Урок WEB. HTTP

Работа с протоколом HTTP

Аннотация

В уроке продолжается рассказ про использование HTTP-API. Показан пример поискового приложения, формирующего запросы к HTTP-API на основании результатов других запросов. Рассматривается API поиска по организациям с примерами задач.

1. Повторение

На одном из прошлых уроков мы начали разговор о разных видах API. Практическую часть мы посвятили использованию HTTP-API на примере Yandex.Maps.StaticAPI и Yandex.Maps.Geocoder. Давайте вспомним, как выглядит это взаимодействие с точки зрения Python-приложения.

2. Задание параметров url

В наших примерах запрос к API был представлен константной строкой, поскольку этого достаточно для первого знакомства с возможностями API. В более сложных задачах запрос приходится формировать в ходе выполнения программы, основываясь на данных, введенных пользователем или полученных иным образом. Делать это путем конкатенации строк или каким-то подобным образом крайне неудобно (это будет работать, но мы настоятельно не рекомендуем так делать) и порождает километровые строки в коде программы. К тому же надо решать вопросы с кодированием различных специальных символов. Про кодирование url-запросов (замены пробелов на «+», %-кодирование) можно прочитать **в Википедии**.

Разумеется, есть и более удобный способ. Для удобства формирования запросов в функциях `requests.request()` и `requests.get()` есть именованный параметр `params`, представляющий собой словарь (dict). В нем содержится отображение из названий параметров в их значения. Давайте рассмотрим пример из предыдущей лекции:

```
map_request = "http://static-maps.yandex.ru/1.x/?ll=37.530887,55.703118&spn=0.002,0.002&l=map"
response = requests.get(map_request)
```

Этот код можно переписать вот так:

```
import requests
api_server = "http://static-maps.yandex.ru/1.x/"

lon = "37.530887"
lat = "55.703118"
delta = "0.002"

params = {
    "ll": ",".join([lon, lat]),
    "spn": ",".join([delta, delta]),
    "l": "map"
}
response = requests.get(api_server, params=params)
```

Результаты выполнения двух вариантов кода будут идентичными. Понятно, что первый вариант использовать проще и быстрее, если нам нужно выполнить всего один константный запрос. Второй же вариант предпочтительнее, когда надо формировать запрос на лету. Например, если координаты нам ввел пользователь или мы получили их, например, в ответе геокодера.

3. Поисковое приложение

Давайте напишем простое поисковое приложение. Пользователь печатает в командной строке запрос, а наша задача состоит в том, чтобы найти координаты запрошенного объекта и показать его на карте, выбрав соответствующий масштаб и позицию карты.

Часть кода, которая выполняет запросы к API, может выглядеть так:

```
import sys
from io import BytesIO
# Этот класс поможет нам сделать картинку из потока байт

import requests
from PIL import Image

# Пусть наше приложение предполагает запуск:
# python search.py Москва, ул. Ак. Королева, 12
# Тогда запрос к геокодеру формируется следующим образом:
toponym_to_find = " ".join(sys.argv[1:])

geocoder_api_server = "http://geocode-maps.yandex.ru/1.x/"

geocoder_params = {
    "apikey": "40d1649f-0493-4b70-98ba-98533de7710b",
    "geocode": toponym_to_find,
    "format": "json"}
```

```
response = requests.get(geocoder_api_server, params=geocoder_params)

if not response:
    # обработка ошибочной ситуации
    pass

# Преобразуем ответ в json-объект
json_response = response.json()
# Получаем первый топоним из ответа геокодера.
toponym = json_response["response"]["GeoObjectCollection"][
    "featureMember"][0]["GeoObject"]
# Координаты центра топонима:
toponym_coodrinates = toponym["Point"]["pos"]
# Долгота и широта:
toponym_longitude, toponym_lattitude = toponym_coodrinates.split(" ")

delta = "0.005"

# Собираем параметры для запроса к StaticMapsAPI:
map_params = {
    "ll": ",".join([toponym_longitude, toponym_lattitude]),
    "spn": ",".join([delta, delta]),
    "l": "map"
}

map_api_server = "http://static-maps.yandex.ru/1.x/"
# ... и выполняем запрос
response = requests.get(map_api_server, params=map_params)

Image.open(BytesIO(
    response.content)).show()
# Создадим картинку
# и тут же ее покажем встроенным просмотрщиком операционной системы
```

4. API Поиска по организациям

Мы рассмотрели две части Yandex.Maps.API из трех. Давайте познакомимся и с третьей, Поиском по организациям. Или Адресным справочником.

