Microsoft Partner
Silver Learning

Информационный видеосервис для разработчиков программного обеспечения

Python Practice

Обеспечение периодического обновления курсов. Выбор решения.

ITVDN
IT VIDEO DEVELOPERS NETWORK

Автор курса

Кременtарь Ксения

Ведущий Python разработчик

Системный архитектор

в компании K-Solutions

После урока обязательно

Повторите этот урок в видео формате на ITVDN.com

Проверьте как Вы усвоили данный материал на TestProvider.com

Обеспечение периодического обновления курсов. Выбор решения.

На этом уроке

1. Узнаем о вариантах организации периодических задач.
2. Узнаем о том, что такое очереди задач и зачем они нужны.
3. Познакомимся с различными вариантами очередей в Python и их плюсами и минусами.
4. Научимся использовать библиотеку APScheduler для организации периодических задач.

Что же именно нужно сделать?

Сегодня нам предстоит добавить новый функционал в проект Golden Eye, но в этот раз нам не придется вносить изменения в web приложение Golden Eye. Необходимо придумать, каким образом можно организовать периодическое обновление курсов валют из удаленных api.

Процесс такого периодического обновления выходит за рамки привычного нам цикла запрос-ответ, так как запрос инициирует удаленный клиент. А для автоматического обновления нет такого "клиента". И процесс обновления должен быть реализован вне цикла обработки HTTP запроса.

Достаточно часто при построении проектов разработчик сталкивается с такой задачей, как организация какой-либо логики, которая должна быть выполнена без запроса на web сервис. А в нашем случае — нужно добавить логику с периодическим вызовом метода api.update_rate() для каждого курса в базе данных. То есть вполне очевидно, что нужно написать небольшой python модуль, который будет выполнять нужные действия — и, главное, организовать его регулярный запуск.

Варианты организации

Существует несколько вариантов организации таких фоновых процессов, основные это:

- На уровне ОС.
- Очереди задач.

Cron в Linux, Планировщик задач в Windows и подобные сервисы на уровне ОС.
Это встроенные в ОС утилиты, которые собственно и созданы для запуска периодических задач по расписанию. Задачей в данном случае может являться наш python скрипт, запускаемый python интерпретатором.
Минусы такого подхода — это то, что он подходит только в случае разворачивания приложения на выделенном сервере с дальнейшим его администрированием. Нам этот вариант сейчас не подходит. Но если вы разворачиваете приложения на выделенном сервере, вы всегда можете иметь в виду такой подход — какие именно возможность есть на уровне ОС вашего сервера.

Очереди задач

Альтернативные варианты — это так называемые очереди задач (Task queues). Это специальные сервисы, которые могут быть использованы в различных ОС как отдельные компоненты системы. В их компетенцию входит уже гораздо больше, нежели просто периодически запускаемые задачи, а именно:

- Выполнять отложенные задания
- Распределенное выполнение (может быть запущен на N серверах)
- Проверять выполнилось ли задание и т.п.

Очереди задач в Python

Де-факто стандартом в мире Python в этой сфере есть библиотека Celery. В принципе, Celery удовлетворяет всем возможным требования в сфере отложенных заданий. Но это достаточно сложная система, требующая тщательного изучения и понимания принципов ее работы.

Альтернативные python библиотеки родились именно из необходимости более простых решений для более простых задач, где не нужна мощность всего инструмента Celery.

Например, RQ (Redis Queue) — это простая библиотека Python для организации очередей заданий и их обработки воркерами в фоновом режиме. RQ имеет низкий барьер для входа. Но минусами обоих указанных вариантов является необходимость в дополнительных сервисах.
Для RQ необходим Redis(NoSQL БД) в качестве бекграунда, для хранения и обработки задач.
Для Celery необходим сервис для отправки и получения сообщений, так называемый брокер сообщений. В качестве брокера может быть использован, например, тоже Redis или RabbitMQ или другие сервисы. Для решения такой простейшей задачи как наша, поднимать дополнительные сервисы только для того, чтоб раз в 10 минут отправлять запросы на получени курсов валют, кажется нецелесообразным.

