:
    print 'Unknown location'
```

1.2.2 High-level API usage

The app provides a convenient way to detect user location automatically. If you've followed *advanced installation instructions*, user's location should be accessible via `request` object:

```
def my_view(request):
    """ Passing location into template """
    ...
    context['geo'] = request.geo
    ...
```

1.3 Under the hood

1.3.1 Data storage

All geoip data, including geography and geoip mapping is stored in the database. To avoid unnecessary database hits user location is stored in a cookie.

Geography

Django-geoip-redis supports only ipgeobase geography, which consist of following entities: Country, Region, City. Database maintains normalized relationships between all entities, i.e. Country has many Regions, Region has many Cities, ISP has many Country, Provider has many ISP.

IP ranges

1.3.2 Backends

There is currently no infrastructure to use alternative geoip backends, but it's planned for future releases. Pull requests are also welcome.

Ipgeobase backend

ipgeobase.ru is a database of Russian and Ukranian IP networks mapped to geographical locations.

It's maintained by RuCenter and updated daily.

As of 11 Nov 2013 it contains info on 992 cities and 152333 Ip Ranges (some networks doesn't belong to CIS).

Here a is demo of ip detection: http://ipgeobase.ru/

1.4 Updating GeoIP database

Note: Currentrly `django-geoip-redis` supports only ipgeobase.ru backend.

To update your database::

```
python manage.py update_geo_db
```

Warning: This is irreversible operation, do not use on production!

Note: If you're having `2006, 'MySQL server has gone away'` error during database update, setting `max_allowed_packet` to a higher value might help. E.g. `max_allowed_packet=16M`

1.5 Settings

`django-geoip-redis` has some configuration:

```
# Values: 'db' or 'redis'
GEO_BACKEND = 'redis'
GEO_USE_CELERY = False

GEO_REDIS_HOST = 'localhost'
GEO_REDIS_PORT = 6379
GEO_REDIS_PASSWORD = None
GEO_REDIS_DB = 1
```

```
# Values: 'name' or 'pk'
GEO_REDIS_TYPE = 'name'
```

1.6 Contributing

If you wish to contribute, please add corresponding tests. Running tests:

```
make test
```

Checking coverage (requires `coverage` package):

```
make coverage
```

Run tests for all python-django combinations

```
tox
```

1.7 Authors

Django-geoip-redis is written and maintained by GoTLiuM.

1.8 Performance

- **django-geoip-redis:**
  - MySQL(SSD): 728 rps
  - SQLite(SSD): 46 rps
  - Redis: **3548 rps**
- **django-geoip(no isp, no provider):**
  - MySQL(SSD): 855 rps
  - SQLite(SSD): 47 rps
- **django.contrib.gis.geoip.GeoIP(no isp, no provider, but C API):**
  - standard: 4666 rps
  - memory: 73 rps
  - check: 4510 rps
  - index: 76 rps
  - mmap: 4425 rps

Tested on Ubuntu 12.04(x86_64), Django(1.6), uWSGI(1.0.3), Nginx(1.1.19) with Apache Benchmark:

```
$ ab -c 100 -n 1000 http://localhost/ip/91.195.136.52/
```

On tests used default configuration for Redis & MySQL without any modifications.

1.9 Installation for development

```
$ sudo apt-get install redis-server virtualenvwrapper
$ mkvirtualenv django-geoip-redis
$ git clone https://github.com/gotlium/django-geoip-redis.git
$ cd django-geoip-redis
$ pip install -r requirements/package.txt
$ python setup.py develop
$ cd demo
$ pip install -r requirements.txt
$ python manage.py syncdb --noinput
$ python manage.py migrate
$ python manage.py loaddata ../fixtures/db.json
$ python manage.py shell

>>> from geoip.geo import record_by_ip_as_dict
>>> print (record_by_ip_as_dict('91.195.136.52'))
```

If you want use native db for local development, you can add `GEO_BACKEND = 'db'` into local_settings.py

CHAPTER 2

Contributing

You can grab latest code on `development` branch at Github.

Feel free to submit issues, pull requests are also welcome.

Good contributions follow *simple guidelines*