Обеспечение периодического обновления курсов. Выбор решения.

№ урока:	13	**Курс:**	Python Practice

Средства обучения:	Интерпретатор Python, virtualenv, текстовый редактор

Обзор, цель и назначение урока

Цель урока: Добавить автоматическое обновление курсов по api в проекте Golden-Eye. Рассмотреть варианты организации периодического обновления курсов. А также добавить логирование во Flask приложение.

Изучив материал данного занятия, учащийся:

- Узнает о том, что такое очереди задач и зачем они нужны
- Познакомится с различными вариантами очередей и их плюсами и минусами
- Научится использовать библиотеку APScheduler для организации периодических задач

Содержание урока

1. Краткое резюме, что мы уже имеем к данному момент и обзор дальнейших действий
2. Постановка задачи на сегодня: организовать и реализовать периодическое обновление курсов из удаленных источников. Отличия "фоновых" задач от обычной от логики, выполняемой в рамках цикла HTTP запрос-ответ
3. Обзор вариантов организации выполнения "фоновых" задач: cron, Celery, Redis Queue, APScheduler. Плюсы и минусы подходов.
4. Выбор APScheduler в качестве решения. Создание простейшей задачи и запуск ее с помощью APScheduler.
5. Создание модуля tasks.py для реализации периодического обновления курсов с помощью APScheduler, запуск процесса. Анализ логов для контроля запуска каждые 10 минут.
6. Добавление логирования во Flask приложение, проверка работы в браузере и запуск тестов для проверки внесенных изменений.

Резюме

- 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.
- Для организации подобных фоновых процессов, задач можно использовать несколько вариантов решения.

CyberBionic systematics

CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

- 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.
- 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 тп
- 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 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 получение курсов валют, кажется нецелесообразным. 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.
- 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.
- 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.
- Создадим простейший модуль tasks.py с задачей, которая будет периодически выводить на печать текущее время и запустим шедулер.

CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com

- 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.
- И пока скрипт будет работать, внесем небольшие изменения в web приложение. Хотелось бы добавить логирование в файл поступающих запросов-ответов в базовом контроллере, а также добавить отображение 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.
- Проверим внесенные изменения в браузере, а затем посмотрим логи пакета api и убедимся, что наша задача с обновлением курсов запускалась, пока мы вносили изменения во Flask приложение.
- Наш проект готов к деплою — размещению в интернете. Это и ждет нас на последнем уроке курса Python Practice.

Закрепление материала

- В чем отличие процессов, происходящих при обработке HTTP запроса и фоновых задач?
- Какие есть основные варианы для организации периодически выполняемых задач? Какие плюсы и минусы у каждого подхода?
- Какова особенность библиотеки APScheduler по сравнению с очередями задач?
- Как организовать простейшую периодически запускаемую задачу с помощью APScheduler библиотеки?

Дополнительное задание

Задание

Добавить сохранение ошибки в таблицу error_logs при обработке ее в базовом контроллере.

Самостоятельная деятельность учащегося

Задание 1
Добавить логирование в файл при работе модуля с задачами. Убедиться, что оно работает.

Задание 2
Переписать условия запуска задачи с использованием триггера типа cron. Для корректного указания cron выражения можно воспользоваться ресурсом https://en.wikipedia.org/wiki/Cron#CRON_expression

Рекомендуемые ресурсы

https://www.fullstackpython.com/task-queues.html
https://habr.com/ru/company/biggo/blog/102742/
https://devcenter.heroku.com/articles/clock-processes-python
https://apscheduler.readthedocs.io/en/3.0/

CyberBionic systematics
CyberBionic Systematics ® 2019
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 750 312
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com