Простые альтернативы

Есть более простые библиотеки, которые позволяют запускать периодические задачи и не требуют дополнительных сервисов.
APScheduler — это простой в использовании, легкий и удобный планировщик задач. Он не требует дополнительного бекенда, не имеет никаких зависимостей и не требует брокера сообщений или иного дополнительного сервиса. Для реализации нашей задачи его будет вполне достаточно.
Еще одним плюсом является то, что он поддерживается на сервисе для хостинга Heroku, где мы планируем размещать наш проект. Что еще раз доказывает его актуальность в решении таких простых задач. Также он, конечно же, может быть использован и при размещении проекта на выделенном сервере.
При необходимости, бекенд для хранения задач может быть подключен в APScheduler. Возможные варианты — реляционные БД, MongoDB и Redis (нереляционные БД). Но нас в первую очередь интересует простейший вариант, без дополнительно бекенда.

Работа с APScheduler

Для организации периодически выполняемой задачи необходимо создать новый модуль, в котором создать экземпляр шедулера, описать функцию, в которой будет описана логика задачи. И указать правила запуска функции. А затем запустить шедулер, вызвав метод start() у объекта. Затем необходимо запустить модуль на исполнение.

Добавление задачи в APScheduler

Есть два способа добавить задания в шедулер:

- вызывая add_job() метод
- декорируя функцию с помощью schedule_job() декоратора

Для указания периодичности запуска необходимо выбрать тип триггера:
date — для запуска в определенную дату;
interval — для запуска с определенным интервалом с момента первого выполнения;
cron — с указанием так называемого cron-выражения, наболее универсальный вариант.

Добавим первую задачу

Создадим простейший модуль tasks.py с задачей, которая будет периодически выводить на печать текущее время и запустим шедулер.

```
from datetime import datetime

from apscheduler.schedulers.blocking import BlockingScheduler

sched = BlockingScheduler()


@sched.scheduled_job('interval', minutes=1)
def get_now():
    print(f"now: {datetime.now()}")


sched.start()
```

```
krementar@MacBook-Pro-Ksenia:~/projects/itvdn_git/lesson 13$ python3 tasks.py
now: 2019-02-22 20:02:43.658033
now: 2019-02-22 20:03:43.658472
now: 2019-02-22 20:04:43.658017
now: 2019-02-22 20:05:43.673294
now: 2019-02-22 20:06:43.657275
```

Задача по обновлению курсов

А теперь модифицируем скрипт, добавив логику с получением всех курсов из БД и обновлением каждого с помощью пакета api. Добавим запуск обновления курсов каждые 10 минут с помощью APScheduler и запустим скрипт.

```
from datetime import datetime

from apscheduler.schedulers.blocking import BlockingScheduler

from models import XRate
import api

sched = BlockingScheduler()


@sched.scheduled_job('interval', minutes=10)
def update_rates():
    print(f"Job started at {datetime.now()}")
    xrates = XRate.select()
    for rate in xrates:
        try:
            api.update_rate(rate.from_currency, rate.to_currency)
        except Exception as ex:
            print(ex)
    print(f"Job finished at {datetime.now()}")


sched.start()
```

Добавление логирования во Flask приложение

Давайте добавим логирование во Flask приложение, для этого воспользуемся встроенным во Flask логгером на базе стандартной библиотеки logging. Объект логгера доступен через атрибут app.logger.
Для конфигурирования логгера приложения удобно воспользоваться специальной функцией logging.config.dictConfig, которая в качестве аргумента принимает словарь определенного вида.

И после этого можно добавлять логи в наше приложение, например, в базовый контроллер — логировать начало обработки запроса, конец обработки запроса, какие-то промежуточные данные и тп.

