Урок 2. Профилирование flask-приложения

Сперва давайте познакомимся с профилировщиком, встроенным в PyCharm.

```
class Cocktail:

   def do(self):
       pass


class Barman:

   def __init__(self):
       self.no_cocktails()

   def no_cocktails(self):
       self.cocktails = []

   def add_cocktail_order(self, cocktail):
       self.cocktails.append(cocktail)

   def do_cocktails(self):
       [cocktail.do() for cocktail in self.cocktails]
       self.no_cocktails()


def main():
   barman = Barman()
   for c in range(10):
       barman.add_cocktail_order(Cocktail())
   barman.do_cocktails()


if __name__ == '__main__':
   main()
```

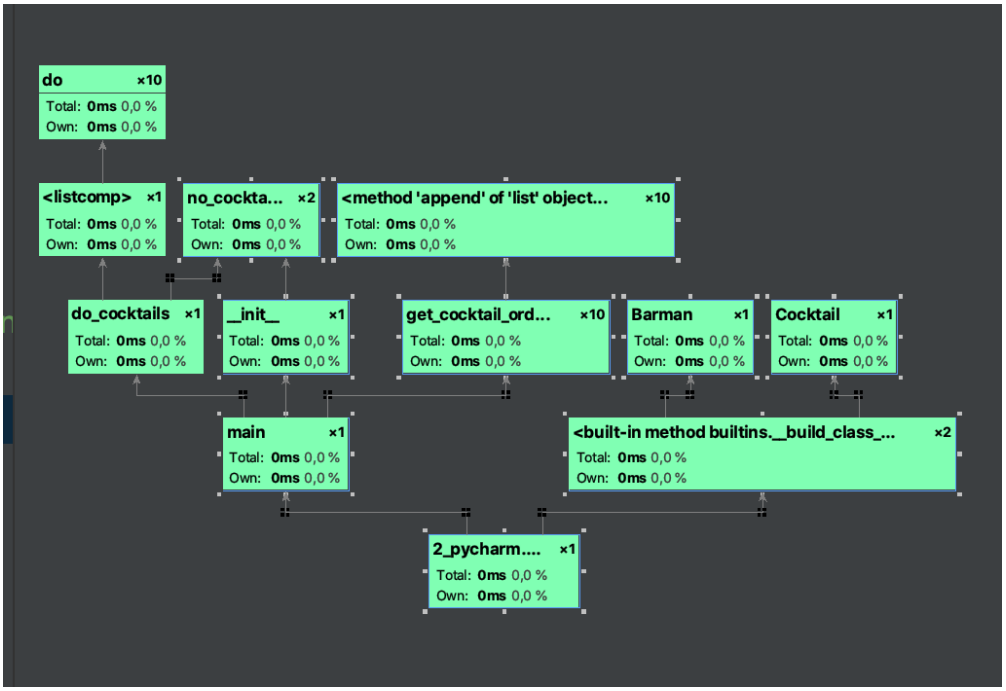
По умолчанию Pycharm использует профилировщик cProfile — это встроенный в Python событийный профайлер, который написан на языке СИ.

Вкладка «Статистика»:

Statistics | Call Graph

Name	Call Count
<method 'append' of 'list' objects>	10
<built-in method builtins.__build_class__>	2
__init__	1
no_cocktails	2
<listcomp>	1
get_cocktail_orders	10
do	10
do_cocktails	1
Barman	1
main	1
2_pycharm.py	1
Cocktail	1

Вкладка «График»:



В случае с flask-приложением данный профайлер достаточно тяжело читать, так как есть множество системных вызовов функций. Для профилирования эндпоинтов давайте рассмотрим пакет flask_profiler.

flask_profiler измеряет эндпоинты приложения, определяет количество вызовов, скорость выполнения роутов, контекст запроса и входные параметры. Последнее

очень удобно для понимания, при каких входных параметрах роут работает медленно. Все это отображается отчётом в веб-интерфейсе.

```
from flask import Flask
import flask.json
import time
import flask_profiler
import decimal

from sqlalchemy import create_engine
engine = create_engine('sqlite:///flask_profiler.db')

app = Flask(__name__)


@app.route("/one")
def one():
    return "one"


@app.route("/two")
def two():
    time.sleep(10)
    return "two"


@app.route("/three")
def three():
    l = []
    for i in range(1000000):
        l.append(i)
    return "three"


@app.route("/four")
def four():
    return "four"

app.config["flask_profiler"] = {
    "enabled": True,
    # "storage": {
    #     "engine": "sqlite"
    # },
    "storage": {
        "engine": "sqlalchemy",
        "db_url": "sqlite:///flask_profiler.db"
    },
    "basicAuth":{
        "enabled": True,
```