Страница API: **https://tech.yandex.ru/maps/geosearch/**

Прочтите описание/соглашение и найдите отличие от того, с чем нам приходилось работать раньше.

Для этого API нужен ключ. Для задач учебного курса ключ с необходимыми правами доступа получен заранее (dda3ddba-c9ea-4ead-9010-f43fbc15c6e3). Этот ключ позволяет выполнить лимитированное число запросов, поэтому пользоваться им разрешается только при решении задач этого курса. После курса ключ будет деактивирован, и запросы с ним перестанут работать. Если же для своих проектов вам понадобится доступ к API поиска по организациям, необходимо получить ключ самостоятельно. Для этого нужно заполнить форму, указав данные для связи, а также сайт, на котором предполагается использовать данное API. Мы говорили о том, что

по условиям результаты надо обязательно использовать на сайте — обратите на это внимание. Исключение сделано только для наших учебных проектов.

Ключ для сервиса можно бесплатно получить по адресу: **https://developer.tech.yandex.ru/services/**

Прочитайте о возможностях API поиска по организациям.

Формат диалога с Поиском по организациям похож на формат геокодера. Давайте найдем ближайшую аптеку к вашему дому. Будет похоже на написанную раньше программу поиска объекта по адресу.

Формат запроса описан на **странице**.

```
search_api_server = "https://search-maps.yandex.ru/v1/"
api_key = "..."

address_ll = "37.588392,55.734036"

search_params = {
    "apikey": api_key,
    "text": "аптека",
    "lang": "ru_RU",
    "ll": address_ll,
    "type": "biz"
}

response = requests.get(search_api_server, params=search_params)
if not response:
    #...
    pass
```

Формат ответа смотрите на странице: **https://tech.yandex.ru/maps/doc/geosearch/concepts/response_structure_business-docpage/**

Продолжим разбор примера:

```
# Преобразуем ответ в json-объект
json_response = response.json()

# Получаем первую найденную организацию.
organization = json_response["features"][0]
# Название организации.
org_name = organization["properties"]["CompanyMetaData"]["name"]
# Адрес организации.
org_address = organization["properties"]["CompanyMetaData"]["address"]

# Получаем координаты ответа.
point = organization["geometry"]["coordinates"]
org_point = "{0},{1}".format(point[0], point[1])
delta = "0.005"

# Собираем параметры для запроса к StaticMapsAPI:
map_params = {
```

```
    # позиционируем карту центром на наш исходный адрес
    "ll": address_ll,
    "spn": ",".join([delta, delta]),
    "l": "map",
    # добавим точку, чтобы указать найденную аптеку
    "pt": "{0},pm2dgl".format(org_point)
}

map_api_server = "http://static-maps.yandex.ru/1.x/"
# ... и выполняем запрос
response = requests.get(map_api_server, params=map_params)
```

5. Еще немного о возможностях requests

Библиотека requests прекрасно умеет работать не только со строками и числами, но и преобразовывать в аргументы запроса и более сложные типы, например, списки.

```
params = {'list_param': ['value1', 'value2']}
response = requests.get('адрес_сайта', params=params)
```

Списки преобразуются в запросе вот к такому виду:

```
адрес_сайта?list_param=value1&list_param=value2
```

Посмотреть получившийся запрос можно с помощью атрибута `url`:

```
print(response.url)
```

Кроме того, можно влиять на служебную информацию, которая передается вместе с запросом в заголовках запроса. На некоторых сервисах код доступа к API передается именно в заголовке, или можно притвориться каким-нибудь браузером, переопределив значение `user-agent` запроса. Подробнее про стандартные HTTP заголовки можно почитать **тут**.

Задать заголовки можно с помощью параметра `headers`, в который передается словарь из заголовков и их значений.

```
headers = {'user-agent': 'yandexlyceum/1.1.1'}
response = requests.get(url, headers=headers)
```

Это далеко не все возможности requests, про некоторые из оставшихся мы еще поговорим в курсе далее.