Пример словаря для конфигурирования логгера

```
LOGGING = {
    'version': 1,
    'formatters': {
        'default': {
            'format': "[%(asctime)s] [%(levelname)s] - %(name)s: %(message)s",
        },
    },
    'handlers': {
        'file': {
            'class': 'logging.FileHandler',
            'formatter': 'default',
            'filename': 'new.log',
        },
    },
    'loggers': {
        'GoldenEye': {
            'handlers': ['file', ],
            'level': logging.DEBUG
        },
        'Api': {
            'handlers': ['file', ],
            'level': logging.DEBUG
        },
    },
}
```

```
import logging
from logging.config import dictConfig

from flask.logging import default_handler
from flask import Flask

from config import LOGGING

dictConfig(LOGGING)
app = Flask(__name__)

app.logger = logging.getLogger('GoldenEye')
app.logger.removeHandler(default_handler)
```

Добавление отображения error логов

Для этого добавим динамическую часть url во view-функцию view_logs, <log_type>, передачу параметра log_type в контроллер ViewLogs при вызове метода call, анализ этого параметра в коде контроллера и выбор из БД записей из нужной таблицы — ApiLog или ErrorLog.

```
@app.route("/logs/<log_type>")
@check_ip
def view_logs(log_type):
    return controllers.ViewLogs().call(log_type)
```

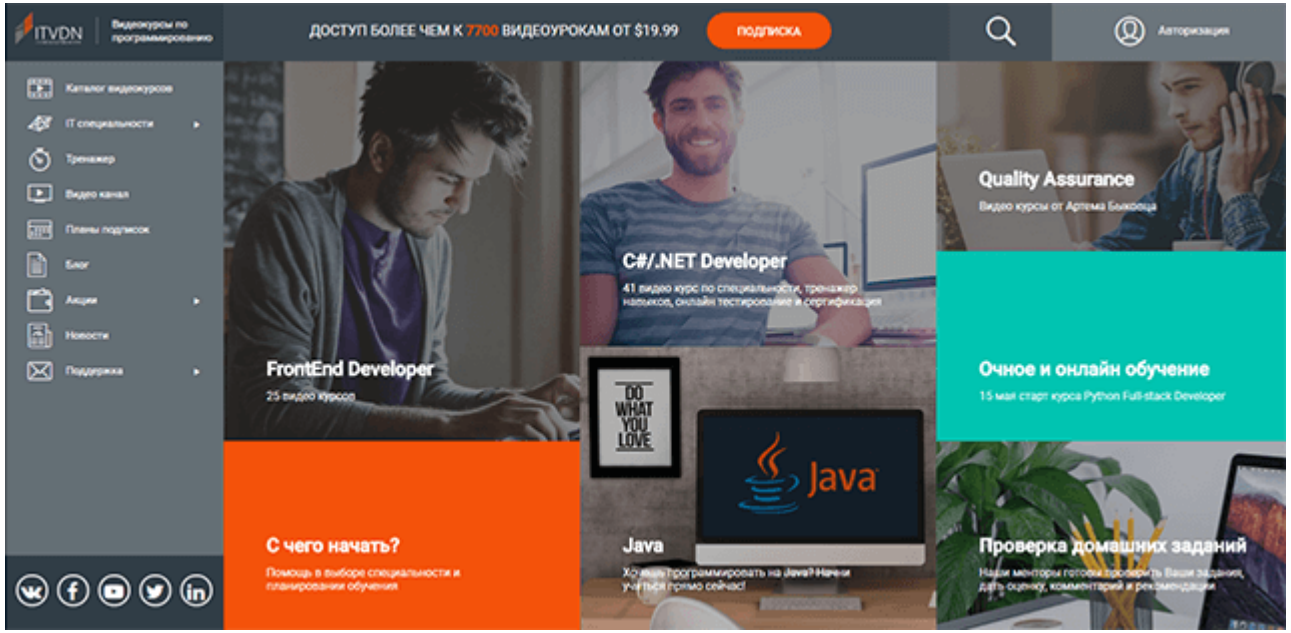
```
class ViewLogs(BaseController):
    def _call(self, log_type):
        page = int(self.request.args.get("page", 1))
        logs_map = {"api": ApiLog, "error": ErrorLog}

        if log_type not in logs_map:
            raise ValueError("Unknown log_type: %s" % log_type)

        log_model = logs_map[log_type]
        logs = log_model.select().paginate(page, 10).order_by(log_model.id.desc())
        return render_template("logs.html", logs=logs)
```

Что дальше?