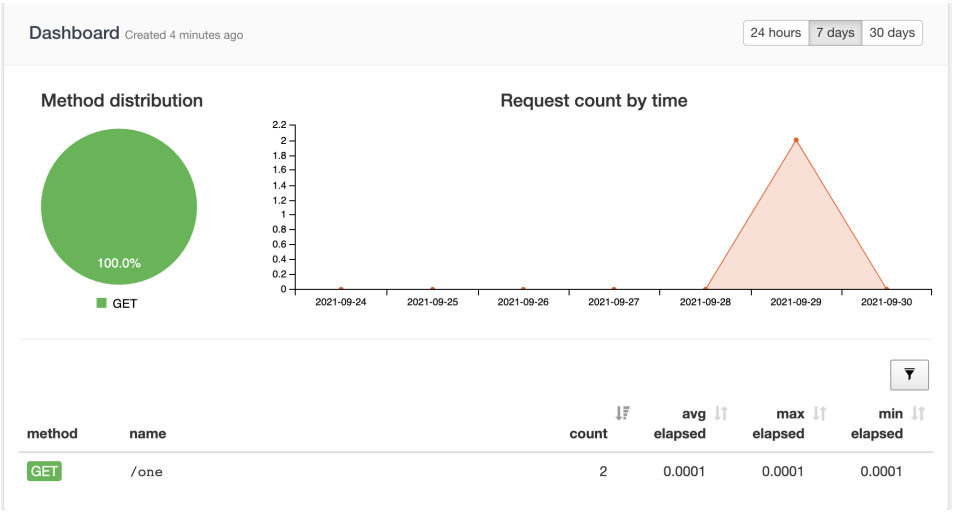
```
        "username": "admin",
        "password": "admin"
    }
    # "sampling_function": lambda: True if 1 is 1 else False,
    # "ignore": [
       #     "^/static/.*"
       # ]
}
flask_profiler.init_app(app)


class NewEncoder(flask.json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            return str(o)
        return super(NewEncoder, self).default(o)

app.json_encoder = NewEncoder
# @app.route('/five', methods=['GET'])
# @flask_profiler.profile()
# def five():
#     return "five"


if __name__ == '__main__':
    app.run()
```

Дашборд:



Информация запросов:

init_app необходимо вызывать после описания всех роутов, если есть необходимость измерять роут, который описан после init_app (есть декоратор).

В целом данный пакет очень удобен, но есть ряд элементов, которые нужно либо поправить для корректной работы, либо дописать для себя. Кстати, это будет интересная практика: вы можете посмотреть внутрь пакета flask_profiler, изучить его и, возможно, поправить, сделав pull request с исправлениями в официальный репозиторий.

Ещё одним интересным инструментом является встроенный во Flask профилировщик пакета werkzeug, Это пакет, предоставляющий промежуточный интерфейс между flask-приложением и профайлером cprofile для анализа эндпоинтов. Чтобы запустить профайлер, необходимо написать две строки кода.

```python
from flask import Flask
import time
from werkzeug.middleware.profiler import ProfilerMiddleware

app = Flask(__name__)
app.wsgi_app = ProfilerMiddleware(app.wsgi_app, profile_dir='.')


@app.route("/one")
def one():
    return "one"


@app.route("/two")
def two():
    time.sleep(10)
    return "two"


@app.route("/three")
def three():
    l = []
    for i in range(1000000):
        l.append(i)
    return "three"


@app.route("/four")
def four():
    return "four"


if __name__ == '__main__':
    app.run()
```

```
--------------------------------------------------------------------------------
PATH: '/one'
         453 function calls in 0.001 seconds

   Ordered by: internal time, call count

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        4    0.000    0.000    0.000    0.000 /Users/a06831813/Desktop/projects/skillbox/env/lib/python3.7/site-packages/flask/wrappers.py:95(blueprints)
        1    0.000    0.000    0.000    0.000 /Users/a06831813/Desktop/projects/skillbox/env/lib/python3.7/site-packages/werkzeug/wrappers/request.py:107(__init__)
        1    0.000    0.000    0.000    0.000 /Users/a06831813/Desktop/projects/skillbox/env/lib/python3.7/site-packages/werkzeug/routing.py:1603(bind_to_environ)
        1    0.000    0.000    0.000    0.000 /Users/a06831813/Desktop/projects/skillbox/env/lib/python3.7/site-packages/flask/ctx.py:372(push)
        2    0.000    0.000    0.000    0.000 /Users/a06831813/Desktop/projects/skillbox/env/lib/python3.7/site-packages/flask/app.py:1748(create_url_adapter)
        1    0.000    0.000    0.000    0.000 /Users/a06831813/Desktop/projects/skillbox/env/lib/python3.7/site-packages/werkzeug/sansio/request.py:120(__init__)
        8    0.000    0.000    0.000    0.000 /Users/a06831813/Desktop/projects/skillbox/env/lib/python3.7/site-packages/werkzeug/local.py:241(top)
```