ТЕХНОАТОМ

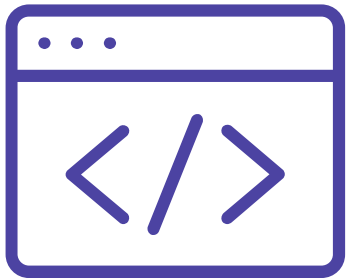
Веб-программирование Python

Лекция 3. Интернет и веб-приложения

Михалев Олег

Сегодня

- Сети и сетевые модели
- Протоколы глобальной сети
- Семейство протоколов HTTP
- Интерфейс WSGI



Сеть - совокупность технологий, обеспечивающих обмен данными между устройствами.

Ethernet
Wi-Fi (IEEE 802.11)

Сети мобильной связи

Локальные сети
Глобальная сеть Интернет

Для работы в сетях с неоднородными устройствами и программным обеспечением в глобальной сети необходимо согласованное взаимодействие, а значит единое представление данных и способов их обработки.

Модель OSI

ТЕХНОАТОМ

- Прикладной уровень
- Представительский уровень
- Сеансовый уровень
- Транспортный уровень
- Сетевой уровень
- Канальный уровень
- Физический уровень

ТЕХНОАТОМ

Прикладной уровень
Предоставляет интерфейсы приложениям

Представительский уровень
Преобразует данные

Сеансовый уровень
Поддерживает взаимодействие

ТЕХНОАТОМ

Транспортный уровень

Обеспечивает передачу данных

Сетевой уровень

Обеспечивает логическую адресацию

Поддерживает маршрутизацию

Канальный уровень
Обеспечивает физическую адресацию

Физический уровень
Обеспечивает доступ к среде передачи данных

Модель DOD и TCP/IP

ТЕХНОАТОМ

- Прикладной уровень
- Транспортный уровень
- Сетевой уровень
- Канальный уровень

- Физический уровень

Прикладной уровень TCP/IP

Прикладной уровень (HTTP)
Представительский уровень (SSL)
Сеансовый уровень (REST)

Транспортный уровень TCP/IP
Транспортный уровень (TCP, UDP)

Сетевой уровень TCP/IP
Сетевой уровень (IP)

Канальный уровень TCP/IP

Канальный уровень (Ethernet)

Физический уровень (драйвера оборудования)

Протокол IP стал в основу глобальной сети Интернет, объединив отдельные разрозненные компьютерные сети.

Протокол IP формирует пакеты содержащие адрес отправителя и адрес получателя.

IPv4-адрес состоит из 32 битов. При записи (в строковом представлении) используется десятичное представление и разделение октетов точкой.

```
1. 127.0.0.1
2. 188.93.56.94
```

IPv6-адрес состоит из 128 битов. При записи (в строковом представлении) используется шестнадцатеричное представление, группировка октетов парами и разделение двоеточием.

```
1. 0:0:0:0:0:0:0:1
2. 2001:0db8:11a3:09d7:1f34:8a2e:07a0:765d
```

TCP предоставляет доступ к передаче данных для приложения и обеспечивает доставку сообщений и их порядок.

Протокол TCP формирует пакеты содержащие порт приложения-отправителя, порт приложения-получателя.

UDP использует простую модель передачи и, в отличии от TCP, не гарантирует доставку и правильный порядок данных.

Сторона инициирующая взаимодействие - клиент.
Сервер отвечает на запросы клиента.

Клиент-серверная архитектура - наиболее распространенный способ организации высокоуровнего сетевого взаимодействия.

Клиент-сервер

ТЕХНОАТОМ

```
1. >>> from socket import socket
2. >>> server = socket()
3. >>> address = '127.0.0.1', 54670
4. >>> server.bind(address)
5. >>> server.listen(1)
6. >>> client, client_address = server.accept()
7. >>> client.recv(1024)
8. b'Hello'
9. >>> client.send(b'Hello')
10.>>> client.close()
11.>>> server.close()
```

Клиент-сервер

ТЕХНОАТОМ

```
1. >>> from socket import socket
2. >>> server = socket()
3. >>> address = '127.0.0.1', 54670
4. >>> client.connect(address)
5. >>> client.send(b'Hello')
6. 5
7. >>> client.recv(1024)
8. b'Hello'
9. >>> client.close()
```

Очевидно, запоминать адреса и обращаться к Интернет-ресурсам по ним неудобно.

Система DNS используется для получения адреса по имени.

```
1. >>> from socket import gethostname, gethostbyname
2. >>> hostname = gethostname()
3. >>> hostname
4. 'computer'
5. >>> gethostbyname(hostname)
6. '192.168.0.125'
7. >>> gethostbyname('atom.mail.ru')
8. '188.93.56.94'
```

База данных DNS распределена и поддерживается иерархией, домены структурированы по уровню и каждый уровень управляется соответствующими структурами.

```
1. atom.mail.ru.
2. mail.ru.
3. ru.
4. .
```

При обращении к серверу мы используем не только его имя.