На следующем уроке мы рассмотрим различные варианты деплоймента python приложений, познакомимся с основными этапами и особенностями размещения проектов на Heroku и задеплоим проект Golden-Eye. Также подведем итоги курса Python Practice.

Смотрите наши уроки в видео формате

ITVDN.com



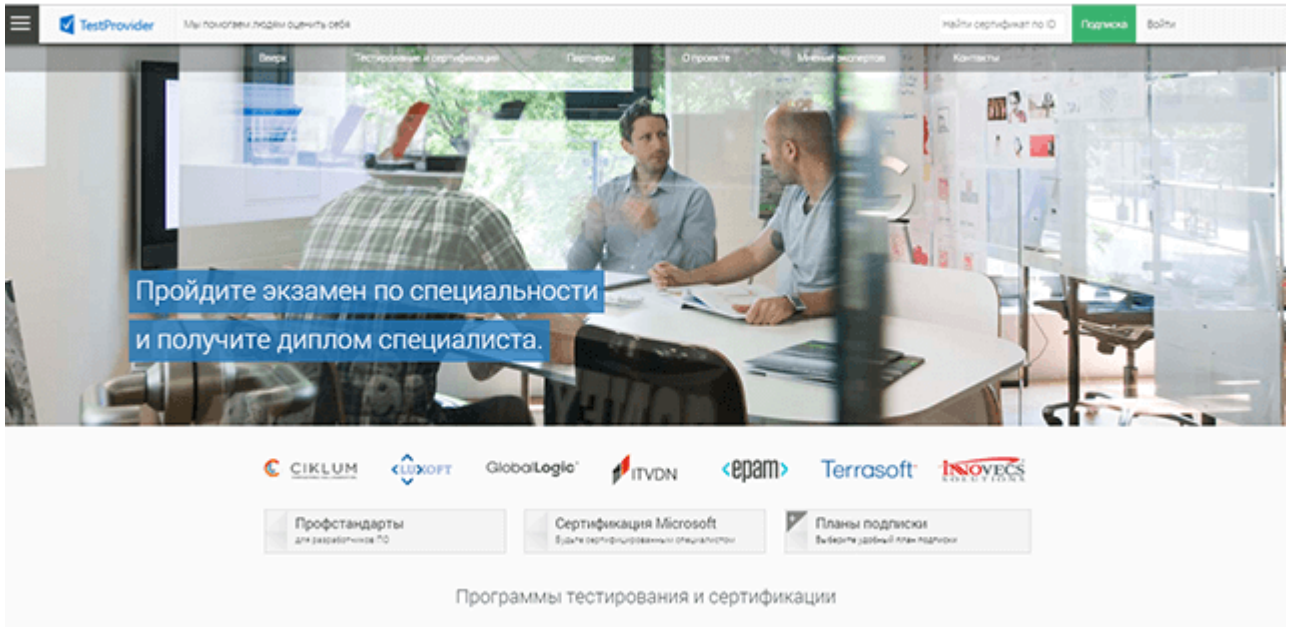
Посмотрите этот урок в видео формате на образовательном портале ITVDN.com для закрепления пройденного материала.

Курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics и другими высококвалифицированными разработчиками.

ITVDN
IT VIDEO DEVELOPERS NETWORK

Проверка знаний

TestProvider.com



TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

TestProvider

Q&A

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association