URL

- Схема (протокол) + "://"
- Доменное имя
- Путь
- "?" + Запрос
- "#" + Фрагмент

Семейство протоколов HTTP

HTTP/0.9
HTTP/1.0
HTTP/1.1
HTTP/2

Запрос

Стартовая строка

Заголовки запроса

Тело сообщения

Ответ

Статусная строка

Заголовки ответа

Тело сообщения

Стартовая строка

- Метод
- URL
- Версия

```
1. GET /feed/?p=4 HTTP/1.1
```

Методы HTTP

GET
POST

HEAD
PUT
DELETE

Заголовки

- Имя + ":" + Значение

```
1. GET /feed/?p=4 HTTP/1.1
2. Host: atom.mail.ru
```

Заголовки HTTP

Общие заголовки
Заголовки запроса
Заголовки ответа
Заголовки сущности

Статусная строка

- Версия
- Код состояния
- Комментарий к состоянию

```
1. HTTP/1.1 200 OK
```

Первая цифра кода указывает на класс состояния.

1xx - Информационное уведомление

```
1. HTTP/1.1 101 Switching Protocols
```

2xx - Уведомление об успехе

```
1. HTTP/1.1 200 OK
```

3xx - Уведомление о перенаправлении

```
1. HTTP/1.1 302 Found
```

5xx - Уведомление об ошибке сервера

```
1. HTTP/1.1 500 Internal Server Error
```

4xx - Уведомление об ошибке клиента

```
1. HTTP/1.1 404 Not Found
```

Протокол HTTP/2

Бинарный протокол
Сжатие
Приоритезация
Мультиплексирование

Как подружить веб-среду с Python?

WSGI - интерфейс между веб-сервером и приложением на Python.

Первая версия
https://www.python.org/dev/peps/pep-0333/

Актуальная версия
https://www.python.org/dev/peps/pep-3333/

Простое WSGI-приложение:

```
>>> def simple_wsgi_application(environment, start_response):
...     http_headers = [
...         ('Content-type', 'text/plain; charset=utf-8')
...     ]
...     if environment.get('PATH_INFO', '/') == '/':
...         start_response('200 OK', http_headers)
...         yield 'Hello, Word!'.encode('utf-8')
...     else:
...         start_response('404 Not Found', http_headers)
...
```

Должно быть вызываемым объектом
Должно принимать контекст (окружение) запроса
Должно принимать обработчик запроса
Должно вызывать обработчик запроса, сообщая статус и заголовки
Может возвращать итерируемый объект с телом ответа

Переменные окружения

REQUEST_METHOD	HTTP-метод
SERVER_NAME	Имя сервера
SERVER_PORT	Порт сервера
SERVER_PROTOCOL	Протокол сервера

Переменные окружения

SCRIPT_NAME	Имя исполняемого скрипта
PATH_INFO	Путь из URL
QUERY_STRING	Запрос из URL
CONTENT_TYPE	Тип содержимого (HTTP-заголовок)
CONTENT_LENGTH	Размер содержимого (HTTP-заголовок)
HTTP_*	HTTP-заголовки запроса

Обработчик запроса принимает состояние ответа, заголовки и, возможно, информацию об ошибках.

```
1. >>> def start_response(status, headers, exc_info=None):
2. ...     ...
3. ...
4. >>> exc_info = exception_type, exception, traceback
```

```
1. class SimpleWSGIApplication:

2.     def __init__(self, environment, start_response):
3.         print('Get request')
4.         self.environment = environment
5.         self.start_response = start_response
6.         self.headers = [
7.             ('Content-type', 'text/plain; charset=utf-8')
8.         ]

        ...
```

```
    ...
1.  def __iter__(self):
2.      print('Wait for response')
3.      if self.environment.get('PATH_INFO', '/') == '/':
4.          yield from self.ok_response('Hello, World!')
5.      else:
6.          self.not_found_response()
7.      print('Done')
    ...
```

```
    ...
1.    def not_found_response(self):
2.        print('Create response')
3.        print('Send headers')
4.        self.start_response('404 Not Found', self.headers)
5.        print('Headers is sent')
    ...
```

```
    ...
1.      def ok_response(self, message):
2.          print('Create response')
3.          print('Send headers')
4.          self.start_response('200 OK', self.headers)
5.          print('Headers is sent')
6.          for i in range(5):
7.              print('Send part of body')
8.              yield ('%s₩n' % message).encode('utf-8')
9.          print('Body is sent')
```

Для запуска WSGI-приложений на данном этапе мы будем использовать стандартную библиотеку **wsgiref**.

```
>>> from wsgiref.simple_server import make_server
>>> if __name__ == '__main__':
...     server = make_server('127.0.0.1', 80, SimpleWSGIApplication)
...     server.serve_forever()
...
```

Спасибо за внимание!

Михалев Олег
mailto:mhalairt@gmail.com

Домашнее задание

Определен следующий бесконечный генератор:

```
1. from random import randint
2. from time import sleep

3. def events(max_delay, limit):
4.     while True:
5.         delay = randint(1, max_delay)
6.         if delay >= limit:
7.             sleep(limit)
8.             yield None
9.         else:
10.            sleep(delay)
11.            yield 'Event generated, awaiting %d s' % delay
```

Домашнее задание

ТЕХНОАТОМ

Генератор симулирует поступление событий в систему, на вход он получает максимальную задержку генерации max_delay для псевдослучайного времени ожидания и лимит времени ожидания. Если время ожидания превышает заданный лимит - генератор вернет значение None, что будет означать, что событий за интервал ожидания не произошло.

Домашнее задание

ТЕХНОАТОМ

Необходимо проинициализировать генератор (с произвольными параметрами) в глобальную переменную и определить класс WSGI-приложения, возвращающий события генератора.

При этом в случае успеха (генератор вернул не **None**) приложение должно возвращать стутус **200 OK**, а в противном случае статус **204 No Content**.